



## Modulul 4

Scrierea și citirea fișierelor

## Overview

- ▶ Administrarea sistemului de fișiere
- ▶ Citirea și scrierea folosind stream-uri
- ▶ Sporirea securității unei aplicații folosind Isolated Storage



Aproape toate aplicații au nevoie de o formă de input și output și, în multe cazuri, aceste operații se fac pe fișiere. Platforma .NET 2.0 oferă un set extins de clase ce permit accesarea sistemului de fișiere, navigarea prin foldere, crearea și ștergerea de fișiere și citirea și scriere de fișiere

Platforma .NET implementează un model de streaming pentru a accesa datele ce sunt ținute într-un fișier. Sunt posibile de asemenea operații de compresie și criptare asupra datelor dintr-un fișier. De asemenea, datele se pot stoca temporar în memorie. Totuși, stocarea datelor în memorie are anumite limitări, de aceea platforma .NET oferă posibilitatea de a stoca datele într-un loc izolat pe disc, protejând datele private ale aplicației de programe malițioase.

Acest modul descrie modul de folosire a claselor din platforma .NET ce permit manipularea fișierelor și a folderelor, precum și modul de citire și scriere a fișierelor în memorie.

### Obiective

La finalul acestui modul, veți putea să:

- Accesați fișiere și directorare folosind clasele oferite de platforma .NET
- folosiți streamuri pentru a citi și scrie date ce sunt stocate în fișiere sau în memorie.
- creați și administrați locații izolate pentru stocarea datelor unei aplicații.

## Administrarea sistemului de fișiere

- ▶ Administrarea fișierelor folosind clasele **File** și **FileInfo**
- ▶ Citirea și scrierea unui fișier folosind clasa **File**
- ▶ Administrarea folderelor folosind clasele **Directory** și **DirectoryInfo**
- ▶ Administrarea dispozitivelor folosind clasa **DriveInfo**
- ▶ Obținerea de informații despre cale pentru fișiere și directoare
- ▶ Monitorizarea modificărilor aduse fișierelor și directoarelor


Majoritatea aplicațiilor rețin date în fișiere și efectuează operații de intrare/ieșire pentru a le citi și scrie. Fișierele sunt stocate în sistemul de fișiere al calculatorului sau în folderele pe care alte calculatoare le-au partajat. Sistemul de fișiere oferit de sistemul de operare Windows stochează fișierele într-un mod ierarhizat în foldere. Folderele conțin o colecție de fișiere și subfoldere. Librăria de clase a platformei .NET 2.0 conține un set de clase în namespace-ul **System.IO** ce pot fi folosite pentru a interoga și manipula fișiere și directoare din sistemul de fișiere. Această lecție vă introduce în aceste clase.

Câteodată este util să monitorizați un fișier sau un folder pentru a observa modificările aduse acestuia și pentru a executa anumite acțiuni când apar modificări. Platforma .NET 2.0 oferă clasa **FileSystemWatcher** ce vă permite să implementați aceste funcționalități. Această lecție descrie modul de folosire a acestei clase pentru a prinde evenimentele ce sunt lansate în momentul în care un fișier sau un folder este creat, șters sau modificat.

### Obiective

La finalul acestei lecții, veți putea să:

- folosiți clasele **File** și **FileInfo** pentru a crea, copia, șterge și muta fișiere, precum și pentru a citi și modifica proprietățile acestora
- folosiți clasa **File** pentru a citi și scrie dintr-un fișier
- folosiți clasele **Directory** și **DirectoryInfo** pentru a crea și șterge foldere, citi proprietățile lor și pentru a obține conținutul acestora.
- folosiți clasa **DriveInfo** pentru a obține informații despre o unitate de stocare
- folosiți clasa **Path** pentru a obține informații despre calea unui fișier sau folder.
- folosiți clasa **FileSystemWatcher** pentru a monitoriza schimbările aduse fișierelor sau folderelor

 itacad  
you@technology  
  
Microsoft  
.NET

## Administrarea fișierelor

### •Clasa File

- ▶ Oferă metode statice
- ▶ Primește numele fișierului ca parametru
- ▶ Oferă metode ce citesc și setează attribute ale fișierului
- ▶ Verifică drepturile utilizatorului asupra fișierului la apelul metodelor
- ▶ Este folosită pentru a executa o singură operație asupra unui fișier

### •Clasa FileInfo

- ▶ Va trebui instanțiată pentru a o folosi
- ▶ Primește numele fișierului ca parametru al constructorului
- ▶ Oferă metode ce citesc și setează attribute ale fișierului
- ▶ Verifică drepturile utilizatorului asupra fișierului
- ▶ Este folosită pentru a executa o serie de operații asupra unui fișier

• Create

• Copy/CopyTo

• Encrypt

• Exists

• Delete

• Move/MoveTo

• Decrypt

• Replace

Academia  
Microsoft

Clasele **File** și **FileInfo** se găsesc în namespace-ul **System.IO** și oferă funcționalitate pentru a manipula fișiere și pentru a examina proprietățile lor. Clasa **File** este o clasă ce oferă metode statice, fiecare dintre acestea primind ca parametru calea completă a unui fișier. Folosiți această clasă pentru a executa operații individuale asupra unui fișier. De fiecare dată când invocați o metodă a clasei **File**, sistemul de fișiere verifică calea către fișier și verifică dacă utilizatorul ce rulează aplicația are drepturi de acces asupra fișierului.

Funcționalitatea clasei **FileInfo** este asemănătoare celei a clasei **File**, diferența fiind că **FileInfo** folosește o instanță și nu metode statice pentru a executa operații asupra fișierelor. Pentru a instanția clasa **FileInfo** va trebui să oferiți numele unui fișier. Clasa **FileInfo** oferă mecanisme eficiente în momentul în care este nevoie să executați o serie de operații asupra aceluiași fișier. Totuși, ca și clasa **File**, clasa **FileInfo** verifică drepturile utilizatorului la fiecare metodă apelată.

4

## Folosirea clasei **File**

Următorul tabel descrie câteva dintre metodele uzuale ale clasei **File**.

Metoda	Descriere
<b>Create</b>	Crează un fișier specificând numele acestuia ca parametru. Directorul unde va fi pus fișierul trebuie să existe deja
<b>Copy</b>	Această metodă copiază un fișier. Numele fișierului sursă și destinație sunt specificați ca parametri. Calea către fișierul destinație trebuie să existe. Opțional, metoda poate suprascrie un fișier existent dacă acesta are același nume cu fișierul destinație
<b>Delete</b>	Această metodă șterge un fișier al cărui nume este primit ca parametru.
<b>Exists</b>	Metoda determină dacă un fișier cu numele specificat există. Întoarce o valoare booleană
<b>Move</b>	Această metodă mută sau redenumeste un fișier. Numele fișierului sursă și destinație sunt specificate ca parametri. Această metodă nu poate fi folosită pentru a suprascrie un fișier existent.
<b>Replace</b>	Această metodă înlocuiește conținutul unui fișier cu cel ce se găsește în fișierul sursă. Opțional, datele ce se aflau în fișierul destinație pot fi salvate într-un al treilea fișier. Fișierul sursă este șters.
<b>Encrypt</b>	Această metodă criptează conținutul unui fișier specificat ca parametru
<b>Decrypt</b>	Decriptează conținutul fișierului specificat ca parametru. Doar utilizatorul care a criptat fișierul poate să îl decripteze
<b>GetAttributes</b>	Întoarce atributele fișierului ce a fost specificat ca parametru. Valoarea întoarsă este de tipul <b>FileAttribute</b> ; valoarea conține un set de flaguri ce indică, de exemplu, dacă fișierul este arhivat, comprimat, criptat, read-only sau ascuns
<b>SetAttributes</b>	Această metodă setează atribute unui fișier. Numele fișierului și valoarea de tipul <b>FileAttributes</b> sunt trimiși ca parametru

Următorul exemplu de cod folosește metoda **Copy** pentru a crea o copie a unui fișier. Codul suprascrive un fișier existent și apoi setează proprietățile de **Read-only** și **Hidden** pe noul fișier copiat. Acest lucru se face executând operații pe biți specificând flag-urile de **ReadOnly** și **Hidden**

```
using System.IO;

...

File.Copy(@"E:\Democode\MyFile.txt", @"E:\Democode\CopyFile.txt",
true);

File.SetAttributes(@"E:\Democode\CopyFile.txt",
    FileAttributes.ReadOnly | FileAttributes.Hidden);
```

Clasa **File** de asemenea oferă metode pentru citirea și scrierea datei la care fișierul a fost creat, accesat sau scris ultima dată.

### Clasa **FileInfo**

Clasa **FileInfo** oferă un set similar de funcții ca și **File**, deși unele sunt oferite ca proprietăți, nu ca funcții. În plus, clasa **FileInfo** oferă proprietăți ce permit izolarea anumitor elemente ale numelui fișierului, cum ar fi numele fără cale sau extensie. Următorul tabel conține cei mai importanți membri ai clasei **FileInfo**.

Membru	Descriere
<b>Create</b>	<b>Crează un fișier specificând numele acestuia ca parametru. Directorul unde va fi pus fișierul trebuie să existe deja</b>
<b>CopyTo</b>	<b>Această metodă crează o copie a fișierului ce este referențiat de obiectul FileInfo.</b> Numele fișierului destinație este specificat ca parametru al metodei. Directorul destinație trebuie să existe. Opțional poate suprascrive un fișier cu același nume cu fișierul destinație
<b>Delete</b>	<b>Această metodă șterge fișierul referențiat de obiectul FileInfo</b>
<b>MoveTo</b>	<b>Această metodă mută sau redenumeste fișierul referențiat de obiectul FileInfo.</b> Numele fișierului destinație este specificat ca parametru. Metoda nu poate suprascrive un fișier existent cu același nume ca fișierul destinație
<b>Replace</b>	<b>Această metodă înlocuiește conținutul unui fișier destinație cu cel aflat în fișierul referențiat de obiectul FileInfo.</b>

Membru	Descriere
<b>Encrypt</b>	<b>Această metodă criptează conținutul fișierului referențiat de obiectul FileInfo</b>
<b>Decrypt</b>	<b>Această metodă decriptează conținutul fișierului referențiat de obiectul FileInfo.</b> Doar utilizatorul care a criptat fișierul îl poate decripta
<b>Exists</b>	<b>Această metodă întoarce o valoare booleană dacă fișierul referențiat de obiectul FileInfo există sau nu</b>
<b>Attributes</b>	<b>Această proprietate întoarce sau setează attributele fișierului referențiat de obiectul FileInfo.</b> Attributele sunt reținute ca o valoare de tipul <b>FileAttribute</b>
<b>Name</b>	<b>Această proprietate întoarce numele fișierului referențiat de obiectul FileInfo, neincluzând calea pe disc</b>
<b>FullName</b>	<b>Această proprietate întoarce calea completă a fișierului referențiat de obiectul FileInfo.</b> Proprietatea este read-only
<b>Extension</b>	<b>Această proprietate întoarce extensia fișierului referențiat de obiectul FileInfo.</b> Proprietatea este read-only

### Folosirea clasei **FileInfo**

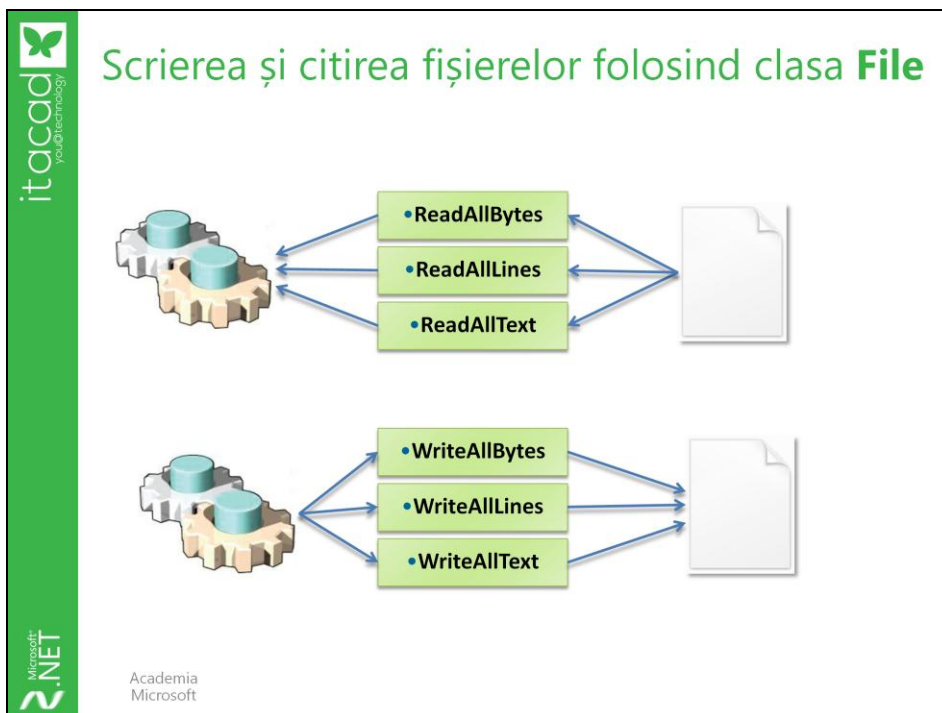
Pentru a utiliza aceste metode, trebuie întâi instanțiată clasa **FileInfo**, specificând calea către un fișier ca parametru constructorului. Toate operațiile ce sunt invocate apoi pe instanță se vor aplica acestui fișier. Pentru a putea compara cele două clase, următorul cod va executa aceleași operații, va copia fișierul și va seta oproprietățile de **ReadOnly** și **Hidden** pe copie

```
using System.IO;

...

FileInfo dataFileInfo = new FileInfo(@"E:\Democode\MyFile.txt");
dataFileInfo.CopyTo(@"E:\Democode\CopyFile.txt", true);

FileInfo copyFileInfo = new FileInfo(@"E:\Democode\CopyFile.txt");
copyFileInfo.Attributes = FileAttributes.ReadOnly |
FileAttributes.Hidden;
```



Clasa **File** oferă deasemenea o serie de operații ce pot fi folosite pentru a scrie și citi fișiere.

Metodă	Descriere
<b>AppendAllText</b>	Această metodă deschide sau crează un fișier și pune la finalul său textul ce este oferit ca parametru string
<b>ReadAllBytes</b>	Această metodă deschide un fișier binar, citește toate datele din el și le stochează într-un vector de octeți
<b>ReadAllLines</b>	Această metodă deschide un fișier, citește tot textul din acesta într-un vector de stringuri și închide fișierul
<b>ReadAllText</b>	Această metodă deschide un fișier text, citește tot textul într-un singur string și închide fișierul
<b>WriteAllBytes</b>	Această metodă crează un nou fișier sau suprascrie unul nou, scrie un vector de octeți și îl închide
<b>WriteAllLines</b>	Această metodă crează un nou fișier text sau suprascrie unul existent, scrie un vector de stringuri și îl închide
<b>WriteAllText</b>	Această metodă crează un nou fișier text sau suprascrie unul existent, scrie un singur string și îl închide



## Citirea dintr-un fișier

Următorul exemplu de cod demonstrează modul de citire a conținutului unui fișier binar într-un vector de valori byte folosind metoda **ReadAllBytes**

```
using System.IO;
...
Byte[] binaryData = null;
binaryData = File.ReadAllBytes(@"E:\Democode\MyData.bin");
```

## Scrierea într-un fișier

Următorul exemplu de cod demonstrează modul de scrierea a unui vector de string-uri într-un fișier text folosind metoda **WriteAllLines**. Dacă nu există, fișierul va fi creat. Clasa **Encoding** se găsește în namespace-ul **System.Text**

```
using System.Text;
using System.IO;
...
string [] data = new string[10];
data[0] = "This is line 1";
data[1] = "This is line 2";
data[2] = "This is line 3";
data[3] = "This is line 4";
data[4] = "This is line 5";
data[5] = "This is line 6";
data[6] = "This is line 7";
data[7] = "This is line 8";
data[8] = "This is line 9";
data[9] = "This is line 10";
File.WriteAllLines(@"E:\Democode\MyFile.txt", data,
Encoding.Unicode);
```

itacad

## Administrarea folderelor

### •Clasa Directory

- Oferă metode statice
- Primește numele directorului ca parametru
- Include metode ce nu sunt disponibile în clasa DirectoryInfo
- Este folosită pentru a executa o singură operație asupra unui director

### •Clasa DirectoryInfo

- Va trebui instanțiată pentru a o folosi
- Primește numele directorului ca parametru al constructorului
- Oferă metoda CreateSubDirectory ce nu este disponibilă folosind clasa Directory
- Este folosită pentru a executa o serie de operații asupra unui director

•CreateDirectory

•GetDirectories

•GetFiles

•Exists

•Delete

•Move/MoveTo

•GetFileSystemEntries/GetFileSystemInfo

Academia Microsoft

Platforma .NET 2.0 oferă clasele **Directory** și **DirectoryInfo** în namespace-ul **System.IO** pentru a întoarce și modifica informații legate de foldere. Aceste clase operează similar cu **File** și **FileInfo**. Clasa **Directory** oferă metode statice ce execută câte o singură operație asupra unui director. Clasa **DirectoryInfo** oferă un subset al funcțiilor din clasa **Directory**, dar metodele executate asupra instanței tind să fie mai eficiente în momentul în care este nevoie să se execute mai multe operații asupra aceluiași folder.

### Folosirea clasei **Directory**

Următorul tabel descrie metodele des folosite ale clase **Directory**

Metodă	Descriere
<b>CreateDirectory</b>	Această metodă crează un nou director folosind calea specificată ca parametru. Dacă este necesar, metoda va crea directoare intermediare ce sunt specificate în cale.
<b>Delete</b>	Această metodă șterge un director a cărui cale este specificată ca parametru. Va șterge de asemenea conținutul acestui folder
<b>Exists</b>	Această metodă determină dacă un folder a cărui cale a fost specificată există.

Metodă	Descriere
<b>GetCurrentDirectory</b>	Această metodă întoarce un string ce conține calea către folderul unde lucrează aplicația
<b>GetDirectoryRoot</b>	Această metodă întoarce calea directorului rădăcină pentru folderul specificat
<b>GetFiles</b>	Această metodă întoarce un vector de stringuri ce conține numele tuturor fișierelor din folderul specificat.
<b>GetFileSystemEntries</b>	Această metodă întoarce un vector de stringuri ce conține numele fișierelor și ale folderelor din directorul specificat.
<b>GetLogicalDrives</b>	Această metodă întoarce un vector de stringuri ce conține o listă a dispozitivelor de stocare de pe calculator
<b>GetParent</b>	Această metodă întoarce un obiect <b>DirectoryInfo</b> ce conține o referință la părintele folderului specificat
<b>Move</b>	Această metodă mută un folder și conținutul acestuia într-o locație nouă. Calea către directorul sursă și destinație sunt specificate ca parametri. Folderul destinație nu trebuie să existe.

Următorul exemplu de cod folosește metodele **GetCurrentDirectory** și **GetFileSystemEntries** pentru a obține și afișa conținutul folderului curent.

```
using System.IO;
...
string currentFolder = Directory.GetCurrentDirectory();
string[] fileAndFolderNames = Directory.GetFileSystemEntries(currentFolder);

foreach (string entry in fileAndFolderNames)
{
    Console.WriteLine("{0}", entry);
}
```

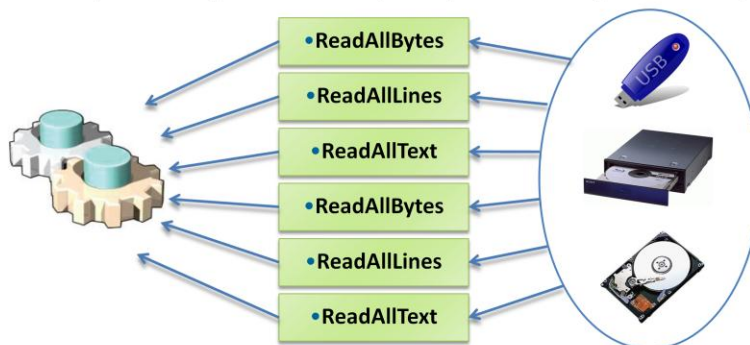
La fel ca și clasa **File**, clasa **Directory** oferă metode ce permit să citiți și modificați

data la care folderul a fost creat, accesat ultima dată sau scris.

## Administrarea dispozitivelor folosind clasa **DriveInfo**

### ► Clasa **DriveInfo**

- Oferă metode și proprietăți ce întorc informații despre dispozitive
- Suportă dispozitive fixe și dispozitive ce pot fi decuplate



Academia  
Microsoft

Clasa **DriveInfo** oferă metode și proprietăți ce pot fi folosite pentru a obține informații despre dispozitivele de stocare de pe un calculator. Puteți folosi aceasta clasă pentru a obține o listă de dispozitive și să aflați dimensiunea, spațiul liber și formatul fiecărui dispozitiv.

Următorul tabel prezintă cele mai des folosite metode și proprietăți ale clasei **DriveInfo**. Cu excepția proprietății **VolumeLabel**, toate informațiile sunt read-only

Membru	Descriere
<b>GetDrives</b>	Aceasta este o metodă statică ce întoarce un vector de obiecte <b>DriveInfo</b> ce sunt disponibile pe calculator
<b>IsReady</b>	Această proprietate returnează o valoare booleană ce indică dacă un dispozitiv de stocare este accesibil. Această proprietate este utilă în momentul în care doriți să determinați dacă un CD-ROM conține un CD etc.
<b>AvailableFreeSpace</b>	Această proprietate întoarce numărul de octeți liberi disponibili utilizatorului pe obiectul <b>DriveInfo</b> referențiat
<b>DriveFormat</b>	Această proprietate întoarce un string ce reprezintă sistemul de fișiere al dispozitivului referențiat de obiectul <b>DriveInfo</b>

Membru	Descriere
<b>DriveType</b>	Această proprietate întoarce tipul dispozitivului de stocare, ca o valoare a enumerației DriveType
<b>RootDirectory</b>	Această proprietate întoarce o referință la folderul radacină al dispozitivului de stocare referențiat de obiectul DriveInfo. Referința către folder este de tipul DirectoryInfo
<b>TotalFreeSpace</b>	Numărul total de octeți liberi pe respectivul dispozitiv
<b>TotalSize</b>	Numărul total de octeți pe respectivul dispozitiv
<b>VolumeLabel</b>	Această proprietate oferă acces la numele dat dispozitivului respectiv; acesta este un string

Folosirea clasei **DriveInfo**

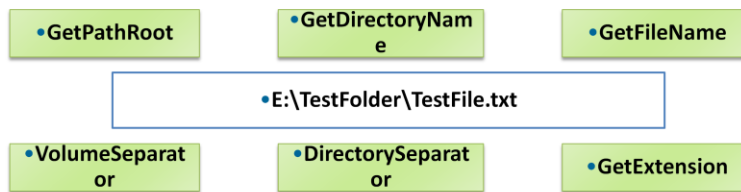
```
using System.IO;

...
DriveInfo[] drivesOnComputer = DriveInfo.GetDrives();
foreach (DriveInfo drive in drivesOnComputer)
{
    Console.WriteLine("Drive {0}", drive.Name);
    Console.WriteLine(" File type: {0}", drive.DriveType);
    if (drive.IsReady)
    {
        Console.WriteLine(" volume label: {0}", drive.VolumeLabel);
        Console.WriteLine(" File system: {0}", drive.DriveFormat);
        Console.WriteLine(" Total size of drive: {0} bytes ",
                           drive.TotalSize);
        Console.WriteLine(" Total available space: {0} bytes",
                           drive.TotalFreeSpace);
    }
    else
    {
        Console.WriteLine(" No media available");
    }
    Console.WriteLine();
}
```

## Obținerea de informații despre cale

### ► Clasa **Path**

- Oferă metode statice ce parsează numele fișierelor și ale directoarelor
- Mecanismele implementate țin cont de avantajele și limitările sistemelor de fișiere
- Conține metode ce generează nume unice de fișiere și fișiere temporare



Academia  
Microsoft

Clasa **Path** este o altă clasă ajutătoare ce oferă o serie de membrii statici ce pot fi folosiți pentru a parsa numele fișierelor și ale folderelor. Puteți folosi clasa **Path** pentru a extrage elemente din numele unui fișier, cum ar fi extensia sau folderul ce conține fișierul. În plus, membrii clasei **Path** țin cont de tipul sistemului de fișiere ce este referențiat de numele fișierului și de sistemul de operare. Membrii vor ține cont de asemenea de orice restricție impusă de sistemul de fișiere, cum ar fi dimensiunea maximă a numelui unui fișier. Dacă folosiți clasa **Path** pentru a examina numele unui fișier, este garantat că se va executa corect codul indiferent de sistemul de operare folosit.

Clasa **Path** conține o serie de metode utile ce pot fi folosite pentru a genera nume de fișiere temporare sau nume de fișiere ce este garantat că sunt unice pe orice calculator. Folosiți aceste metode când o aplicație trebuie să stocheze date într-un fișier temporar, nu se recomandă generarea proprie de nume de fișiere.

Următorul tabel prezintă membrii cei mai des folosiți ai clasei **Path**

Membru	Descriere
<b>DirectorySeparatorChar</b>	Acest câmp conține caracterul folosit de sistemul de operare pentru a separa numerele folderelor într-o structură ierarhică. Pe un sistem Windows, acesta este \
<b>VolumeSeparatorChar</b>	Acest câmp conține caracterul folosit de sistemul de operare pentru a separa numele dispozitivului de stocare (ex: C etc.) de folderele și fișierele din aceasta. Pe un sistem Windows, acesta este :
<b>PathSeparatorChar</b>	Acest câmp conține caracterul folosit de sistemul de operare pentru a separa nume de căi în variabile de mediu. Pe un sistem Windows, aceasta este ;
<b>ChangeExtension</b>	Această metodă primește un nume de fișier și o extensie ca parametriși întoarce un nou nume de fișier, extensia fișierului original fiind înlocuită cu extensia oferită ca parametru
<b>GetDirectoryName</b>	Această metodă primește un nume de fișier ca parametru și întoarce partea de cale din numele fișierului. Dacă numele oferit nu conține o cale, metoda întoarce un string gol.
<b>GetExtension</b>	Metoda primește numele unui fișier ca parametru și întoarce extensia sa
<b>GetFileName</b>	Metoda primește ca parametru numele complet al unui fișier (inclusiv calea) și întoarce numele fișierului fără calea sa
<b>GetFileNameWithoutExtension</b>	Metoda primește ca parametru numele complet al unui fișier (inclusiv calea) și întoarce numele fișierului fără cale sau extensie
<b>GetPathRoot</b>	Această metodă primește un nume de fișier ca parametru și întoarce folderul rădăcină
<b>HasExtension</b>	Această metodă întoarce o valoare booleană, dacă fișierul oferit ca parametru are sau nu extensie
<b>IsPathRooted</b>	Această metodă întoarce o valoare booleană, dacă fișierul oferit ca parametru are cale absolută
<b>GetTempFileName</b>	Această metodă crează un fișier nou, gol, ce are un nume unic, în folder-ul temporar al utilizatorului și întoarce numele complet al acestui fișier.



## Folosirea clasei **Path**

Următorul exemplu de cod folosește clasa **Path** și afișază rezultatele. Codul demonstrează de asemenea modul de generare a fișiere temporare.

```
using System.IO;

...

FileInfo testInfo = new
FileInfo(@"E:\Democode\TestFolder\TestFile.txt");

Console.WriteLine("Test File Name: {0}", testInfo.FullName);
Console.WriteLine("Folder Name: {0}",
Path.GetDirectoryName(testInfo.FullName));
Console.WriteLine("File Extension: {0}",
Path.GetExtension(testInfo.FullName));
Console.WriteLine("File Name: {0}",
Path.GetFileName(testInfo.FullName));
Console.WriteLine("File Name without Extension: {0}",
    Path.GetFileNameWithoutExtension(testInfo.FullName));
Console.WriteLine("Full Path: {0}",
Path.GetFullPath(testInfo.FullName));
Console.WriteLine("Path Root: {0}",
Path.GetPathRoot(testInfo.FullName));
Console.WriteLine("Has Extension: {0}",
Path.HasExtension(testInfo.FullName));

Console.WriteLine("File Name with New Extension: {0}",
    Path.ChangeExtension(testInfo.FullName, "bin"));

Console.WriteLine("Temporary Folder: {0}", Path.GetTempPath());
Console.WriteLine("Created Temporary File: {0}",
Path.GetTempFileName());
Console.WriteLine("Random Full File Name: {0}",
    Path.Combine(Path.GetRandomFileName(),
Path.GetRandomFileName()));
```

## Monitorizarea modificărilor aduse fișierelor și directoarelor

- ▶ Monitorizează un director și lansează evenimente când fișierele și directoarele incluse sunt modificate
  - ▶ Creați un instanță a **FileSystemWatcher**
  - ▶ Setări proprietatea **Path** pe folderul ce conține fișierele ce trebuie monitorizate
  - ▶ Setări proprietățile **Filter** și **NotifyFilter** pentru a specifica ce modificări vor fi luate în considerare
  - ▶ Opțional, setări proprietatea **IncludeSubDirectory**
  - ▶ Pot fi monitorizate evenimentele **Created, Changed, Deleted, Renamed**
  - ▶ Setări proprietatea **EnableRaisingEvents** pe **True**

Academia  
Microsoft

Există posibilitatea ca o aplicație să trebuiască să monitorizeze un folder pentru orice schimbări ce sunt făcute fișierelor și folderelor sau dacă fișiere sunt adăugate sau șterse din folder. Platforma .NET 2.0 oferă aceste facilități prin clasa **FileSystemWatcher**.

### Clasa **FileSystemWatcher**

Pe slide sunt prezentate evenimentele pe care clasa **FileSystemWatcher** le oferă. Aceste evenimente pot fi lansate de fiecare dată când o modificare are loc într-un fișier sau folder. Observați că este posibil ca un este posibil ca mai multe evenimente de să fie lansate de aceeași operație. De exemplu, atât evenimentul **Created** cât și **Deleted** au loc la mutarea unui fișier.

Evenimentele **Changed, Created** și **Deleted** sunt tratate folosind delegatul **FileSystemEventHandler**. Acest delegat primește doi parametri, un obiect ce reprezintă sursa evenimentului și un obiect de tipul **FileSystemEventArgs** ce conține informații despre eveniment, cum ar fi calea fișierului ce este afectat și tipul modificării ce are loc. Evenimentul **Renamed** folosește delegatul **RenamedEventArgs**. Acest delegat primește de asemenea doi parametri, un obiect sursă și un obiect **RenamedEventArgs** ce conține informații cum ar fi numele vechi și numele nou al fișierului sau folderului ce a fost redenumit.

Trebuie să specificați ce folder să fie urmărit de obiectul **FileSystemWatcher** setând proprietatea **Path**. Implicit, acest obiect va urmări toate modificările aduse tuturor fișierelor din folder, dar nu și în subfoldere. Totuși, modificând proprietățile **Filter**, **IncludeSubdirectories** și **NotifyFilter** puteți schimba acest comportament.

### Folosirea clasei **FileSystemWatcher**

Pentru a folosi clasa **FileSystemWatcher**, executați următorii pași:

1. Instanțiați clasa
2. Setati proprietatea **Path** pentru a referenția folderul unde se găsesc fișierele și folderele ce vor fi monitorizate
3. Setati valori pentru proprietățile **Filter** și **NotifyFilter** dacă nu doriți să monitorizați toate tipurile de modificări aduse tuturor fișierelor și directoarelor.
4. Opțional, setati proprietatea **IncludeSubdirectories** la valoarea **True** dacă doriți să monitorizați subfoldere
5. Adăugați metode de tratare a evenimentelor **Changed**, **Created**, **Deleted** și **Renamed** și înregistrați aceste metode obiectului **FileSystemWatcher** ca metode de tratare a evenimentelor.
6. Setati proprietatea **EnableRaisingEvents** pe **True**

Un obiect de tipul **FileSystemWatcher** va face buffering pentru informațiile despre evenimente intern până când acestea vor fi tratate, în cazul în care operațiile au loc mai rapid decât poate evenimentul corespunător să fie lansat. Acest buffer are o dimensiune finită dar ajustabilă, folosind proprietatea **InternalBufferSize**.

## Citirea și scrierea folosind stream-uri

- Ce este un stream I/O
- Scrierea și citirea fișierelor folosind clasa **FileStream**
- Scrierea și citirea datelor text cu un stream
- Scrierea și citirea datelor binare cu un stream
- Scrierea și citirea datelor temporare în memorie
- Adăugarea unui buffer unui stream *unbuffered*

Clasa **File** oferă metode statice ce execută operații de citire și scriere de bază pe fișiere. Totuși, aplicațiile este posibil să aibe nevoie de a accesa datele într-o manieră mai selectivă decât oferă această clasă. Platforma .NET 2.0 implementează un model de streaming ce poate fi folosit pentru a accesa datele dintr-un fișier; o aplicație poate citi datele dintr-un fișier deschizând un stream ce este bazat pe respectivul fișier și apoi citind date din stream. Dacă stream-ul suporta acces random, aplicația poate găsi datele pe care le caută în fișier căutând locația datelor și apoi citind un număr de biți.

Această lecție prezintă modul de folosire a stream-urilor pentru citirea și scrierea datelor în fișiere binare și fișiere text, precum și modul de execuție a accesului random când un stream suportă acest mod de operare. Se vor discuta de asemenea modurile de stocare a datelor ce se găsesc în memorie.

### Obiective

La finalul acestei lecții, veți putea să:

- descrieți modelul de streaming ce este implementat în platforma .NET 2.0
- citiți și scrieți date într-un fișier folosind clasa **FileStream**
- citiți și scrieți date într-un fișier folosind clasele **StreamReader** și **StreamWriter**
- citiți și scrieți tipuri de date primitive folosind clasele **BinaryRead** și **BinaryWrite**
- stocați și să recuperați date ce au fost puse temporar în memorie.

- descrieți modul de folosire a clasei **BufferedStream** pentru a adăuga buffering unui stream ce nu suportă acest lucru.

## Ce este un stream I/O

- ▶ Un flux de date de la o sursă la o destinație ce circulă printr-un canal
- ▶ Datele ajung în ordinea în care au fost trimise
- ▶ Clasa **Stream**
  - ▶ Clasa de bază pentru toate stream-urile în platforma .NET
  - ▶ Implementează metode și proprietăți ce administrează un stream ca o serie de octeți
  - ▶ Oferă acces pentru acces random, nu doar secvențial

•Read	•ReadByte	•Write	•WriteByte
•Seek	•CanRead	•CanWrite	•CanSeek
•Flush	•Close	•Length	•Position

Academia  
Microsoft

Un stream I/O este un flux de date de la o sursă la o destinație. Toate datele din stream circulă în ordine printr-un singur canal și ajung la destinație în ordinea în care au fost trimise. Această procedură este foarte eficientă în momentul în care sunt necesare operații I/O. Acest lucru se datorează în primul rând faptului că fișierele pot fi foarte mari ca dimensiune sau cu lungime variabilă.

### Clasa **Stream**

Platforma .NET oferă câteva clase ce se găsesc în namespace-ul **System.IO** ce pot fi folosite pentru a citi și scrie streamuri. La cel mai înalt nivel de abstractizare este clasa **Stream**, ce definește funcționalități ce sunt oferite de toate streamurile. Această clasă oferă accesul la o secvență de biți dintr-un fișier, precum și la operații și proprietăți pe care toate streamurile le oferă. Intern, un obiect de tipul **Stream** menține un pointer ce identifică locația curentă în sursa de date. În momentul instanțierii clasei **Stream**, pointerul este pus înaintea primului byte din sursă. Pe măsură ce citiți și scrieți date, clasa **Stream** avansează acest pointer către sfârșitul datelor.

Următorul tabel prezintă cele mai des folosite metode și proprietăți ale clasei **Stream**

Membru	Descriere
<b>Read</b>	Această metodă citește o secvență de octeți dintr-un stream într-un vector de octeți și întoarce un întreg reprezentând numărul de octeți citiți.
<b>ReadByte</b>	Această metodă citește un singur octet dintr-un stream într-o variabilă întreagă și avansează pointerul cu 1. se întoarce -1 dacă nu s-a putut citi
<b>Seek</b>	Această metodă avansează pointer-ul la poziția specificată în stream. Poziția este specificată printr-un offset și o poziție de start.
<b>Write</b>	Această metodă scrie conținutul unui vector de octeți într-un stream și avansează poziția pointerului
<b>WriteByte</b>	Această metodă scrie un singur octet într-un stream și avansează poziția pointerului cu 1
<b>Flush</b>	Un stream poate oferi buffering pentru date pentru a optimiza anumite operații. Această metodă golește memoria buffer și forțează ca toate datele să fie scrise
<b>Close</b>	Această metodă execută întâi Flush și apoi închide streamul, eliberând resursele asociate stream-ului
<b>CanRead, CanSeek, CanWrite</b>	Proprietăți ce întorc valori boolene dacă respectivul stream permite citirea, mutarea pointerului sau respectiv scrierea
<b>Length</b>	Proprietate ce întoarce un întreg long ce indică lungimea totală a streamului

Clase ce fac streaming în platforma .NET

Clasa **Stream** nu poate fi folosită direct. În schimb, sunt folosite clase specializate ce sunt derivate din aceasta, clase ce sunt optimizate pentru anumite operații de intrare/ieșire. De exemplu, clasa **FileStream** implementează un stream ce folosește un fișier de pe disk drept sursă. Clasa **MemoryStream** implementează un stream ce folosește un bloc de memorie ca sursă.

Tineți minte că un stream este o secvență de octeți complet neprocesată. Dacă un fișier are o structură internă, va trebui să parsați acest stream pentru a o recrea. Deși programatorul poate face aceste operații, acest lucru durează mult și este destul de susceptibil la apariția de erori de programare. De aceea, sunt puse la dispoziție clase


## StreamReader, StreamWriter, BinaryReader şı BinaryWriter



itacad  
you@technology  
  
Microsoft  
.NET

## Scrierea și citirea unui fișier folosind FileStream

- ▶ **FileStream**
  - ▶ Este o implementare a clasei **Stream** pentru scrierea și citirea fișierelor
  - ▶ Specifică fișierul, modul de deschidere, tipul de acces și modul de sharing al fișierului în constructor



Clasa **FileStream** implementează metodele și proprietățile clasei **Stream** pentru a accesa date ce au ca sursă un fișier.

### Clasa **FileStream**

La construcția unui obiect **FileStream**, trebuie să specificați calea către fișier împreună cu alte opțiuni. Acești parametri specifică modul de deschidere a unui fișier, modul de acces la fișier, dimensiunea buffer-ului, dacă operațiile asupra fișierului se vor executa sincron sau asincron etc.

Implicit, la crearea unui obiect **FileStrem**, aveți acces exclusiv asupra unui fișier. Totuși, specificând un mod de partajare, un fișier poate fi deschis și în alt proces pentru a se putea lucra cu el. Dacă la un moment se dorește acces exclusiv pe fișier, se pune la dispoziție metoda **Lock** pentru a preveni accesul altor procese.

## Folosirea clasei **FileStream**

Următorul cod demonstrează modul de folosire al clasei **FileStream** pentru a deschide un fișier numit Greetings.txt și pentru a scrie două mesaje scurte, "Hello" și "World", în fișier. Codul citește înapoi acești octeți din fișier și îi afișază pe ecran. Obiectul **FileStream** va crea fișierul dacă acesta nu există, îl va deschide pentru citire și scriere și va specifica modul de partajare pentru a permite și altor obiecte **FileStream** să acceseze fișierul la un anumit moment. Pentru a nu permite altor procese să scrie date în momentul în care aplicația curentă face acest lucru, se vor folosi metodele **Lock** și **Unlock** pentru a administra accesul la fișier.

```
using System.IO;

...

byte[] firstMessage = new byte[] { Convert.ToByte('H'), Convert.ToByte('e'),
                                   Convert.ToByte('l'), Convert.ToByte('l'), Convert.ToByte('o') };
byte[] secondMessage = new byte[] { Convert.ToByte(' '), Convert.ToByte('W'),
                                   Convert.ToByte('o'), Convert.ToByte('r'), Convert.ToByte('l'),
                                   Convert.ToByte('d') };

using (FileStream file = new FileStream(@"E:\Democode\Greetings.txt",
                                       FileMode.OpenOrCreate, FileAccess.ReadWrite, FileShare.ReadWrite))
{
    file.Lock(0, firstMessage.Length + secondMessage.Length);

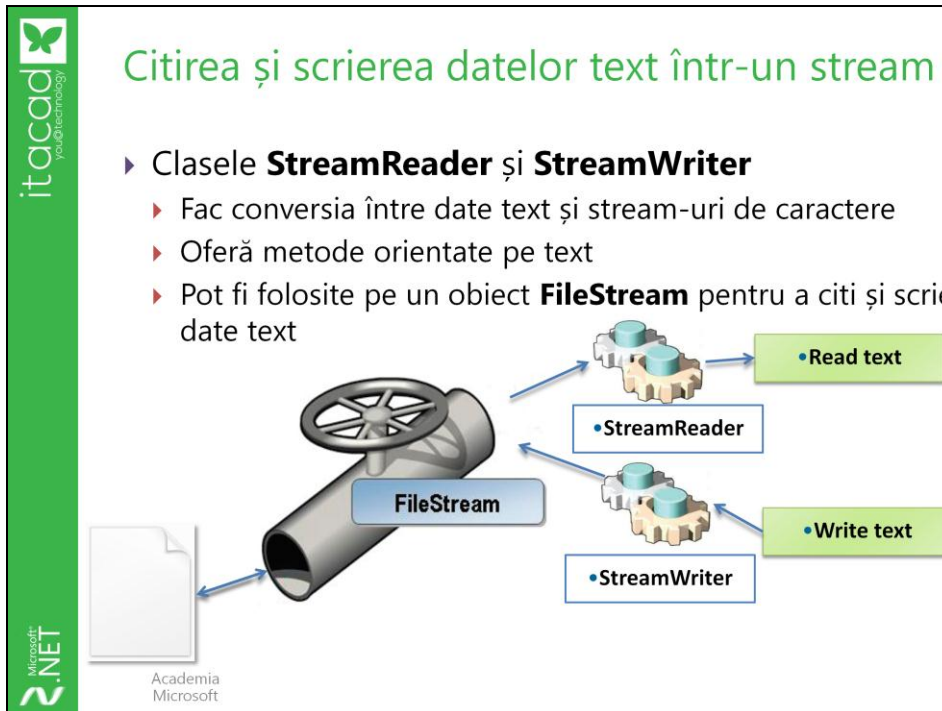
    file.Write(firstMessage, 0, firstMessage.Length);
    file.Write(secondMessage, 0, secondMessage.Length);

    file.Unlock(0, firstMessage.Length + secondMessage.Length);

    file.Seek(-3, SeekOrigin.End);

    byte[] dataRead = new byte[3];
    file.Read(dataRead, 0, dataRead.Length);
    Console.WriteLine("Final 3 bytes: {0}{1}{2}", Convert.ToChar(dataRead[0]),
                    Convert.ToChar(dataRead[1]), Convert.ToChar(dataRead[2]));
}
```

```
file.Close();  
}
```



Clasa **FileStream** oferă acces foarte rapid la datele dintr-un fișier. Totuși, aceste clase citesc și scriu date ca o serie de octeți. Dacă aceste date sunt un șir de string-uri, date binare ce reprezintă date numerice cum ar fi lista de prețuri dintr-un supermarket, este treaba programatorului să convertească octeții citați în tipul corect. Acest proces poartă numele de serializare / deserializare.

Platforma .NET 2.0 oferă clase ce pot să facă automat serializarea datelor ce se găsesc în tipuri primitive cum ar fi tipurile întregi, sau chiar în tipuri mai complexe cum ar fi string-uri, într-un șir de octeți ce poate fi trimis unui stream. Platforma .NET oferă de asemenea clase ce pot deserializa o serie de octeți într-un tip primitiv sau într-un string. De exemplu, clasele **StreamReader** și **StreamWriter** pot lucra cu un obiect de tipul **FileStream** pentru a converti automat între date text și o serie de biți ce pot fi scriși sau citați într-un stream de date

### Clasa **StreamReader**

Această clasă este folosită pentru a citi date de tip text dintr-un stream. Constructorului clasei i se poate trimite ca parametru fie un stream existent, fie numele unui fișier ce este deschis automat ca un stream. Se poate specifica și modul de codificare al caracterelor.

Datele citite sunt fie o secvență de caractere, fie un string, depinzând de metoda apelată. Următorul tabel oferă o listă a metodelor importante din clasa **StreamReader**

Metoda	Descriere
<b>Read</b>	Această metodă are două versiuni. Prima dintre acestea citește următorul caracter din stream, avansează pointerul cu 1; a doua variantă primește ca parametri un vector de octeți, citește un număr specificat de caractere pe care le pune în acest vector și avansează pointerul cu numărul de caractere citite. Metoda întoarce numărul de caractere ce a fost citit
<b>ReadLine</b>	Această metodă citește o linie de caractere dintr-un stream, avansează pointerul și întoarce linia citită ca un string.
<b>ReadToEnd</b>	Această metodă citește restul datelor începând de la poziția curentă dintr-un stream și întoarce un string cu aceste date.
<b>Peak</b>	Această metodă întoarce următorul caracter, fără a muta pointerul streamului. Dacă acest caracter nu poate să fie citit, metoda întoarce -1
<b>Close</b>	Această metodă închide obiectul <b>StreamReader</b> și streamul asociat.

Clasa **StreamWriter** este folosită pentru scrierea de date text într-un stream, construcția sa fiind asemănătoare clasei **StreamReader**. Următorul tabel prezintă câteva dintre metodele des folosite ale acestei clase.

Metoda	Descriere
<b>Write</b>	Această metodă convertește tipurile primitive de date într-o reprezentare pe caractere și scrie aceste caractere în stream. Metoda este supraîncărcată pentru a primi ca parametru diverse tipuri primitive de date
<b>WriteLine</b>	Asemănătoare metodei precedente, dar va adăuga caracterul de linie nouă la finalul datelor scrise
<b>NewLine</b>	Această proprietate specifică stringul ce este folosit pentru a marca o linie nouă. Proprietatea poate fi modificată pentru a schimba terminatorul de linie
<b>Flush</b>	Această metodă golește memoria buffer internă și scrie toate datele către stream
<b>Close</b>	Această metodă închide obiectul <b>StreamWriter</b> și streamul asociat.

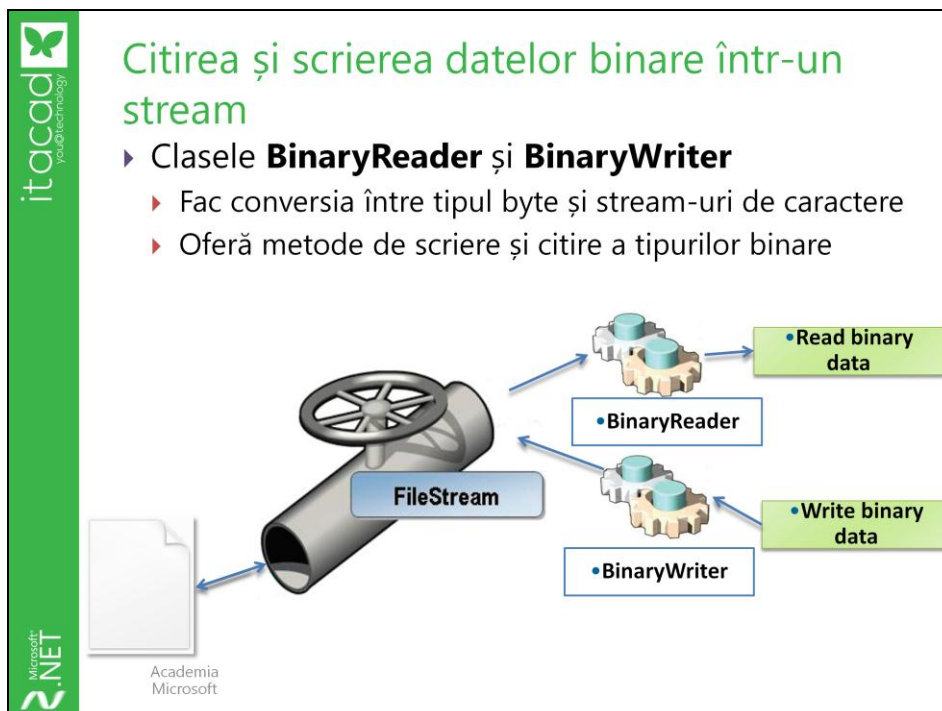
## Folosirea claselor **StreamReader** și **StreamWriter**

Următorul exemplu crează un obiect **StreamWriter** pentru a scrie informații despre un client la finalul unui fișier. Dacă fișierul nu există, este creat. Codul apoi crează un obiect **StreamReader** pentru a citi informațiile ce au fost scrise și afișază rezultatul.

```
using System.IO;
...
string customerName = "John";
int customerAge = 43;
string customerNationality = "British";
string data = "";

using (StreamWriter writer = new StreamWriter(@"E:\Democode\Customers.txt"))
{
    writer.WriteLine("Customer Name: {0}", customerName);
    writer.WriteLine("Age: {0}", customerAge);
    writer.WriteLine("Nationality: {0}", customerNationality);
    writer.Close();
}

using (StreamReader reader = new StreamReader(@"E:\Democode\Customers.txt"))
{
    data = reader.ReadLine();
    Console.WriteLine("{0}", data);
    data = reader.ReadLine();
    Console.WriteLine("{0}", data);
    data = reader.ReadLine();
    Console.WriteLine("{0}", data);
    reader.Close();
}
```



Clasele **StreamReader** și **StreamWriter** sunt excelente pentru a procesa date text. Totuși, dacă este nevoie de citire și scriere de date binare, platforma .NET oferă clasele **BinaryReader** și **BinaryWriter** pentru acest scop.

### Clasa **BinaryReader**

Această clasă operează asemănător clasei **StreamReader**. Diferența dintre acestea este că **BinaryReader** convertește datele citite dintr-un stream în valori bazate pe oricare dintre tipurile primitive ale platformei .NET. Pentru a realiza acest lucru, platforma oferă o serie de metode ce permit citirea diverselor tipuri de date. Următorul tabel prezintă câteva dintre acestea.

Metoda	Descriere
<b>Read</b>	Această metodă este supraîncărcată; fie va citi următorul caracter din stream și îl va întoarce valoare (sau -1 dacă nu mai există date), fie va citi un număr specificat de octeți într-un vector, fie va citi un număr specificat de caractere într-un vector
<b>ReadBoolean, ReadByte, ReadChar, ReadDecimal, ReadDouble, ReadInt32, ReadString</b>	Aceste metode vor citi tipul de date specificat de numele metodei din streamul asociat

Clasa **BinaryReader** de asemenea oferă metoda **PeekChar** ce citește următorul caracter fără a muta pointerul în stream. Metoda va întoarce -1 dacă nu mai sunt date. Metoda este folosită dacă trebuie să stabiliți dacă mai sunt date de citit înainte de a încerca să faceți asta.

### Clasa **BinaryWriter**

Această clasă este analog clasei **StreamWriter** dar aplicată pe date binare. Clasa **BinaryWriter** suprascrie metoda **Write**, aceasta putând să primească ca parametru un tip primitiv, un vector de valori byte sau un vector de caractere, le convertește apoi într-o serie de octeți ce sunt trimiși streamului.

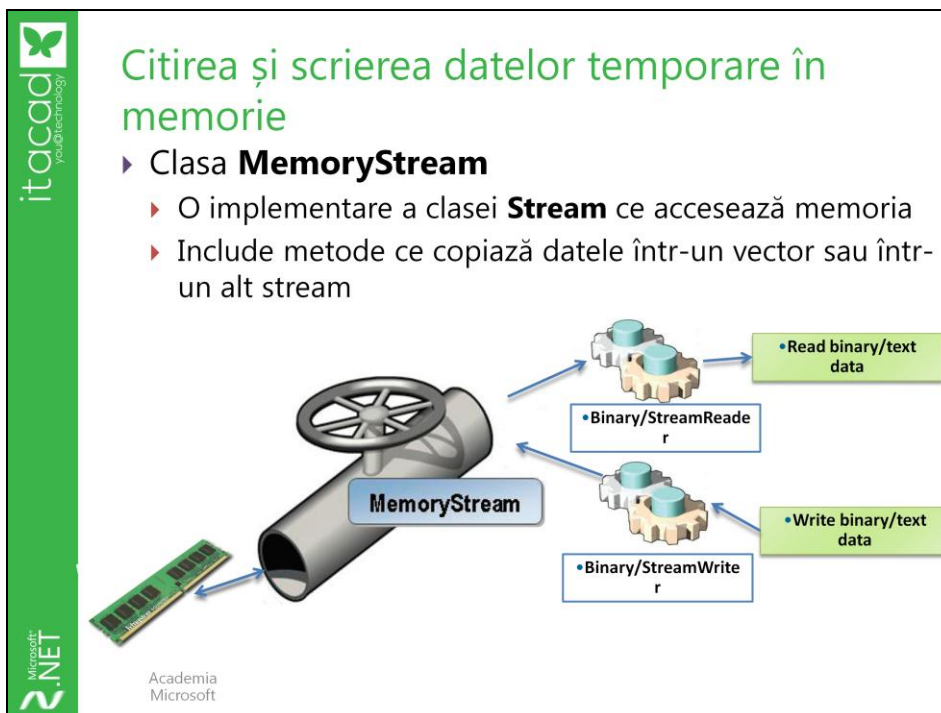
### Folosirea claselor **BinaryReader** și **BinaryWriter**

Următorul exemplu de cod prezintă modul de folosire al clasei **BinaryWriter** pentru a stoca o serie de ID-uri de produse, nume și prețuri ce sunt ținute într-un fișier ce este accesibil printr-un obiect de tipul **FileStream**. Codul crează apoi o instanță a clasei **BinaryReader** ce citește aceste date înapoi și afișază rezultatul

```
using System.IO;
using System.Text;
...
struct ProductData
{
    public int ProductID;
    public string ProductName;
    public decimal ProductPrice;
    public ProductData(int id, string name, decimal price)
    {
        this.ProductID = id;
        this.ProductName = name;
        this.ProductPrice = price;
    }
}
...
```



```
ProductData[] products = new ProductData[] {  
    new ProductData(1, "Bucket", 1.99M),  
    new ProductData(2, "Spade", 1.50M),  
    new ProductData(3, "Deckchair", 7.99M),  
    new ProductData(4, "Windbreak", 5.25M)};  
  
using (FileStream file = new FileStream(@"E:\Democode\Products.dat",  
    FileMode.OpenOrCreate, FileAccess.ReadWrite))  
{  
    BinaryWriter writer = new BinaryWriter(file, Encoding.Unicode);  
    foreach (ProductData item in products)  
    {  
        writer.Write(item.ProductID);  
        writer.Write(item.ProductName);  
        writer.Write(item.ProductPrice);  
    }  
    writer.Flush();  
  
    file.Seek(0, SeekOrigin.Begin);  
    BinaryReader reader = new BinaryReader(file, Encoding.Unicode);  
  
    while (reader.PeekChar() != -1)  
    {  
        int productID = reader.ReadInt32();  
        string productName = reader.ReadString();  
        decimal productPrice = reader.ReadDecimal();  
        Console.WriteLine("Product: {0} {1} {2:C}", productID, productName,  
            productPrice);  
    }  
    reader.Close();  
}
```



Aplicațiile în mod frecvent trebuie să creeze fișiere mici, temporare. Aceste fișiere pot fi create pe disc folosind metoda **GetTempFileName** a clasei **Path**. Totuși, dacă datele au o durată de viață scurtă și trebuie șterse înainte ca aplicația să se închidă, o variantă foarte utilă este stocarea în memorie și nu ca un fișier fizic pe hard. Accesarea memoriei este mult mai rapidă decât citirea și scrierea din fișiere și, de asemenea, nu trebuie ca programatorul să se ocupe de ștergerea ei la finalizarea rulării aplicației.

Platforma .NET 2.0 oferă clasa **MemoryStream** ce operează într-o manieră similară clasei **FileStream**, diferența fiind că datele vor fi stocate în memorie.

### Clasa **MemoryStream**

Un obiect de tipul **MemoryStream** implementează metodele clasei **Stream** peste o zonă de memorie. Obiecte de tipul **MemoryStream** pot fi construite în jurul unui vector de octeți existent. De asemenea, se poate specifica cantitatea de memorie necesară, caz în care obiectul **MemoryStream** va încerca să aloce respectiva zonă de memorie. Alternativ, puteți lăsa obiectul **MemoryStream** să administreze singur necesarul de memorie. Dacă legați o instanță a lui **MemoryStream** de un vector de octeți, dimensiunea sa este fixă și nu se poate modifica.

Dacă specificați cantitatea inițială de memorie sau permiteți obiectului **MemoryStream** să își administreze singur necesarul de memorie, veți putea mări sau micșora dimensiunea stream-ului după cum dictează cerințele aplicației. Pentru a face acest lucru, apăsați metoda **SetLength**, metodă moștenită din clasa **Stream**. Dacă încercați să scrieți mai multe date decât este spațiu disponibil, obiectul **MemoryStream** va face automat redimensionarea; acest lucru se va realiza însă în cuante de dimensiune mică, metoda nefiind eficientă.

Ca și clasa **FileStream**, clasa **MemoryStream** oferă metodele **Read**, **ReadByte**, **Write**, **WriteByte**. Dacă doriți să stocați text sau date binare, folosiți instanțe ale claselor **StreamWriter**, **BinaryWriter** peste **MemoryStream** pentru a vă permite să faceți asta. Similar, citrea poate fi făcută folosind clasele **StreamReader** și **BinaryReader**.

Clasa **MemoryStream** oferă o pereche de metode în plus, metode ce nu sunt accesibile clasei **FileStream**. Aceste metode sunt **ToArray**, ce întoarce conținutul streamului într-un vector de octeți, și **WriteTo**, ce scrie conținutul streamului într-un alt stream. **WriteTo** este în special utilă dacă doriți să procesați date în memorie și la final luați decizia de a le scrie într-un fișier.

### Folosirea clasei **MemoryStream**

Următorul cod exemplifică modul de construire a obiectului **MemoryStream** și folosirea sa împreună cu un obiect **StreamWriter** pentru a scrie date ce sunt stocate într-un vector de stringuri. Codul stochează apoi datele într-un fișier folosind metoda **WriteTo**. O astfel de operație de scriere în bloc a datelor este mult mai eficientă decât o serie de operații de scriere a unor segmente mici.

```
using System.IO;
...
string[] data = new string[] { "The ", "cat ", "sat ", "on ", "the ", "mat" };

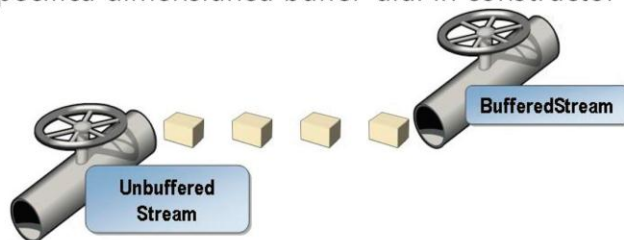
// Create a MemoryStream object that manages its own memory
using (MemoryStream memory = new MemoryStream())
{
    // Copy the data in the string array to the MemoryStream
    StreamWriter writer = new StreamWriter(memory);
    foreach (string item in data)
    {
        writer.Write(item);
    }
    writer.Flush();

    // Persist the data in the MemoryStream to a file
    using (FileStream file = new FileStream(@"E:\Democode\StringData.txt",
        FileMode.Create, FileAccess.Write))
    {
        memory.WriteTo(file);
        file.Close();
    }
    memory.Close();
}
```

## Adăugarea unui buffer unui stream *unbuffered*

### ► Clasa **BufferedStream**

- Adaugă facilitatea de buffering unui stream unbuffered
- Acționează ca un wrapper peste un unbuffered stream
- Implementează metodele clasei **Stream**, dar datele sunt reținute până la apelul metodelor **Flush** sau **Close**, sau până când buffer-ul este plin
- Specifică dimensiunea buffer-ului în constructor



Academia  
Microsoft

Clasa **FileStream** oferă buffering pentru a oferi performanțe bune operațiilor. Buffering permite unui obiect **FileStream** să execute operații I/O mai eficient stocând mai multe instrucțiuni **Read** și **Write** și executându-le într-un singur acces la disc. Totuși, nu toate stream-urile oferă această facilitate. De exemplu, clasa **NetworkStream** din namespace-ul **System.Net.Sockets** nu oferă buffering. De asemenea, clasa **Stream** nu oferă buffering deci, implicit, nici clasele pe care le derivați din aceasta nu vor oferi. Pentru a realiza acest lucru, va trebui să folosiți clasa **BufferedStream**.

### Clasa **BufferedStream**

Această clasă adaugă posibilitatea unei clase **Stream** să facă buffering. O instanță a clasei **BufferedStream** se obține trimițând constructorului instanța clasei ce nu suportă buffering, împreună, eventual, cu dimensiunea buffer-ului. Stream-ul de bază va fi în acest moment comandat indirect, prin intermediul instanței **BufferedStream**.

## Folosirea clasei **BufferedStream**

Următorul exemplu de cod demonstrează modul în care este creat un obiect de tipul **BufferedStream**

```
using System.IO;

...

class UnbufferedCustomStream : Stream
{
    ...
}

...

using (UnbufferedCustomStream ucs = new UnbufferedCustomStream())
{
    using (BufferedStream buffered = new BufferedStream(ucs))
    {
        byte[] data = new byte[100];
        buffered.Write(data, 0, data.Length);
        buffered.Close();
    }
}
```

## Sporirea securității - **Isolated Storage**

- ▶ Ce este **Isolated Storage**?
- ▶ Administrarea fișierelor și directoarelor în locații izolate
- ▶ Citirea și scrierea fișierelor în locații izolate

Aplicațiile trebuie să creeze fișiere temporare sau fișiere ce conțin informații specifice pentru fiecare utilizator. Aceste fișiere pot conține date cu caracter privat, ce nu trebuie să poată fi citite de orice proces. Pentru date de dimensiuni mici și ale căror durată de viață este limitată la timpul de rulare al aplicației, acestea pot fi reținute în memorie. Totuși, dacă datele nu satisfac aceste cerințe, o alternativă trebuie găsită.

Platforma .NET oferă locații izolate în care aplicațiile își pot stoca fișiere pe care să le protejeze de acces neautorizat al utilizatorilor sau a altor aplicații.

### Obiective

La finalul acestei lecții, veți putea să:

- descrieți modul de funcționare a locațiilor izolate
- folosiți clasa **IsolatedStorageFile** pentru a crea o locație izolată și pentru a stoca și administra fișierele din aceasta
- folosiți clasa **IsolatedStorageFileStream** pentru a crea fișiere și pentru a crea și citi date aflate în fișiere ce se găsesc în locații izolate

## Ce este **Isolated Storage**?

- ▶ Loc de stocare a fișierelor privat unei unități de asamblare și complet izolat de spațiul de stocare al altor unități de asamblare
- ▶ Un alt set de privilegii sunt necesare pentru a putea accesa și modifica fișierele de aici
- ▶ Există locații specifice pentru fiecare utilizator, precum și pentru întreaga mașină

Locațiile izolate oferă un mod rapid de creare a fișierelor și a folderelor într-un mod ce permite accesarea lor doar din unitatea de asamblare care le-a creat.

Spre deosebire de sistemul de fișiere obișnuit, ce operează folosind privilegii **FileIOPermission**, locațiile izolate sunt accesibile folosind un alt sistem de privilegii, **IsolatedStorageFilePermission**. Aceste privilegii sunt acordate implicit aplicațiilor ce rulează în medii restricționate, cum ar fi cele ce au un nivel de încredere scăzut sau cele din zona Internet.

Locațiile izolate sunt organizate per utilizator, fiecare utilizator având o zonă de stocare numită **data compartment**. Doi utilizatori nu pot partaja același data compartment.

Într-un data compartment, locațiile izolate sunt împărțite în store-uri. Fiecare unitate de asamblare are propriul său store. Dacă o unitate de asamblare crează un fișier într-o locație izolată, fișierul este creat în store-ul respectivei unități de asamblare. Dacă două unități de asamblare crează un fișier ce are același nume în locații izolate, fișierele vor fi diferite, fiecare în store-ul său.

## Administrarea fișierelor și a folderelor

### ► Clasa **IsolatedStorageFile**

- Implementează un sistem de fișiere ierarhica într-o locație izolată
- Oferă metode ce creează și accesează un store aflat într-o locație izolată
- Conține metode ce creează și administrează fișierele și folderele într-o locație izolată

Academia  
Microsoft

Platforma .Net oferă suport pentru crearea și administrarea de fișiere de date ce sunt ținute în locații izolate folosind clasa **IsolatedStorageFile** ce se găsește în namespace-ul **System.IO.IsolatedStorage**.

Clasa oferă o serie de metode statice ce întorc obiecte **IsolatedStorageFile**, depinzând de ce locație izolată se dorește a fi accesată.

Obiectele **IsolatedStorageFile** obținute pot fi folosite pentru crearea și administrarea fișierelor și a folderelor.



Următorul tabel prezintă metodele statice ale clasei **IsolatedStorageFile**

Metoda	Descriere
<b>GetUserStoreForAssembly</b>	Această metodă întoarce o locație unică, specifică utilizatorului pentru unitatea de asamblare în care a fost apelată metoda. Locația este stabilită analizând identitatea oferită de unitatea de asamblare și identitatea utilizatorului. Dacă unitatea de asamblare nu oferă informații privind identitatea sa (de exemplu, dacă nu are specificat un nume unic), apelul metodei va întoarce câte o locație nouă de fiecare dată.
<b>GetUserStoreForDomain</b>	Această metodă întoarce o locație unică, specifică utilizatorului pentru domeniul aplicației și unitatea de asamblare în care metoda a fost apelată. Locația este stabilită analizând identitatea unității de asamblare, a domeniului aplicației și a utilizatorului. De aceea, locația întoarsă va fi tot timpul diferită de la aplicație la aplicație, chiar dacă metoda se apelează dintr-o unitate de asamblare cu nume unic.
<b>GetUserStoreForApplication</b>	Această metodă întoarce o locație unică, specifică utilizatorului în cazul în care se apelul se face dintr-o aplicație ClickOnce
<b>GetMachineStoreForAssembly</b>	Această metodă este similară <b>GetUserStoreForAssembly</b> , diferența fiind că locația va fi specifică mașinii, nu utilizatorului
<b>GetMachineStoreForDomain</b>	Această metodă este similară <b>GetUserStoreForDomain</b> , diferența fiind că locația va fi specifică mașinii, nu utilizatorului
<b>GetMachineStoreForApplication</b>	Această metodă este similară <b>GetMachineStoreForApplication</b> , diferența fiind că locația va fi specifică mașinii, nu utilizatorului

- O aplicație ClickOnce este o aplicație pe care utilizatorul o instalează și rulează la un click pe un link într-o pagină web

## Crearea și administrarea fișierelor și a folderelor

După ce se obține accesul la o locație izolată printr-un obiect **IsolatedStorageFile**, acesta poate fi folosit pentru a crea fișiere și foldere unde pot fi reținute date. Următorul tabel prezintă câteva dintre metodele des folosite pentru astfel de operații

Metoda	Descriere
<b>CreateDirectory</b>	Această metodă crează un folder în locația izolată. Dacă folderul există, nu se crează unul nou și nu se aruncă excepție
<b>DeleteDirectory</b>	Această metodă șterge folderul specificat din locația izolată. Folderul trebuie să fie gol.
<b>DeleteFile</b>	Această metodă șterge un fișier cu numele specificat
<b>GetDirectoryNames</b>	Această metodă primește un string ce specifică o expresie regulată și întoarce toate numele folderelor ce corespund.
<b>GetFileNames</b>	Asemănător metodei <b>GetDirectoryNames</b> , metoda va întoarce nume de fișiere
<b>Remove</b>	Această metodă șterge locația izolată împreună cu tot conținutul acesteia
<b>Close</b>	Această metodă închide locația izolată fără a șterge conținutul acesteia. Locația va putea să fie redeschisă

Următorul exemplu de cod prezintă modul în care se deschide store-ul utilizatorului curent dintr-o locație izolată și apoi crearea unei serii de foldere. Al doilea exemplu deschide același store și afișază lista de foldere.

```
using System;
using System.IO;
using System.IO.IsolatedStorage;
...
// Create a series of folders
IsolatedStorageFile isolatedStorage =
IsolatedStorageFile.GetUserStoreForDomain();

isolatedStorage.CreateDirectory("PrivateFolder1");
isolatedStorage.CreateDirectory("PrivateFolder2");
isolatedStorage.CreateDirectory("PrivateFolder3");
isolatedStorage.CreateDirectory("TestFolder");
isolatedStorage.Close();

// List folders
IsolatedStorageFile isolatedStorage =
IsolatedStorageFile.GetUserStoreForDomain();
string[] foldernames = isolatedStorage.GetDirectoryNames("Private*");
foreach (string name in foldernames)
{
    Console.WriteLine("{0}", name);
}
isolatedStorage.Close();
```

## Citirea și scrierea fișierelor

### ► Clasa **IsolatedStorageFileStream**

- Clasă de acces la un fișier ce crează, citește și scrie fișiere într-o locație izolată
- Folosită împreună cu obiecte de tipul **BinaryReader/Writer** sau **StreamReader/Writer**
- Se pot folosi obiecte de tipul **DeflateStream**, **GZipStream**, **CryptoStream** pentru compresie sau criptare

Puteți citi și scrie date într-un fișier ce este ținut într-o locație izolată folosind clasa **IsolatedStorageFileStream** ce se află în namespace-ul **System.IO.IsolatedStorage**

### Clasa **IsolatedStorageFileStream**

Această clasă este derivată din **FileStream**, oferind aceeași funcționalitate cu clasa părinte, diferența fiind fișierele asupra cărora se aplică. Constructorului clasei **IsolatedStorageFileStream** îi trebuie oferită o instanță a clasei **IsolatedStorageFile**.

Mai departe, obiectul obținut poate fi trimis ca parametru constructorului uneia dintre clasele **BinaryReader/Writer**, **StreamReader/Writer**, **DeflateStream**, **GZipStream**, **CryptoStream**, în funcție de operațiile ce se doresc a fi executate asupra fișierului.

### Utilizarea clasei **IsolatedStorageFileStream**

Următorul exemplu demonstrează modul de creare a unui fișier într-un folder numit Demodata, ce este ținut într-o locație izolată, scrierea și citrea acestui fișier.

```

using System;
using System.IO;
using System.IO.IsolatedStorage;
...
// Create a file in isolated storage
string[] data = new string[] { "Private", "User", "Data" };

IsolatedStorageFile isolatedStorage =
IsolatedStorageFile.GetUserStoreForDomain();
isolatedStorage.CreateDirectory("Demodata");
IsolatedStorageFileStream fileStream = new
IsolatedStorageFileStream(@"Demodata\Private.txt",
    FileMode.Create, FileAccess.Write, isolatedStorage);

StreamWriter writer = new StreamWriter(fileStream);
foreach (string item in data)
{
    writer.WriteLine(item);
}
writer.Close();
isolatedStorage.Close();
...
// Read the file
IsolatedStorageFile isolatedStorage =
IsolatedStorageFile.GetUserStoreForDomain();
IsolatedStorageFileStream fileStream = new
IsolatedStorageFileStream(@"Demodata\Private.txt",
    FileMode.Open, FileAccess.Read, isolatedStorage);

StreamReader reader = new StreamReader(fileStream);
string line = reader.ReadLine();
while (line != null)
{
    Console.WriteLine("{0}", line);
    line = reader.ReadLine();
}
reader.Close();
isolatedStorage.Close();

```

## Review

- ▶ Administrarea sistemului de fișiere
- ▶ Citirea și scrierea folosind stream-uri
- ▶ Sporirea securității unei aplicații folosind Isolated Storage

