Exercise - Read and write to files

7 minutes

100 XP

You can also use the `File` class in .NET to write data to files and read data from files.

You're almost finished creating a .NET masterpiece for Tailwind Traders. So far, your code reads any directory, finds all .json files, and creates a *totals.txt* file.

In this exercise, you complete the project by reading the .json files, adding up the store totals, and writing the grand total to the *totals.txt* file.

Add Json.NET to the project

1. Using the terminal, add *Json.NET* to the project.

   Bash

   Copy

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

Preparation for sales data

1. At the top of `Program.cs`, add `using Newtonsoft.Json`:

   C#

   Copy

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

2. In `Program.cs` directly under the `FindFiles` method, add a new record that models the *sales.json* data:

   C#

   Copy

   ```csharp
   record SalesData (double Total);
   ```

Create a method to calculate sales totals

1. In `Program.cs`, just before the `record` line that you added in the previous step, create a new function that calculates the sales total. This method should take an `IEnumerable<string>` of file paths that it can iterate over.

   C#

   Copy

   ```csharp
   double CalculateSalesTotal(IEnumerable<string> salesFiles)
   {
       double salesTotal = 0;

       // READ FILES LOOP

       return salesTotal;
   }
   ```

2. Within that method, replace `// READ FILES LOOP` with a loop that iterates over the `salesFiles`, reads the file, parses the content as JSON, and then increments the `salesTotal` variable with the `total` value from the file:

   C#

   Copy

   ```csharp
   double CalculateSalesTotal(IEnumerable<string> salesFiles)
   {
       double salesTotal = 0;

       // Loop over each file path in salesFiles
       foreach (var file in salesFiles)
       {
           // Read the contents of the file
           string salesJson = File.ReadAllText(file);

           // Parse the contents as JSON
           SalesData? data = JsonConvert.DeserializeObject<SalesData?>(salesJson);
   ```

```
        // Add the amount found in the Total field to the salesTotal variable
        salesTotal += data?.Total ?? 0;
    }

    return salesTotal;
}
```

Call the CalculateSalesTotals method

1. In the `Program.cs` file, add a call to the `CalculateSalesTotal` function just above the `File.WriteAllText` call:

   C# Copy

   ```
   var currentDirectory = Directory.GetCurrentDirectory();
   var storesDir = Path.Combine(currentDirectory, "stores");

   var salesTotalDir = Path.Combine(currentDirectory, "salesTotalDir");
   Directory.CreateDirectory(salesTotalDir);

   var salesFiles = FindFiles(storesDir);

   var salesTotal = CalculateSalesTotal(salesFiles); // Add this line of code

   File.WriteAllText(Path.Combine(salesTotalDir, "totals.txt"), String.Empty);
   ```

Write the total to the totals.txt file

1. In the `Program.cs` file, modify the `File.WriteAllText` block to write the value of the `salesTotal` variable to the *totals.txt* file. And while you're at it, change the `File.WriteAllText` call to `File.AppendAllText` so nothing in the file gets overwritten.

   C# Copy

   ```
   var currentDirectory = Directory.GetCurrentDirectory();
   var storesDir = Path.Combine(currentDirectory, "stores");

   var salesTotalDir = Path.Combine(currentDirectory, "salesTotalDir");
   Directory.CreateDirectory(salesTotalDir);

   var salesFiles = FindFiles(storesDir);

   var salesTotal = CalculateSalesTotal(salesFiles);

   File.AppendAllText(Path.Combine(salesTotalDir, "totals.txt"), $"{salesTotal}{Environment.NewLine}");
   ```

2. Press `Ctrl+S` / `Cmd+S` to save the *Program.cs* file.

Run the program

1. Run the program from the terminal:

   Bash Copy

   ```
   dotnet run
   ```

   There's no output from the program. If you look in the *salesTotalDir/totals.txt* file, you find the total of all the sales from the *sales.json* file.

2. Run the program from the terminal again.

   Bash Copy

   ```
   dotnet run
   ```

3. Select the *salesTotalDir/totals.txt* file.

   The *totals.txt* file now has a second line. Every time you run the program, the totals are added up again and a new line is written to the file.

Outstanding work! You've written a smart, robust, and handy tool that Tailwind Traders can use to process all of its stores' sales every night. In the next unit, we'll review what you learned and a few tips to remember.

Got stuck?

If you got stuck during this exercise, here's the full code for this project:

C# Copy

```csharp
using Newtonsoft.Json;

var currentDirectory = Directory.GetCurrentDirectory();
var storesDirectory = Path.Combine(currentDirectory, "stores");

var salesTotalDir = Path.Combine(currentDirectory, "salesTotalDir");
Directory.CreateDirectory(salesTotalDir);

var salesFiles = FindFiles(storesDirectory);

var salesTotal = CalculateSalesTotal(salesFiles);

File.AppendAllText(Path.Combine(salesTotalDir, "totals.txt"), $"{salesTotal}{Environment.NewLine}");

IEnumerable<string> FindFiles(string folderName)
{
    List<string> salesFiles = new List<string>();

    var foundFiles = Directory.EnumerateFiles(folderName, "*", SearchOption.AllDirectories);

    foreach (var file in foundFiles)
    {
        var extension = Path.GetExtension(file);
        if (extension == ".json")
        {
            salesFiles.Add(file);
        }
    }

    return salesFiles;
}

double CalculateSalesTotal(IEnumerable<string> salesFiles)
{
    double salesTotal = 0;

    // Loop over each file path in salesFiles
    foreach (var file in salesFiles)
    {
        // Read the contents of the file
        string salesJson = File.ReadAllText(file);

        // Parse the contents as JSON
        SalesData? data = JsonConvert.DeserializeObject<SalesData?>(salesJson);

        // Add the amount found in the Total field to the salesTotal variable
        salesTotal += data?.Total ?? 0;
    }

    return salesTotal;
}

record SalesData (double Total);
```