**"Streams in C# – Daten fließen lassen"**

**Was ist ein Stream?**

Ein Stream ist ein **kontinuierlicher Datenfluss** in eine Richtung:

* Entweder **lesen** (input),
* oder **schreiben** (output).

**Wofür ist ein Stream gut?**

* **Einheitliches Arbeiten mit Datenquellen**  
  Egal ob Sie eine Datei lesen, eine Netzwerkverbindung nutzen oder Daten im Speicher puffern: Überall verwenden Sie dasselbe Grundprinzip (**Read, Write, Seek**).
* **Arbeiten mit großen Datenmengen**  
  Anstatt eine komplette Datei oder Nachricht in den Speicher zu laden, können Sie Daten Stück für Stück verarbeiten. Das spart **Arbeitsspeicher** und ermöglicht das Verarbeiten von sehr großen Daten.
* **Flexibilität und Erweiterbarkeit**  
  Es gibt viele spezialisierte Streams in .NET:
  + FileStream → für Dateien
  + MemoryStream → für RAM-Daten
  + NetworkStream → für Sockets

**Typische Anwendungsfälle**

|  |  |
| --- | --- |
| Anwendungsfall | Stream-Typ |
| Datei lesen/schreiben | FileStream |
| Arbeitsspeicher nutzen | MemoryStream |
| Netzwerkkommunikation | NetworkStream |
| Bild-Upload verarbeiten | Stream (abstrakt) |

**Eigenschaften eines Streams**

* **Sequentiell**: Ein Stream verarbeitet Daten nacheinander.
* **Positionierbar**: Viele Streams haben eine Position, die man setzen oder lesen kann.
* **Lesen/Schreiben** getrennt oder kombiniert.
* Ressourcenintensiv ( **using oder Dispose() ist Pflicht! )**

**MemoryStream für Text**

**Beispiel:**

*using System;*

*using System.IO;*

*using System.Text;*

*class Program*

*{*

*static void Main()*

*{*

*using var stream = new MemoryStream();*

*using var writer = new StreamWriter(stream, Encoding.UTF8);*

*writer.Write("Hallo Welt!");*

*writer.Flush(); // unbedingt nötig, damit alles in den Stream geschrieben wird*

*stream.Position = 0; // zurück an den Anfang springen*

*using var reader = new StreamReader(stream, Encoding.UTF8);*

*string result = reader.ReadToEnd();*

*Console.WriteLine("Gelesen aus MemoryStream: " + result);*

*}*

*}*

Der Text wird **in den RAM** geschrieben.

**Flush()** überträgt den Inhalt des Writers in den Stream.

Position = 0: „Spulen“ zum Anfang.

**using** sorgt für korrektes Schließen und Freigeben.

**FileStream für Textdateien**

**Schreiben in eine Datei:**

*using var fs = new FileStream("ausgabe.txt", FileMode.Create);*

*using var sw = new StreamWriter(fs);*

*sw.WriteLine*("Dies ist eine Datei.");

**Lesen aus der Datei:**

*using var fs = new FileStream("ausgabe.txt", FileMode.Open);*

*using var sr = new StreamReader(fs);*

*Console.WriteLine("Dateiinhalt: " + sr.ReadToEnd());*

**Erklärungen:**

* FileMode.**Create**: erstellt oder überschreibt Datei
* FileMode.**Open**: erwartet vorhandene Datei
* StreamWriter und -Reader übernehmen die Codierung
* **Unterschied** zu File.WriteAllText etc.: **manuellere** Kontrolle, flexibler, **effizienter** bei großen Daten

**Binärdaten mit BinaryWriter / BinaryReader**

Wann verwendet man binäre Dateien?

* Wenn Platz oder Geschwindigkeit wichtig ist (z. B. bei großen Datenmengen)
* Wenn man **nicht möchte**, dass jemand die Datei einfach liest
* Wenn man Daten in **exakter Form** speichern und laden will (z. B. bei Spielen, Konfigurationsdateien, selbst definierten Formaten)

z. B. Zahlen, Booleans oder Objekte speichert, nicht als Text.

*using var fs = new FileStream("daten.bin", FileMode.Create);*

*using var bw = new BinaryWriter(fs);*

*bw.Write(42);*

*bw.Write(3.14);*

*bw.Write("Hallo");*

**Lesen:**

*using var fs = new FileStream("daten.bin", FileMode.Open);*

*using var br = new BinaryReader(fs);*

*int zahl = br.ReadInt32();*

*double d = br.ReadDouble();*

*string text = br.ReadString();*

Console.WriteLine($"{zahl}, {d}, {text}");

**Hinweise:**

* **Wichtig**: Reihenfolge und Typen müssen beim Lesen stimmen!
* Ideal für Bilder, Konfigurationen, Netzwerkprotokolle.

**NetworkStream**

Ein NetworkStream ist ein Stream, der Daten über ein Netzwerk sendet oder empfängt – also zum Beispiel über TCP. Er ist Teil der Netzwerkprogrammierung mit Sockets und funktioniert wie andere Streams (z. B. FileStream, MemoryStream), nur eben für Netzwerkverbindungen.

**Zweck von NetworkStream:**

* Bietet **Stream-Zugriff** auf eine Netzwerkverbindung (z. B. TCP)
* Ermöglicht das Lesen und Schreiben mit StreamReader, StreamWriter, BinaryReader, BinaryWriter, etc.
* Wird meist in Verbindung mit einem **TcpClient** oder **TcpListener** verwendet

**Übungen**

**Aufgabe 1 – MemoryStream**

Schreiben Sie einen beliebigen Text in einen MemoryStream und lesen Sie ihn direkt danach wieder aus.

Erwarteter Ablauf:

MemoryStream, StreamWriter, Flush, Position, StreamReader

**Aufgabe 2 – Datei speichern**

Schreiben Sie die Elemente eines Arrays (string[]) mit StreamWriter in eine Datei und wenden Sie dabei die StreamWriter-Methode WriteLine an.

Lesen Sie die Datei zeilenweise ein und geben Sie jede Zeile mit Zeilennummer aus.

**Aufgabe 3 – Binärdaten**

Schreiben Sie 3 verschiedene Datentypen (z. B. int, double, string) in eine .bin-Datei.  
Lesen Sie diese korrekt wieder aus und geben Sie sie aus.

|  |  |
| --- | --- |
| Klasse | Verwendung |
| Stream | Abstrakte Basisklasse |
| MemoryStream | RAM-Speicher |
| FileStream | Dateisystem |
| StreamWriter | Text schreiben |
| StreamReader | Text lesen |
| BinaryWriter | Binär schreiben |
| BinaryReader | Binär lesen |