



// 01

System.IO en paths

Met `using System.IO`; lees, schrijf, maak en verwijder je bestanden en folders. Elk bestand heeft een unieke locatie: het **path**, een `string`.

```
File.WriteAllText("doem.txt", "Hallo");
File.WriteAllText(@"c:\temp\doem.txt", "Hallo");
```

Zonder path komt het bestand in de folder van je programma. Een path is **niet hoofdlettergevoelig**.

// LET OP

Een bestand staat niet altijd waar je denkt en je hebt niet altijd de rechten: gebruik try-catch (`FileNotFoundException`, `UnauthorizedAccessException`).

// 02

Paths samenstellen

`Path.Combine` zet het juiste scheidingsteken: `\` op Windows, `/` op Mac OS. Een speciale folder vraag je op via `Environment`:

```
string pad = Path.Combine("data", "dagboek.txt");
string bureau = Environment.GetFolderPath(
    Environment.SpecialFolder.Desktop);
```

// LET OP

Een hardgecodeerd path als `@"C:\Users\Steven\log.txt"` werkt enkel op de pc van Steven.

// 03

Eerst controleren

Overschrijf of verwijder nooit iets zonder de gebruiker te waarschuwen. Voor een folder: `Directory.Exists` en

`Directory.CreateDirectory`.

```
if (!File.Exists(pad))
{
    using (FileStream fs = File.Create(pad)) { }
}
```

// LET OP

`File.Create` geeft een open `FileStream` terug. Zonder `using` blijft het bestand gelockt en faalt de volgende bewerking (`IOException`).

// 04

Tekst schrijven en lezen

Zet een `StreamWriter` of `StreamReader` in een `using`-blok: aan de `}` gaat het bestand dicht, ook bij een fout.

```
using (var writer = new StreamWriter(pad, true))
{
    writer.WriteLine("Beste dagboek...");
}
using (var reader = new StreamReader(pad))
{
    string regel;
    while ((regel = reader.ReadLine()) != null)
    {
        Console.WriteLine(regel);
    }
}
```

`true`: achteraan toevoegen. `ReadLine` geeft `null` aan het einde. Een **klein** bestand kan korter: `File.ReadAllText` (één `string`), `File.ReadAllLines` (een `string[]`), `File.WriteAllText`.

// 05

Binaire bestanden

Een `BinaryWriter` schrijft elk datatype weg. `FileMode.Create` overschrijft, `FileMode.Open` vereist een bestaand bestand.

```
var fs = File.Open("bond.dat", FileMode.Create);
using (var writer = new BinaryWriter(fs))
{
    writer.Write("Bond"); // 1 + 4 bytes
    writer.Write(7);      // 4 bytes
    writer.Write(true);   // 1 byte
}
```

UITVOER

```
04 42 6F 6E 64 07 00 00 00 01
```

// LET OP

Een `BinaryReader` (`ReadString`, `ReadInt32`, ...) leest in **exact dezelfde volgorde** als je schreef. Anders krijg je onzin of een crash (`EndOfStreamException`).

// 06

File of FileInfo

`File` en `Directory` zijn **static**. `FileInfo` en `DirectoryInfo` maak je eerst aan met `new`.

```
FileInfo info = new FileInfo("bond.dat");
Console.WriteLine(info.Length); // in bytes
info.CopyTo("superman.dat");
info.MoveTo("bond2.dat");
info.Delete();
```

Ook: `Name`, `Exists`, `CreationTime`. `Refresh()` leest de info opnieuw in.

// 07

Folders doorzoeken

```
DirectoryInfo map = new DirectoryInfo("data");
FileInfo[] teksten = map.GetFiles("*.txt");
DirectoryInfo[] folders = map.GetDirectories();
```

Met `SearchOption.AllDirectories` als tweede argument zoekt `GetFiles` ook in de subfolders.

// 08

Serialiseren naar JSON

Een object naar tekst en terug, om de staat van je programma te bewaren. Enkel **publieke properties** komen mee.

```
string json = JsonSerializer.Serialize(barry);
File.WriteAllText("student.json", json);
Student terug =
    JsonSerializer.Deserialize<Student>(json);
```

UITVOER

```
{"Naam": "Barry", "Leeftijd": 25}
```

// 09

Attributen

Boven een property of instantievariabele:

ATTRIBUUT	DOET
<code>[JsonIgnore]</code>	niet mee in de JSON
<code>[JsonPropertyName("X")]</code>	andere naam in de JSON
<code>[JsonInclude]</code>	private instantievariabele toch mee