Simpel tekst bestand

Het is mogelijk om in C# eenvoudige tekst bestanden (plain text) op te slaan en op te vragen. In deze les wordt gekeken hoe die gedaan kan worden. Er zijn twee manieren om dit te doen:

1. Met de StreamWriter/StreamReader Class
2. Met Serilization

StreamReader

De eenvoudigste manier om een tekst bestand te maken en op te vragen is met de StreamWriter en StreamReader Classes. Je hebt dan controle over hoe het bestand wordt opgebouwd.

Voor het gebruik van deze classes heb je de System.IO bibliotheek nodig. Deze moet met de using opdracht toegevoegd worden want standaard is het niet aanwezig bij het aan maken van een nieuw project.

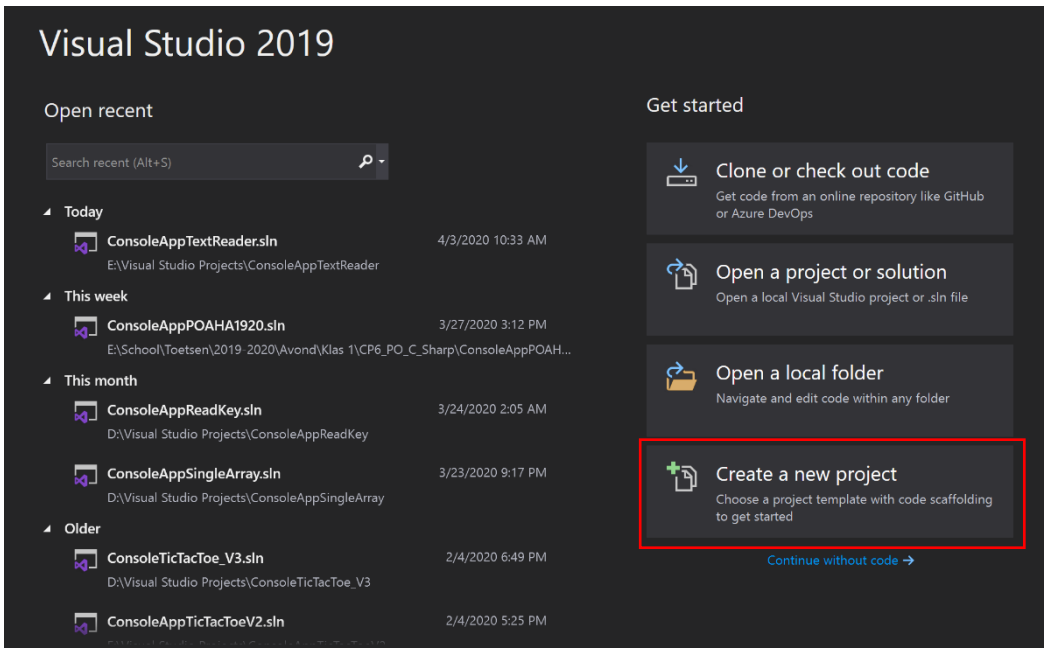
Stap 1

Voor dit voorbeeld, maak een simpel tekst bestand en Notepad, Notepad++ of Visual Studio Code, op een Windows computer (*voor Mac gebruikers gebruik een editor waarmee plain text bestanden gemaakt kunnen worden. Bijvoorbeeld Visual Studio Code, Sublime Tekst, TextWrangler, etc.*).
Sla dit bestand op in de root van de C: drive met de bestandsnaam Example.txt.

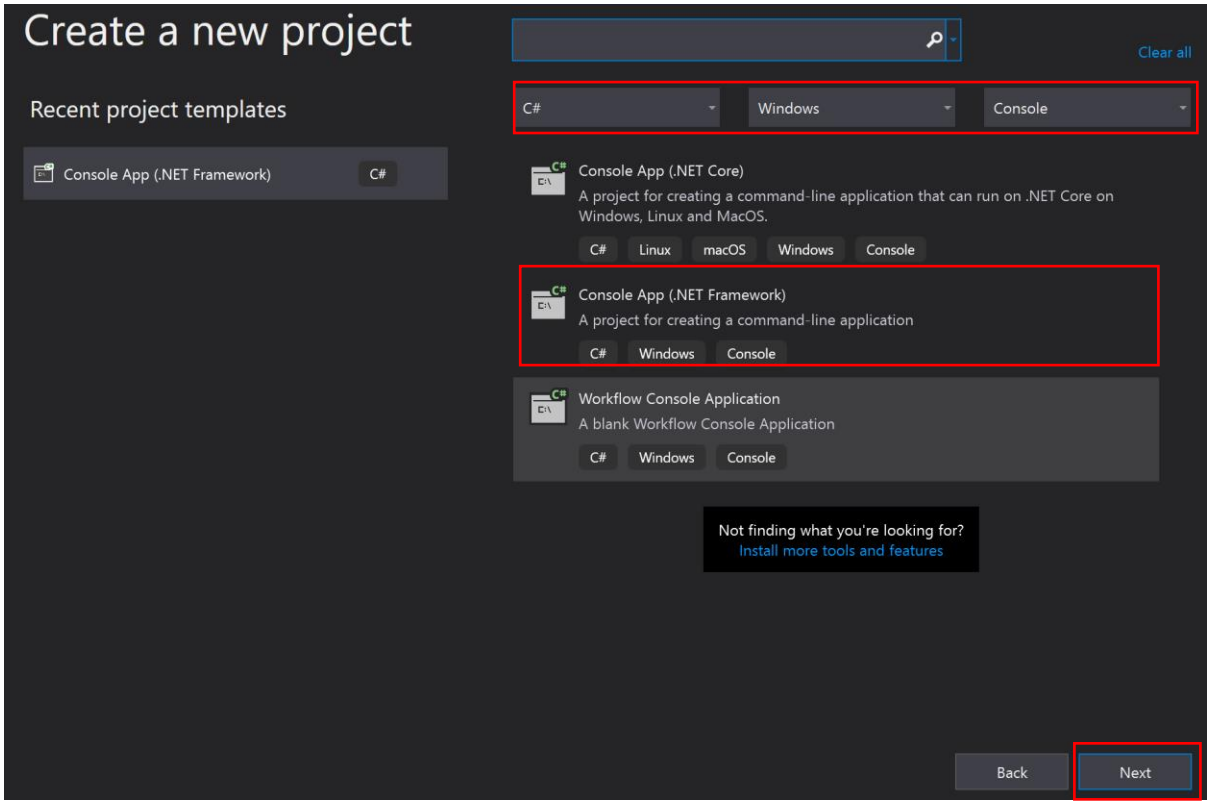
Stap 2

Start Visual Studio 2019 community of gebruik dotnetfiddle.net website.

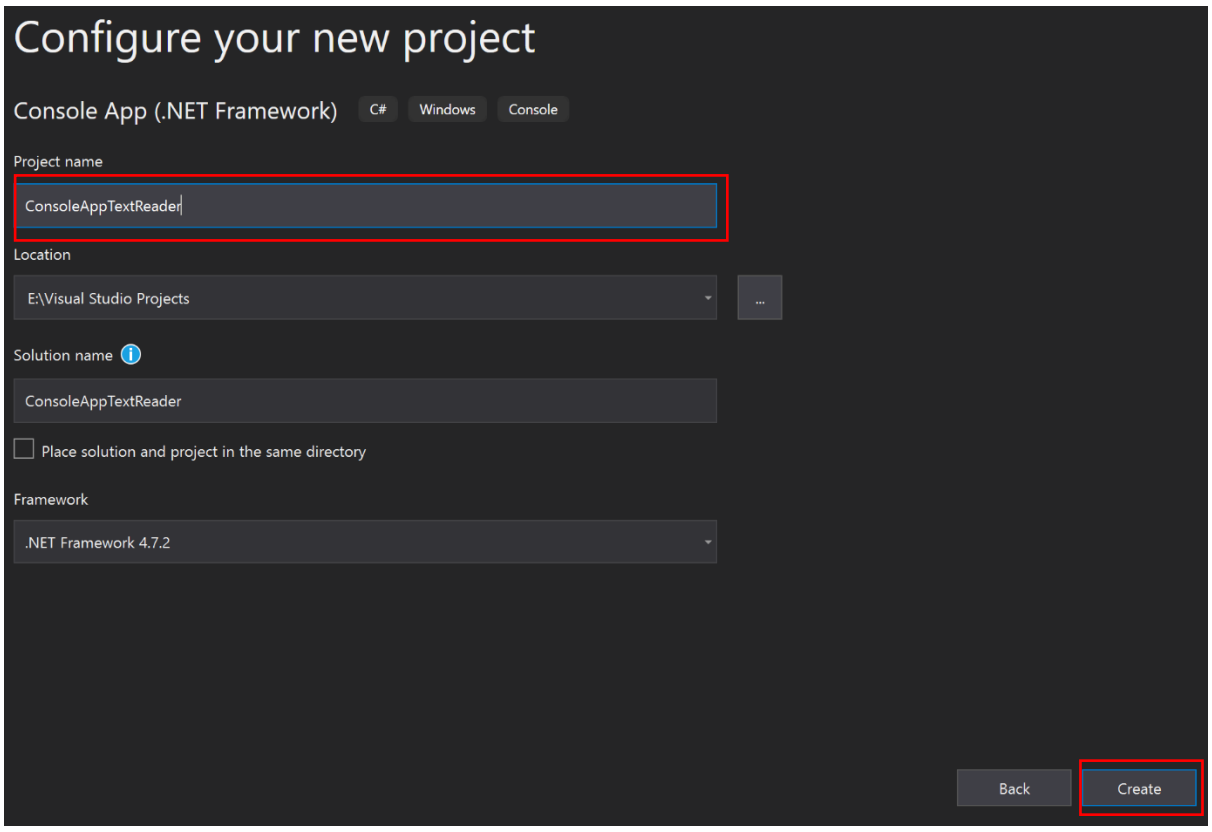
Bij Visual Studio 2019 maak een nieuwe project aan.



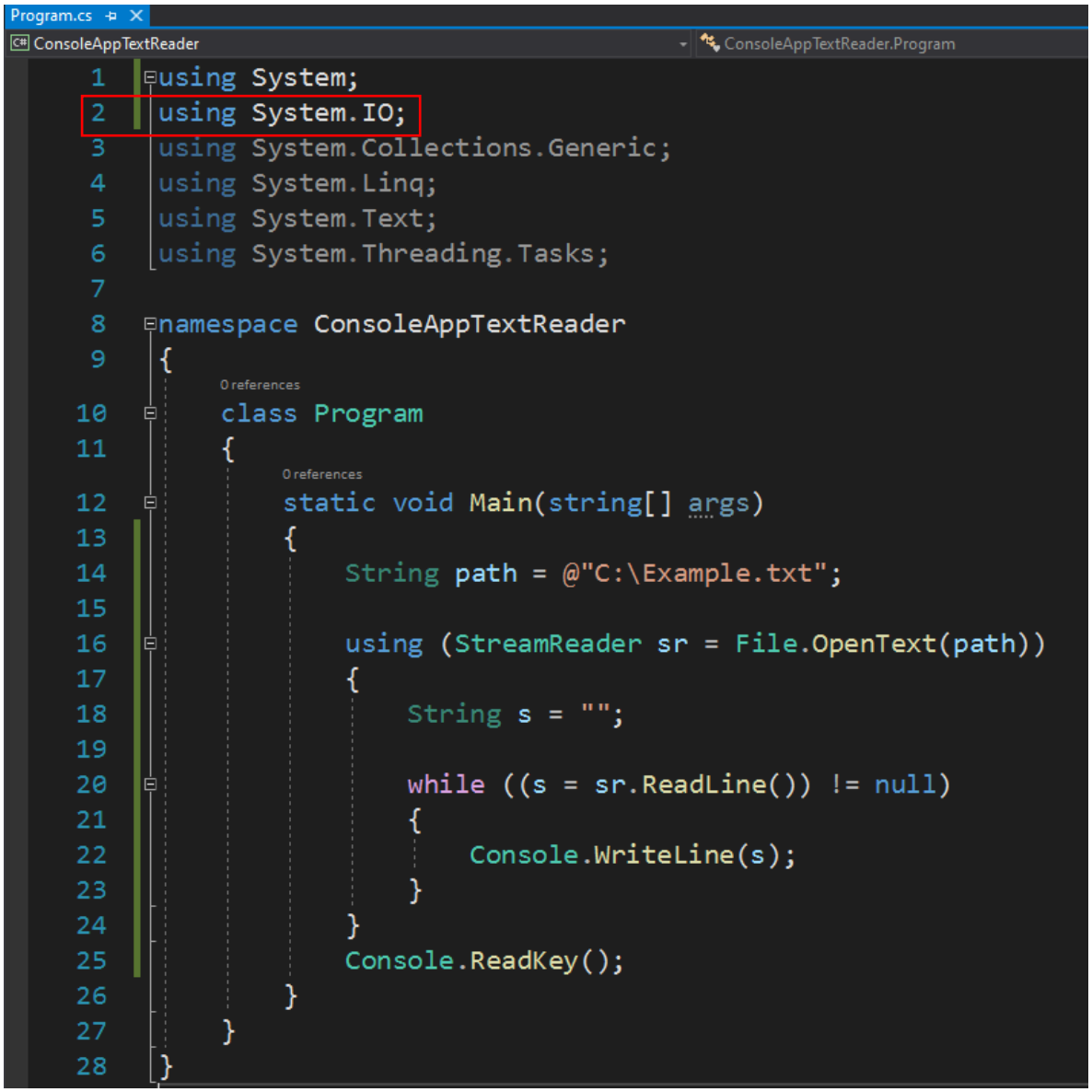
Kies als type project een **Console App (.NET Framework)** project. Je kunt de filters gebruiken om het sneller te vinden.



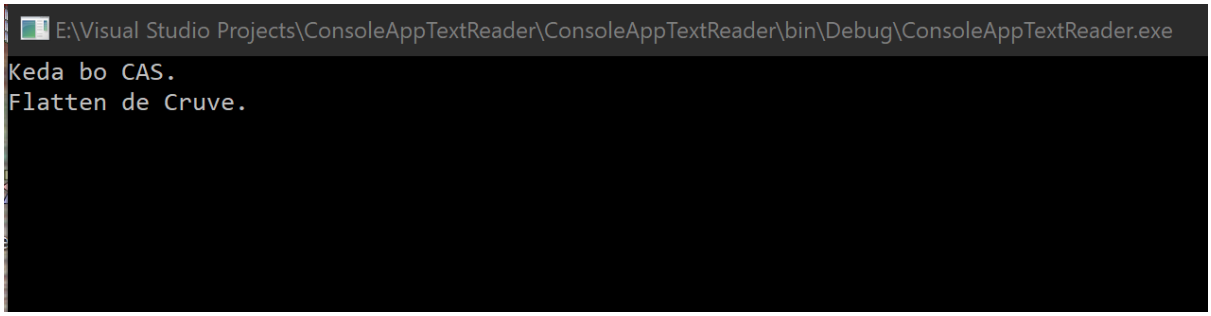
Klik op de **Next** knop. In het volgende venster vul je de naam in van het programma (**ConsoleAppTextReader**) en druk op de knop **Create**.



In het code venster voer je de code in zoals in de afbeelding hier onder. **Let op** dat de extra regel: using System.IO; wordt toegevoegd bovenin.



Sla de code op en voer het uit. De uitvoer wordt getoond in het **Command Prompt** venster van Windows. Bijvoorbeeld:



Wat gebeurt er in de code:

In regel 14 wordt er een pad gedefinieerd waar straks de reader het tekst bestand gaat lezen.

```
14            String path = @"C:\Example.txt";
```

Dit zou dus ook een variabele kunnen zijn waar een string in staat met het pad waar het bestand staat dat gelezen moet worden.

Regel 16 start met het openen en lezen van het tekst bestand op het aangegeven pad.

```
16            using (StreamReader sr = File.OpenText(path))
```

De **Streamer** gaat nu regel voor regel het bestand lezen tot dat alle regels gelezen zijn en het einde van het bestand (EOF = End Of File) bereikt is.

Elke regel kan nu in een variabele of meerdere variabele geplaatst worden om verder gebruikt te worden in het programma. In dit voorbeeld wordt er een lokale variabele '**s**' aangemaakt van het type **string**.

In een while lus worden de regels van bestand één voor één in gelezen en gecontroleerd of ze niet leeg (**null**) zijn. Als dat het geval is wordt de regel in het Console venster afgedrukt.

StreamWriter

De StreamWriter doet exact het omgekeerde van wat de StreamReader doet.

Stap 1.
Maak weer een nieuw project aan in Visual Studio 2019 of de dotnetfiddle.net website.

Kies opnieuw als type project een **Console App (.NET Framework)** project.

Noem het project ConsoleAppTextWriter en voer de code van de afbeelding in.

Let er weer op dat bovenaan de extra regel: using System.IO; wordt toegevoegd.

```
1   using System;
2   using System.IO;
3   using System.Collections.Generic;
4   using System.Linq;
5   using System.Text;
6   using System.Threading.Tasks;
7
8   namespace ConsoleAppTextWriter
9   {
        0 references
10      class Program
11      {
            0 references
12          static void Main(string[] args)
13          {
14              String path = @"C:\Example2.txt";
15
16              using (StreamWriter sr = File.AppendText(path))
17              {
18
19                  sr.WriteLine("Laga nos reboot 2020.");
20                  sr.WriteLine("Ban spera cu humanidad ta sinja y 2021 ta bay ta Bon.");
21                  sr.Close();
22
23                  Console.WriteLine(File.ReadAllText(path));
24              }
25              Console.ReadKey();
26          }
27      }
28  }
```

Wat gebeurt er in de code:

In regel 14 wordt het pad en de naam van het tekstbestand (Example2.txt) bepaald waar de StreamWriter object het tekstbestand moet gaan opslaan.

```
14              String path = @"C:\Example2.txt";
```

In regel 16 wordt de StreamWriter gestart en opent het een tekstbestand op het aangegeven pad.

In de regels 18 en 19 worden twee tekst regels toegevoegd aan het bestand

In regel 20 wordt het bestand afgesloten met een EOF aanduiding.

In regel 22 wordt de inhoud van het bestand opnieuw gelezen en getoond in het Console venster.

Opmerkingen:

In regel 16 wordt gebruik gemaakt van de AppendText Method van het File object. Dit zorgt ervoor dat er meerdere regels aan het bestand kunnen worden toegevoegd. Stel je zou dit programma meerdere keren

uitvoeren dan wordt meerdere keren dezelfde regels aan het bestand toegevoegd.

StreamWriter en StreamReader Object

Je kunt in de C# code ook een object maken van StreamWriter en StreamReader. Tekst bestanden kunnen dan eenvoudig opgebouwd worden met de instructies Write, WriteLine.

Bekijk de onderstaande code regel

```
1022            System.IO.StreamWriter file = new System.IO.StreamWriter(filename+".plz");
1023
1024            // Save total amount of words in  wordlist
1025            file.WriteLine(wordList.Count);
1026
1027            // Save WordList
1028            foreach (string word in wordList)
1029            {
1030                file.WriteLine(word);
1031            }
```

In regel 1022 wordt een object **file** aangemaakt die de StreamWriter opend met een bestandsnaam dat in de variabele filename staat en daar aan wordt de extensie .plz toegevoegd.

In de foreach-lus wordt nu door alle elementen van een lijst gelopen en met de instructie file.WriteLine(word); weg geschreven naar het tekstbestand.

In dit voorbeeld wordt geen Append gebruikt dus als er een tekstbestand was dan wordt deze nu overschreven.

Om zo'n tekstbestand terug te kunnen lezen wordt een object gemaakt voor de StreamReader. Bekijk de code.

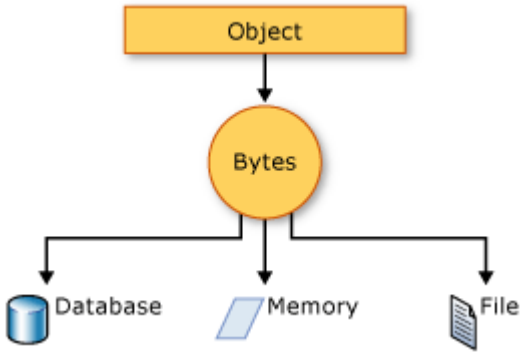
```
1083            System.IO.StreamReader file = new System.IO.StreamReader(filename + ".plz");
1084
1085            // Load total amount of words in wordList
1086            wordListCount = Convert.ToInt16(file.ReadLine());
1087
1088            // Load to WordList
1089            for (int word = 0; word <= (wordListCount - 1); word++)
1090            {
1091                //wordList[word] = file.ReadLine();
1092                strDummy = file.ReadLine();
1093                wordList.Add(strDummy);
1094            }
```

In plaats van WriteLine wordt nu ReadLine gebruikt. Merk ook op dat waar waarden van een andere type zijn dan tekst, deze opnieuw naar de correcte waarden moeten worden omgezet (cast), bijvoorbeeld in regel 1086.

Door gebruik te maken van de object methode is het mogelijk om een eigen bestands structuur op te bouwen.

Serialization

Serialisatie is het proces waarbij een object in een vorm wordt gebracht die op de stream kan worden geschreven. Het is het proces waarbij het object wordt omgezet in een formulier zodat het kan worden opgeslagen in een bestand, database of geheugen; of het kan via het netwerk worden overgedragen. Het belangrijkste doel is om de staat van het object op te slaan, zodat het indien nodig opnieuw kan worden gemaakt.



Deserialization

Zoals de naam al doet vermoeden, is het omgekeerde proces van serialisatie. Het is het proces van het terughalen van het geserialiseerde object zodat het in het geheugen kan worden geladen. Het brengt de toestand van het object weer tot leven door eigenschappen, velden enz. In te stellen.

Typen:

- Binary Serialization
- XML Serialization
- JSON Serialization

Gebruikt voor serialisatie

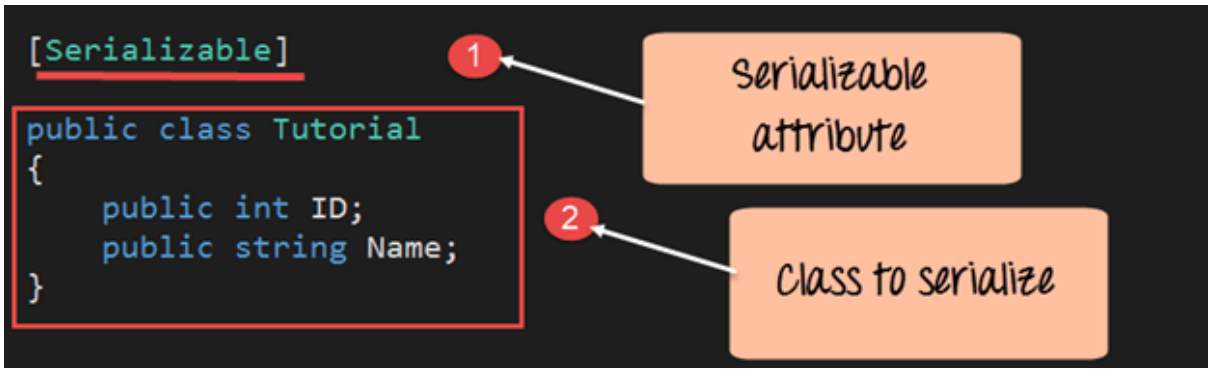
Serialisatie stelt de ontwikkelaar in staat om de status van een object op te slaan en indien nodig opnieuw te creëren, met opslag van objecten en gegevensuitwisseling. Door serialisatie kan een ontwikkelaar acties uitvoeren zoals:

- Het object verzenden naar een externe applicatie met behulp van een webservice.
- Een object doorgeven van het ene domein naar het andere.
- Het doorgeven van een object door een firewall als een JSON- of XML-string.
- Behoud van beveiliging of gebruikersspecifieke informatie over applicaties heen.

Wanneer je in C# met objecten werk is het meestal handig om je object data doormiddel van serialisatie extern op te slaan en met deserialisatie op te vragen.

Voorbeeld van serialisatie en deserialisatie

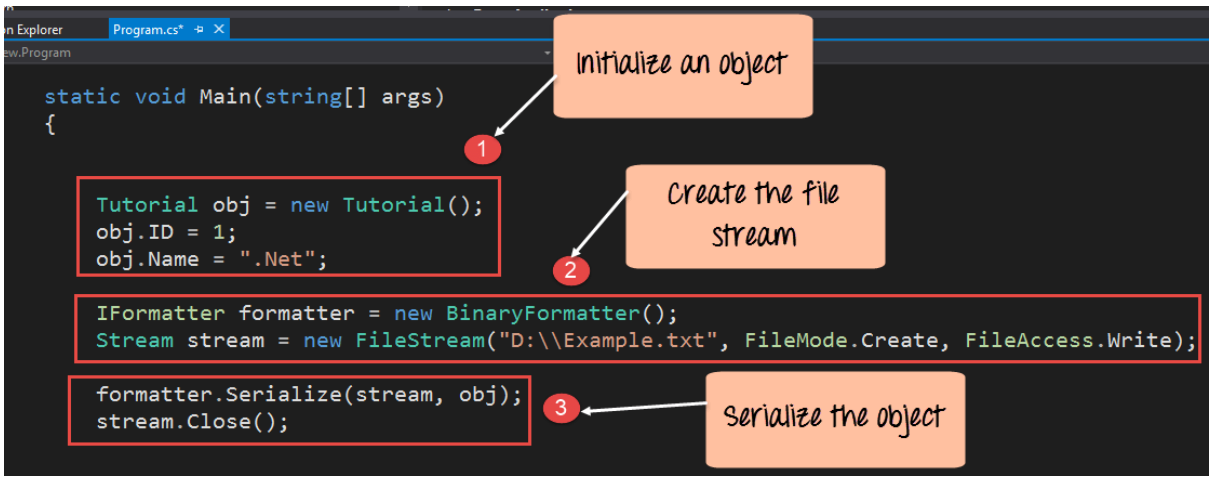
Stap 1
Voer de onderstaande code in in het program.cs-bestand van de consoletoepassing. De eerste stap is het toevoegen van de klasse die zal worden gebruikt voor serialisatie.



Code Uitleg

1. De klasse die moet worden geserialiseerd, moet het kenmerk **[Serializable]** hebben. Dit is een trefwoord in C #. Dit trefwoord wordt vervolgens toegevoegd aan de Klasse Tutorial. Als u dit kenmerk niet vermeldt, krijgt u een foutmelding wanneer u de klasse probeert te serialiseren.
2. Hierna volgt de definitie van de klasse die zal worden geserialiseerd. Hier definiëren we een klasse met de naam "Tutorial" en bieden we 2 eigenschappen, één is "ID" en de andere is "Naam".

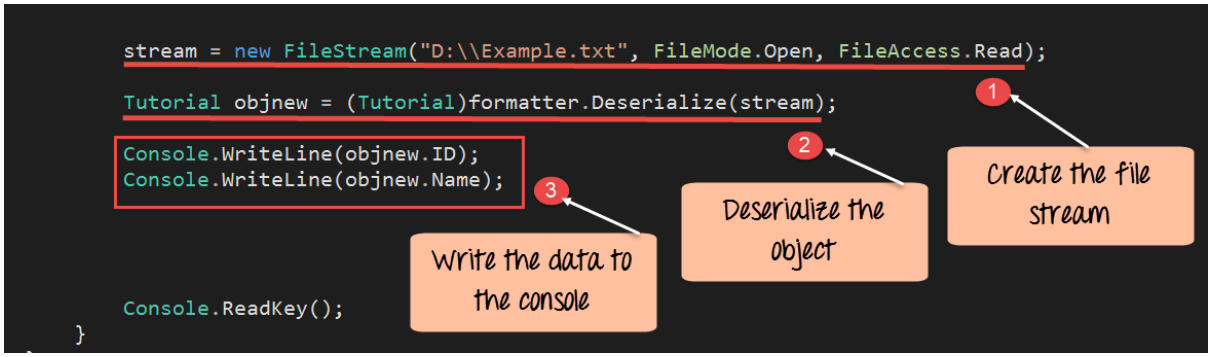
Stap 2
In deze stap maken we eerst het object van de klasse Tutorial en serialiseren we het naar het bestand met de naam Example.txt



Code Uitleg:

1. Eerst maken we een object van de klasse Tutorial. Vervolgens wijzen we de waarde "1" toe aan ID en ".net" aan de eigenschap name.
2. Vervolgens gebruiken we de formatter-klasse die wordt gebruikt om het object te serialiseren of om te zetten naar een binair formaat. De gegevens in het bestand in serialisatie worden gedaan in binair formaat. Vervolgens maken we een bestandsstroomobject. Het bestandsstroomobject wordt gebruikt om het bestand Voorbeeld.txt te openen voor schrijfdoeleinden. De sleutelwoorden FileMode.Create en FileMode.Write wordt gebruikt om specifiek te vermelden dat het bestand moet worden geopend voor schrijfdoeleinden.
3. Ten slotte gebruiken we de Serialize-methode om de binaire gegevens naar het bestand over te brengen. Vervolgens sluiten we de stream, omdat de schrijfbewerking is voltooid.

Stap 3



Om er zeker van te zijn dat de gegevens in het bestand aanwezig zijn, gebruiken we deserialisatie om het object uit het bestand te deserialiseren.

Code voorbeeld

Hier onder is een uitgewerkt voorbeeld van hoe men serialization en deserialization zou kunnen gebruiken in C#.

Let op dat bij het using deel een aantal regels moeten worden toegevoegd aan de standaardlijst die Visual Studio er al automatisch plaatst.

Deze regels zijn:

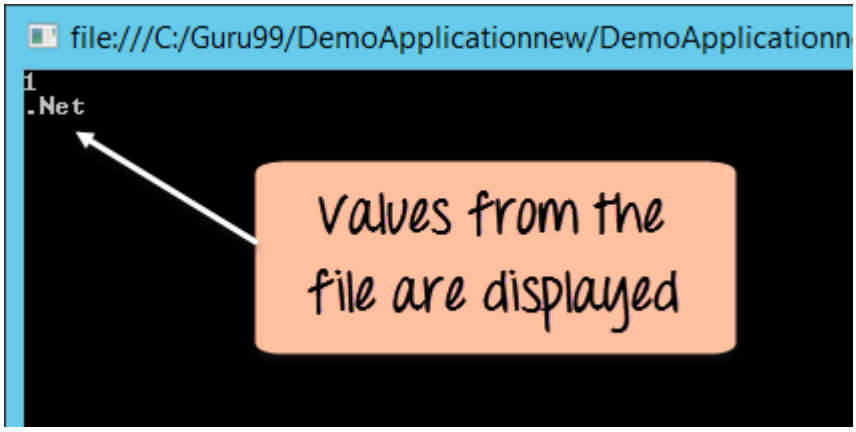
1. using System.IO;
2. using System.Runtime.Serialization;
3. using System.Runtime.Serialization.Formatters.Binary;

```
using System;
using System.IO;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
using System.Text;
using System.Threading.Tasks;

namespace ConsoleAppSerialDeSerial
{
    [Serializable]
    class Tutorial
    {
        public int ID;
        public String Name;
        static void Main(string[] args)
        {
            Tutorial obj = new Tutorial();
            obj.ID = 1;
            obj.Name = ".Net";

            IFormatter formatter = new BinaryFormatter();
            Stream stream = new FileStream(@"D:\ExampleNew.txt", FileMode.Create,
FileAccess.Write);

            formatter.Serialize(stream, obj);
            stream.Close();

            stream = new FileStream(@"D:\ExampleNew.txt", FileMode.Open,
FileAccess.Read);
            Tutorial objnew = (Tutorial)formatter.Deserialize(stream);

            Console.WriteLine(objnew.ID);
            Console.WriteLine(objnew.Name);

            Console.ReadKey();
        }
    }
}
```

Code Uitleg:

1. We creëren het object "stream" om het bestand Example.txt te openen in alleen-lezen modus.
2. Vervolgens gebruiken we de klasse formatter die wordt gebruikt om het object te deserialiseren, dat is opgeslagen in het bestand Example.txt. Het geretourneerde object is ingesteld op het object objnew.
3. Ten slotte tonen we de eigenschappen van het object "objnew" aan de console met de eigenschappen "ID" en "naam".

Wanneer de bovenstaande code is ingesteld en het project wordt uitgevoerd met Visual Studio, krijgt u de onderstaande uitvoer.



U kunt aan de bovenstaande uitvoer zien dat de waarden uit het bestand correct zijn gedeserialiseerd en weergegeven in de console.

Mocht je een foutmelding krijgen dan kan het zijn dat de rechten niet voldoende zijn om op die drive tektst op te slaan of misschien heb is die drive niet aanwezig op de computer. Oplossing is om de drive letter aan te passen. De code hier gaat uit van een D-drive.

Samenvatting

Serialisatie wordt gebruikt om klasseobjecten naar bestanden te schrijven.

De-serialisatie wordt gebruikt om de objecten uit het bestand te herstellen.