# Read and write to files

2 minutes

100 XP

Reading data from files and writing data to files are core concepts in .NET.

Tailwind Traders needs to write the total of all the individual store sales files to a new file. Then, load the file into the company's sales system.

Here, you learn how to use the `File` class to read and write to files.

## Read data from files

Files are read through the `ReadAllText` method on the `File` class.

C#

Copy

```csharp
File.ReadAllText($"stores{Path.DirectorySeparatorChar}201{Path.DirectorySeparatorChar}sales.json");
```

The return object from `ReadAllText` is a string.

JSON

Copy

```json
{
  "total": 22385.32
}
```

## Parse data in files

This data in its string format doesn't do you much good. It's still just characters, but now in a format that you can read. You want the ability to parse this data into a format that you can use programmatically.

There are many ways to parse JSON files with .NET, including a community library known as *Json.NET*.

You can add the *Json.NET* package to your project by using NuGet:

Bash

Copy

```bash
dotnet add package Newtonsoft.Json
```

Then, add `using Newtonsoft.Json` to the top of your class file:

C#

Copy

```csharp
using Newtonsoft.Json;
```

And use the `JsonConvert.DeserializeObject` method:

C#

Copy

```csharp
var salesJson = File.ReadAllText($"stores{Path.DirectorySeparatorChar}201{Path.DirectorySeparatorChar}sales.json");
var salesData = JsonConvert.DeserializeObject<SalesTotal>(salesJson);

Console.WriteLine(salesData.Total);

class SalesTotal
{
  public double Total { get; set; }
}
```

Tip

Files come in a variety of formats. JSON files are the most desirable to work with because of the built-in support in the language. You also might encounter files that are .csv, fixed width, or some other format. In that case, it's best to search nuget.org for a parser for that file type.

## Write data to files

You learned how to write files in the previous exercise; it's just that you wrote an empty one. To write data to a file, use the same `WriteAllText` method, but pass in the data that you want to write.

C#

Copy

```csharp
var data = JsonConvert.DeserializeObject<SalesTotal>(salesJson);

File.WriteAllText($"salesTotalDir{Path.DirectorySeparatorChar}totals.txt", data.Total.ToString());

// totals.txt
// 22385.32
```

## Append data to files

In the preceding example, the file is overwritten every time you write to it. Sometimes, you don't want that. You want to append data to the file, instead of replacing it entirely. You can append data with the `File.AppendAllText` method. By default, `File.AppendAllText` creates the file if it doesn't already exist.

C#

Copy

```csharp
var data = JsonConvert.DeserializeObject<SalesTotal>(salesJson);

File.AppendAllText($"salesTotalDir{Path.DirectorySeparatorChar}totals.txt", $"{data.Total}{Environment.NewLine}");

// totals.txt
// 22385.32
// 22385.32
```

**Tip**

In the preceding code example, `Environment.NewLine` prompts .NET to put the value on its own line. If you didn't pass this value, you would get all the numbers squished together on the same line.

In the next exercise, you'll finish the sales-total project for Tailwind Traders by reading all the sales files and writing the grand total to a .txt file. The company's commerce system can then process the file.