



Leseprobe

Walter Doberenz, Thomas Gewinnus

Der Visual C#-Programmierer

Visual C# lernen - Professionell anwenden - Lösungen nutzen

ISBN: 978-3-446-42021-2

Weitere Informationen oder Bestellungen unter

<http://www.hanser.de/978-3-446-42021-2>

sowie im Buchhandel.

# 12

## Zugriff auf das Dateisystem

Dieses Kapitel behandelt die folgenden Themen:

- Klassen für Verzeichnis- und Dateioperationen
- Operationen auf Verzeichnisebene
- Weitere wichtige Klassen

Inhalt dieses Kapitels ist die Arbeit auf Verzeichnisebene, womit das Erstellen, Löschen, Kopieren, Verschieben, Umbenennen, Durchsuchen und Überwachen von Verzeichnissen und Dateien gemeint ist.



### **Hinweis**

Zum Lesen und Schreiben von Dateiinhalten sowie der Arbeit mit Streams kommen wir erst im nachfolgenden Kapitel. ■

## 12.1 Grundlagen

Dateien dienen dazu, Daten auf Festplatte (oder anderen Speichermedien) dauerhaft zu sichern bzw. Daten zwischen einer Quelle und einem Ziel zu transportieren. Das .NET-Framework stellt dafür im Namespace `System.IO` eine Anzahl leistungsfähiger Klassen zur Verfügung, die diese Aufgabe vereinfachen sollen.

### 12.1.1 Das Datei-System von Windows

Wenn Sie an Dateien denken, so fallen Ihnen dazu meist Begriffe wie Dateinamen, Dateigröße, Dateiattribute und Dateiverzeichnis ein. Als Programmierer sollten Sie aber auch den tieferen Sinn des Datei-Systems von Windows verstanden haben:

- Eine Datei unter Windows ist eine geordnete und benannte Sammlung von Bytefolgen, die persistent (meist auf der Festplatte) abgespeichert sind.
- Ein Verzeichnis (bzw. Ordner) in Windows ist einfach ein anderer Dateityp, der wiederum andere Dateien und Unterverzeichnisse enthalten kann.

In früheren Programmiersprachen waren Sie meist auf das `FileSystemObject` oder den Direktzugriff auf die Win32 API angewiesen, um Ihre Datei-Operationen zu implementieren.



#### Hinweis

Unter .NET stellt der `System.IO`-Namensraum eine ausreichende Anzahl robuster Objekte zur Verfügung, mit der die Interaktion mit dem Windows-Dateisystem erleichtert und vereinfacht werden kann. ■

### 12.1.2 Klassen für Verzeichnis- und Dateioperationen

Die wichtigsten Klassen für Manipulationen mit Verzeichnissen und Dateien sind die Pärchen `Directory/DirectoryInfo` und `File/FileInfo`, die sich vor allem hinsichtlich ihrer Instanzierbarkeit unterscheiden. Weitere interessante Klassen des `System.IO`-Namespace entnehmen Sie der folgenden Tabelle.

#### Übersicht

**Tabelle 12.1** Wichtige Klassen für Verzeichnis- und Dateioperationen

Klasse	Beschreibung
<code>Directory</code>	Die statischen Methoden erlauben das Erstellen, Verschieben und Benennen von Verzeichnissen und Unterverzeichnissen.
<code>DirectoryInfo</code>	Ähneln der <code>Directory</code> -Klasse, enthält aber nur Instanz-Methoden.
<code>Path</code>	Die statischen Methoden erlauben die plattformübergreifende Arbeit mit Verzeichnissen.
<code>File</code>	Die statischen Methoden erlauben das Erzeugen, Kopieren, Löschen, Verschieben und Öffnen von Dateien.

Klasse	Beschreibung
FileInfo	Ähneln der File-Klasse, enthält aber nur Instanz-Methoden.
FileSystemWatcher	Löst Ereignisse zum Überwachen des Dateisystems aus.
DriveInfo	Liefert Laufwerksinformationen

Die Methoden der Klassen File und FileInfo liefern auch die Voraussetzungen für den Schreib- und Lesezugriff auf Dateien.



### Hinweis

Beachten Sie auch die Unter-Namensräume wie System.IO.Compression (Komprimieren von Daten) und System.IO.Ports (Zugriff auf die serielle Schnittstelle des PC). ■



## 12.1.3 Statische versus Instanzen-Klassen

Bei statischen Klassen müssen Sie jeder Methode den Dateinamen oder den Verzeichnispfad übergeben. Das kann dann ziemlich lästig werden, wenn Sie diese Methoden öfters hintereinander aufrufen müssen. Die entsprechenden Eigenschaften der Instanzen-Klassen FileInfo und DirectoryInfo hingegen erlauben es Ihnen, den Datei- oder Verzeichnisnamen bereits im Konstruktor einmalig zu spezifizieren.



**Beispiel** Zwei alternative Möglichkeiten zum Anzeigen von Erstellungsdatum und Zeitpunkt des letzten Zugriffs auf die Datei `c:\test\info.txt`:

```
using System.IO;
```

Mit statischer Klasse:

```
label1.Text = File.GetCreationTime("c:\\test\\info.txt").ToString();
label2.Text = File.GetLastAccessTime("c:\\test\\info.txt").ToString();
```

Mit Instanzen-Klasse:

```
FileInfo myFile = new FileInfo("c:\\test\\info.txt");
label1.Text = myFile.CreationTime.ToString();
label2.Text = myFile.LastAccessTime.ToString();
```

Ein weiterer wichtiger Unterschied zwischen beiden Klassen soll keinesfalls verschwiegen werden:



### Hinweis

Wenn Sie mit den Methoden der statischen Klassen File, Directory und Path arbeiten, werden Sicherheitsüberprüfungen bei jedem Methodenaufruf vorgenommen, bei den Instanzen-Methoden der Klassen FileInfo und DirectoryInfo geschieht dies nur ein einziges Mal. ■

## 12.2 Operationen auf Verzeichnisebene

Wir behandeln hier das Erstellen, Löschen, Kopieren, Verschieben, Umbenennen, Durchsuchen und Überwachen von Verzeichnissen und Dateien.

### 12.2.1 Verzeichnisse erzeugen und löschen

Wie es für viele Dateioperationen typisch ist, haben Sie auch hier die Qual der Wahl zwischen zwei Klassen.

#### Mit Directory-Klasse

Die einfachsten Möglichkeiten zum Erzeugen und Löschen von Verzeichnissen bieten die statischen Methoden `CreateDirectory` und `Delete` der `Directory`-Klasse.



**Beispiel** Ein Verzeichnis erzeugen und anschließend wieder löschen.

```
using System.IO;
...
string pfad = "c:\\temp";
Directory.CreateDirectory(pfad); // falls Verzeichnis bereits vorhanden, passiert nichts!
Directory.Delete(pfad, true); // löscht auch vorhandene Unterverzeichnisse und Dateien
```

#### Mit DirectoryInfo-Klasse

Das gleiche Ziel, allerdings etwas umständlicher, erreicht man mit der `Create`-Methode der instanziierten `DirectoryInfo`-Klasse, wobei mittels `CreateSubdirectory` auch das Hinzufügen von Unterverzeichnissen möglich ist.



**Beispiel** Ein Verzeichnis und ein Unterverzeichnis anlegen und wieder löschen.

```
using System.IO;
...
DirectoryInfo di = new DirectoryInfo("c:\\temp");
di.Create();
di.CreateSubdirectory("temp1");
di.Delete(true); // löscht inklusive vorhandener Unterverzeichnisse und Dateien
```



#### Hinweis

Der Aufruf von `Delete` ohne Parameterangabe funktioniert nur, wenn das Verzeichnis leer ist! ■

## 12.2.2 Verzeichnisse umbenennen und verschieben

Für beide Aufgaben verwenden Sie am besten die Move-Methode der statischen Directory-Klasse.



**Beispiel** Verzeichnis *c:\tempX* wird nach *c:\beispiele* verschoben und umbenannt in *tempY*

```
using System.IO;
...
Directory.Move("c:\\tempX", "c:\\beispiele\\tempY");
```



### Hinweis

Leider gibt sowohl in der Directory- als auch in der DirectoryInfo-Klasse noch keine Methode, die das Kopieren eines kompletten Verzeichnisses ermöglicht. ■

## 12.2.3 Aktuelles Verzeichnis bestimmen

Verwenden Sie dazu die GetCurrentDirectory- bzw. SetCurrentDirectory-Methode der (statischen) Directory-Klasse.



**Beispiel** Festlegen und Anzeigen des aktuellen Arbeitsverzeichnisses.

```
using System.IO;
...
Directory.SetCurrentDirectory("c:\\test");
label1.Text = Directory.GetCurrentDirectory(); // zeigt "c:\test"
```



### Hinweis

Wenn der Dateiname ohne Pfad angegeben wird, bezieht sich die Datei automatisch auf das Projekt- bzw. Arbeitsverzeichnis. ■



**Beispiel** Im Projektverzeichnis wird ein Verzeichnis *\temp* angelegt.

```
using System.IO;
...
Directory.CreateDirectory("temp");
```

## 12.2.4 Unterverzeichnisse ermitteln

Um alle Unterverzeichnisse zu ermitteln, verwenden Sie die GetDirectories-Methode der DirectoryInfo-Klasse.



**Beispiel** Es werden alle Unterverzeichnisse von *c:\* in einer ListBox angezeigt.

```
using System.IO;
...
```

```
DirectoryInfo myDir = new DirectoryInfo("c:\\"); // neues DirectoryInfo-Objekt
DirectoryInfo[] myDirs = myDir.GetDirectories(); // Unterverzeichnisse ermitteln
// und im Array ablegen
for (int i = 0; i < myDirs.Length; i++) // alle Unterverzeichnisse durchlaufen ...
    listBox1.Items.Add(myDirs[i].Name); // ... und Verzeichnisnamen zur ListBox hinzufügen
```

Eine alternative Lösung bietet sich mit der gleichnamigen Methode der (statischen) Directory-Klasse:



**Beispiel** Lösung des gleichen Problems wie im Vorgängerbeispiel

```
using System.IO;
...
string[] myDirs = Directory.GetDirectories("c:\\");
for (int i = 0; i < myDirs.Length; i++)
    listBox1.Items.Add(myDirs[i]);
```



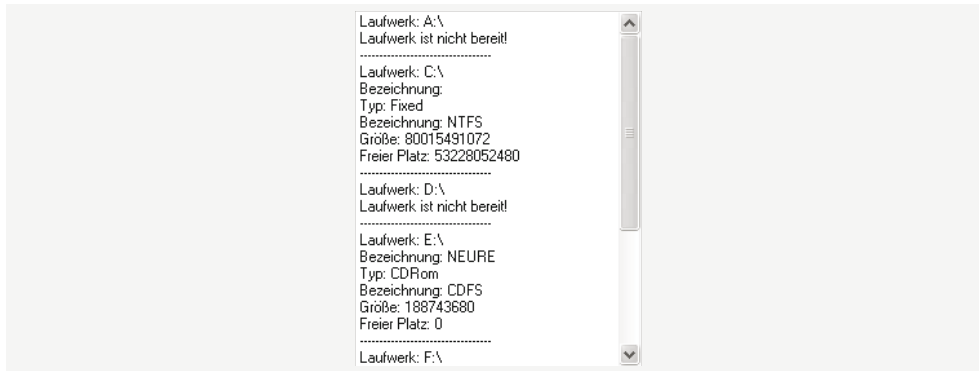
### 12.2.5 Alle Laufwerke ermitteln

Die Klasse DriveInfo ermöglicht den Zugriff auf Laufwerksinformationen.



**Beispiel** Auflisten aller Laufwerke des Systems mit den dazugehörigen Informationen

```
foreach (DriveInfo di in DriveInfo.GetDrives())
{
    listBox1.Items.Add("Laufwerk: " + di.Name);
    if (di.IsReady)
    {
        listBox1.Items.Add("Bezeichnung: " + di.VolumeLabel);
        listBox1.Items.Add("Typ: " + di.DriveType);
        listBox1.Items.Add("Bezeichnung: " + di.DriveFormat);
        listBox1.Items.Add("Größe: " + di.TotalSize);
        listBox1.Items.Add("Freier Platz: " + di.TotalFreeSpace);
        listBox1.Items.Add("-----");
    }
    else
    {
        listBox1.Items.Add("Laufwerk ist nicht bereit!");
        listBox1.Items.Add("-----");
    }
}
```



### 12.2.6 Im Verzeichnis enthaltene Dateien ermitteln

Gewissermaßen als Ergänzung zu `GetDirectories` können Sie mit der `GetFiles`-Methode der Klasse `DirectoryInfo` alle in einem Verzeichnis enthaltenen Dateien ermitteln.



**Beispiel** Alle im Rootverzeichnis `c:\` abgelegten Dateien werden in einer `ListBox` angezeigt.

```
using System.IO;
...
DirectoryInfo myDir = new DirectoryInfo("c:\\"); // neues DirectoryInfo-Objekt
FileInfo[] myFiles = myDir.GetFiles();        // Dateien ermitteln und im Array ablegen
for (int i = 0; i < myFiles.Length; i++)      // alle Dateien durchlaufen ...
    listBox1.Items.Add(myFiles[i].Name);      // ... und Dateinamen zur ListBox hinzufügen
```

Noch kürzer ist der Code bei Verwendung der (statischen) `Directory`-Klasse.



**Beispiel** Das gleiche Problem wie im Vorgängerbeispiel wird gelöst.

```
using System.IO;
...
string[] myFiles = Directory.GetFiles("c:\\"); // String-Array füllen ...
for (int i = 0; i < myFiles.Length; i++)      // alle Einträge durchlaufen
    listBox1.Items.Add(myFiles[i]);           // ... und anzeigen
```

### 12.2.7 Dateien kopieren und verschieben

Am einfachsten realisieren Sie diese Aufgabe mit den statischen `Copy`- bzw. `Move`-Methoden der `File`-Klasse.



**Beispiel** Datei kopieren und anschließend verschieben.

```
using System.IO;
...
string sourcePath = "c:\\sample.txt" ;
string destPath = "c:\\sample1.txt";
```

```
string movePath = "c:\\temp\\sample1.txt";
File.Copy(sourcePath, destPath);      // kopieren
File.Move(sourcePath, movePath);     // verschieben
```

Falls Sie lieber mit Instanzen arbeiten, können Sie die Methoden `CopyTo` und `MoveTo` der Klasse `FileInfo` verwenden.



**Beispiel** Obiges Beispiel wird mit Methoden der `FileInfo`-Klasse realisiert.

```
using System.IO;
...
string sourcePath = "c:\\sample.txt" ;
string destPath = "c:\\sample1.txt";
string movePath = "c:\\temp\\sample1.txt";

FileInfo myFile = new FileInfo(sourcePath);
myFile.CopyTo(destPath);      // kopieren
myFile.MoveTo(movePath);     // verschieben
```

## 12.2.8 Dateien umbenennen

Leider bietet das .NET-Framework keinerlei Möglichkeit zum direkten Umbenennen einer Datei, da die `Name`-Eigenschaft der `FileInfo`-Klasse schreibgeschützt ist und eine `Rename`-Methode fehlt. Verwenden Sie zum Umbenennen also die Methoden `Move` der Klasse `File` bzw. `MoveTo` der Klasse `FileInfo`.



**Beispiel** Umbenennen der Datei `info.txt` in `info_1.txt`.

```
using System.IO;
...
FileInfo myFile = new FileInfo("c:\\test\\info.txt");
myFile.MoveTo("c:\\test\\info_1.txt");
```

## 12.2.9 Dateiattribute feststellen

Um die Dateiattribute zu ermitteln, kann man entweder auf die Eigenschaften der `FileInfo`-Klasse oder aber auch auf die entsprechenden (statischen) Methoden der `File`-Klasse zugreifen:

**Tabelle 12.2** Eigenschaften und Methoden der `FileInfo`/`File`-Klasse

Eigenschaft FileInfo-Klasse	Methode File-Klasse	Beschreibung
Attributes	GetAttributes() SetAttributes()	Wert basiert auf Dateiattribute-Flags ( <i>Archive</i> , <i>Compressed</i> , <i>Directory</i> , <i>Hidden</i> ...)
CreationTime	GetCreationTime() SetCreationTime()	Datum/Uhrzeit der Erstellung

Eigenschaft FileInfo-Klasse	Methode File-Klasse	Beschreibung
LastAccessTime	GetLastAccessTime() SetLastAccessTime()	Datum/Uhrzeit des letzten Zugriffs
LastWriteTime	GetLastWriteTime() SetLastWriteTime()	Datum/Uhrzeit des letzten Schreibzugriffs
Exists	Exists()	Liefert true, falls Datei physikalisch existiert



**Beispiel** Anzeige des Erstellungsdatums einer Datei.

```
using System.IO;
...
tabell.Text = File.GetCreationTime("Liesmich.txt").ToString();
```



**Beispiel** Feststellen, ob eine Datei im Arbeitsverzeichnis existiert.

```
using System.IO;
...
if (File.Exists("Liesmich.txt")) MessageBox.Show("Datei ist vorhanden!");

oder

FileInfo myFile = new FileInfo("Liesmich.txt");
if (myFile.Exists) MessageBox.Show("Datei ist vorhanden!");
```

### 12.2.10 Die FileAttribute-Enumeration

Die verschiedenen Attribute für Dateien und Verzeichnisse sind in der FileAttribute-Enumeration anzutreffen. Die folgende Tabelle zeigt nur die wichtigsten:

**Tabelle 12.3** Die FileAttribute-Enumeration

Member	Beschreibung
Archive	Entspricht dem Archiv-Status der Datei, wie er häufig zum Markieren einer zu löschen- oder einer Backup-Datei verwendet wird
Compressed	Entspricht einer gepackten Datei
Directory	Zeigt an, dass die Datei in Wirklichkeit ein Verzeichnis ist
Encrypted	Die Datei ist verschlüsselt
Hidden	Die Datei ist versteckt und demzufolge in einem gewöhnlichen Verzeichnis unsichtbar
Normal	Es wurden keine Datei-Attribute gesetzt
ReadOnly	Die Datei kann nicht verändert, sondern nur gelesen werden
System	Die Datei gehört zum Betriebssystem oder wird exklusiv von diesem benutzt
Temporary	Die Datei ist temporär, sie wird vom Programm angelegt und wieder gelöscht

Um das Vorhandensein eines bestimmten Datei-Attributes festzustellen, muss eine bitweise ODER-Verknüpfung durchgeführt werden.



**Beispiel** In einer CheckBox wird angezeigt, ob es sich um eine Archiv-Datei handelt.

```
FileAttributes attbs = File.GetAttributes("c:\\beispiele\\test.dat");
if(attbs==(attbs|FileAttributes.Archive)) checkBox2.Checked = true;
else checkBox2.Checked = false;
```

## 12.3 Mehr zur FileInfo-Klasse

Werfen wir noch einen kurzen Blick auf weitere wichtige Member der FileInfo-Klasse.

### 12.3.1 Weitere wichtige Eigenschaften

Die folgende Tabelle liefert eine kurze Zusammenstellung.

**Tabelle 12.4** Wichtige Eigenschaften der FileInfo-Klasse

Eigenschaft	Beschreibung
Directory	Liefert Instanz des übergeordneten Verzeichnisses
DirectoryName	Liefert den vollständigen Dateipfad
Extension	Liefert Dateierweiterung (z.B. <i>.txt</i> für Textdateien)
FullName	Liefert den vollständigen Dateipfad plus Dateinamen
Length	Liefert die Dateigröße in Bytes
Name	Liefert den Dateinamen



**Beispiel** Der Pfad der Datei *liesmich.txt* wird in einem Label angezeigt.

```
using System.IO;
...
FileInfo myFile = new FileInfo("Liesmich.txt");
label1.Text = myFile.DirectoryName;
```

### 12.3.2 GetFileSystemInfos-Methode

Im Zusammenhang mit der Directory-Eigenschaft der FileInfo-Klasse verdient die GetFileSystemInfos-Methode der DirectoryInfo-Klasse unsere besondere Aufmerksamkeit.



**Beispiel** In einer TextBox (MultiLine = true) werden neben dem Verzeichnis einer Datei alle weiteren sich im gleichen Verzeichnis befindlichen Dateien und Unterverzeichnisse angezeigt.

```
using System.IO;
...
```