Writing Text File in C#

Introduction

In the world of programming, writing text files is a fundamental skill that plays a crucial role in various tasks. Whether you're storing logs, saving user data, or generating reports, the ability to write text files efficiently and effectively is essential.

One of the key components in C# for writing text files is the StreamWriter class. This class provides a convenient and intuitive way to write text to files. In this lesson, we will explore the StreamWriter class and learn how to utilize its features to create and write to text files.

To begin, we will dive into the steps involved in writing to a text file using the traditional C-style file I/O functions - fopen(), fprintf(), and fclose(). Understanding these steps will give us a solid foundation and a deeper appreciation of the StreamWriter class.

Additionally, we will explore different file access modes, such as "w" (write mode), "a" (append mode), "r+" (read and write mode), "w+" (write and read mode), and "a+" (append and read mode). Each mode has specific characteristics that dictate how files can be accessed and modified.

By the end of this lesson, you will have a comprehensive understanding of the importance of writing text files in programming tasks, the capabilities of the StreamWriter class, the steps involved in writing to a text file using C-style functions, and the different file access modes available in C#. Armed with this knowledge, you will be able to confidently write text files, manipulate data, and accomplish a wide range of programming tasks. So, let's dive in and unlock the power of writing text files in C#!

Objectives

The objectives of this lesson are to help learners understand, learn, practice, and apply the skills required for writing text files in C#. By the end of this lesson, you will be able to confidently write text files, manipulate data, and accomplish a wide range of programming tasks involving text file operations.

1. Understand the Importance of Text File Writing: Grasp the significance of writing text files in programming, recognizing its role in data persistence, configuration management, and inter-application communication.

2. Learn the StreamWriter Class: Familiarize yourself with the StreamWriter class in C#, comprehending its functions and capabilities for simplifying text file writing operations.

3. Practice Basic File Operations (fopen(), fprintf(), fclose()): Engage in practical exercises to reinforce the foundational concepts of file writing using traditional methods like fopen(), fprintf(), and fclose(). Gain hands-on experience with these fundamental file handling functions.

4. Apply File Access Modes in C#: Explore and apply various file access modes ("w," "a," "r+," "w+," "a+," etc.) in C# to tailor file operations according to specific needs. Understand how these modes influence writing, appending, and reading data from text files.

Source code example

```
1.  using System;
2.  using System.IO;
3.
4.  class Program
5.  {
6.      static void Main()
7.      {
8.          Console.WriteLine("iNetTutor.com - Write File Example");
9.          Console.WriteLine("Enter the text you want to save:");
10.
11.         string text = Console.ReadLine();
12.
13.         Console.WriteLine("Enter the file name (without extension):");
14.         string fileName = Console.ReadLine();
15.
16.         // Append ".txt" extension if not provided
17.         if (!fileName.EndsWith(".txt"))
18.         {
19.             fileName += ".txt";
20.         }
21.
22.         Console.WriteLine("Choose an option:");
23.         Console.WriteLine("1. Create a new file and write the text");
24.         Console.WriteLine("2. Append the text to an existing file");
25.         Console.WriteLine("3. Overwrite an existing file");
26.
27.         int option = Convert.ToInt32(Console.ReadLine());
28.
29.         switch (option)
30.         {
31.             case 1:
32.                 WriteToFile(fileName, text);
33.                 Console.WriteLine($"Text saved to a new file '{Path.GetFullPath(
    fileName)}' successfully!");
34.                 break;
35.             case 2:
36.                 AppendToFile(fileName, text);
37.                 Console.WriteLine($"Text appended to the file '{Path.GetFullPath
    (fileName)}' successfully!");
38.                 break;
39.             case 3:
40.                 OverwriteFile(fileName, text);
41.                 Console.WriteLine($"Text overwritten in the file '{Path.GetFullP
    ath(fileName)}' successfully!");
42.                 break;
43.             default:
44.                 Console.WriteLine("Invalid option!");
45.                 break;
46.         }
```

```
47.         Console.ReadKey();
48.     }
49.
50.     static void WriteToFile(string fileName, string text)
51.     {
52.         using (StreamWriter writer = new StreamWriter(fileName))
53.         {
54.             writer.Write(text);
55.         }
56.     }
57.
58.     static void AppendToFile(string fileName, string text)
59.     {
60.         using (StreamWriter writer = new StreamWriter(fileName, append: true))
61.         {
62.             writer.Write(text);
63.         }
64.     }
65.
66.     static void OverwriteFile(string fileName, string text)
67.     {
68.         using (StreamWriter writer = new StreamWriter(fileName, append: false))
69.         {
70.             writer.Write(text);
71.         }
72.     }
73. }
```

Explanation

This code demonstrates how to interactively write text to a file, offering flexibility for different file operations based on user input.

Here's a breakdown of what it does:

1. Introduction:
   - Prints a welcome message and prompts the user to enter the text they want to save.
   - Asks for the file name without the extension.
2. Handling File Extension:
   - Checks if the user-provided name ends with ".txt".
   - If not, appends the ".txt" extension to ensure proper file format.
3. User Choice:
   - Presents three options:
     - Create a new file and write the text.
     - Append the text to an existing file.
     - Overwrite an existing file.
   - Reads the user's choice as an integer.
4. Processing User Choice:
   - Uses a switch statement to handle each option:
     - WriteToFile: Creates a new file and writes the text using StreamWriter.

- AppendToFile: Opens the file in append mode and writes the text, adding it to the existing content.
- OverwriteFile: Opens the file without append mode, effectively overwriting the existing content with the new text.
- Default case handles invalid options.

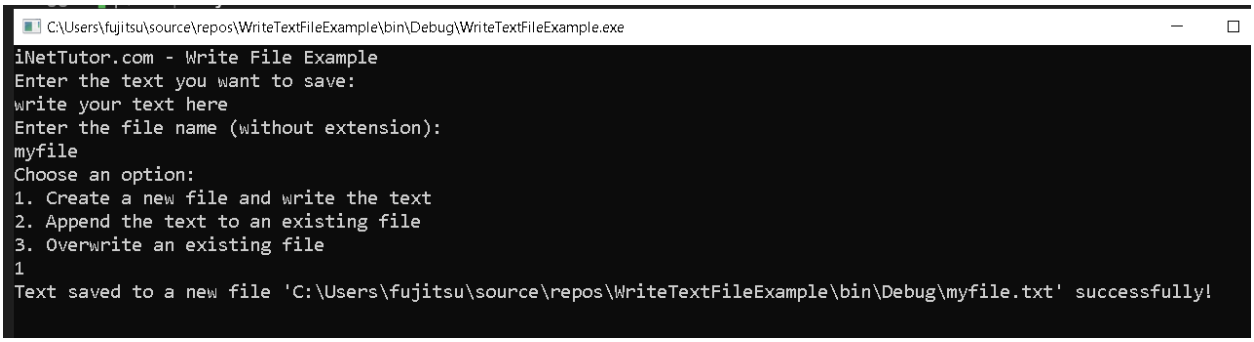
5. Success Messages and Closing:
   - Prints a success message with the full path of the file where the text was saved or modified.
   - Waits for the user to press any key before exiting.

Additional Notes:

- The code uses using statements to automatically close the StreamWriter objects, ensuring proper resource management.
- The Path.GetFullPath method is used to obtain the absolute path of the file, making it more user-friendly and independent of the current working directory.

Overall, this code provides a well-structured and user-friendly solution for writing text to files in C#, with clear guidance and feedback for the user throughout the process.

Output



Summary

This C# lesson guides users in creating a program that interactsively writes text to a file. Users can input the desired text and choose from options to create a new file, append to an existing file, or overwrite content in an existing file. The program ensures file names have the ".txt" extension, enhancing user-friendly functionality. It utilizes StreamWriter for file operations, offering a comprehensive example of file handling in C#. The feedback messages provide clear indications of successful operations, fostering a practical understanding of text file manipulation within the context of user-driven applications.

Exercises and Assessment

Exercises:

1. File Validation: Enhance the program to validate if the entered file name is valid (e.g., no invalid characters, restricted file names).

2. User Confirmation: Implement a feature to ask the user for confirmation before executing any file operations, preventing accidental overwrites.

3. Error Handling: Introduce robust error handling mechanisms to gracefully handle unexpected scenarios, such as invalid user inputs or file access issues.

4. Menu Refinement: Refine the user interface by providing a more intuitive menu system, perhaps using a switch statement for better readability and user experience.

Assessment:

Create a scenario-based assessment where learners are given specific requirements to modify the program. Evaluate their ability to implement changes effectively, considering both functionality and code readability.

1. **Scenario:** Assume the user wants to save files in different formats (e.g., .txt, .csv). Instruct learners to extend the program to support different file formats dynamically.

2. **Criteria:**

    - Correct implementation of file format handling.
    - Proper integration without compromising existing functionality.
    - Clear and concise code documentation.

3. **Feedback:** Provide constructive feedback on the code structure, adherence to best practices, and the effectiveness of the implemented features.

Lab Exam with Rubrics:
Design a lab exam that assesses students' comprehension and application skills in working with file manipulation in C#. Include a set of tasks with specific requirements and allocate points based on predefined rubrics.

1. Task 1: File Validation (20 points)
    - Requirement: Implement file name validation to ensure it adheres to specific criteria.
    - Rubrics: Points awarded for accuracy, handling edge cases, and maintaining program integrity.
2. Task 2: Confirmation Mechanism (15 points)
    - Requirement: Add a user confirmation step before executing file operations.
    - Rubrics: Points for correct implementation, user-friendly interface, and preventing accidental data loss.
3. Task 3: Error Handling (25 points)
    - Requirement: Implement robust error handling to address potential issues during file operations.

- Rubrics: Points for effectiveness in error identification, user-friendly error messages, and maintaining program stability.

4. Task 4: Menu Refinement (15 points)
    - Requirement: Refine the user interface using a switch statement for menu options.
    - Rubrics: Points for improved readability, logical organization, and user-centric design.
5. Task 5: Comprehensive Scenario (25 points)
    - Requirement: Combine previous tasks into a comprehensive scenario, such as saving and appending data to different file formats.
    - Rubrics: Points for successful integration, maintaining code clarity, and meeting specific scenario requirements.

Quiz

1. What method is used to create a new file for writing?
   a) OpenText
   b) WriteFile
   c) StreamWriter
   d) File.ReadAllText

2. Which option appends text to an existing file?
   a) open("file.txt", "w")
   b) open("file.txt", "r")
   c) open("file.txt", "a")
   d) open("file.txt", "r+")

3. How can you check if a file already exists using C#?
   a) File.Exists("file.txt")
   b) StreamWriter.CheckFile("file.txt")
   c) File.Open("file.txt", FileMode.Open)
   d) Path.GetFullPath("file.txt")

4. What is the purpose of using a using statement with StreamWriter?
   a) To enforce proper indentation.
   b) To ensure automatic file closing.
   c) To improve code readability.
   d) To check for writing errors.

5. Which is NOT a good practice for writing to text files?
   a) Handling potential exceptions like file not found.
   b) Closing the StreamWriter object after writing.
   c) Using absolute file paths instead of relative paths.
   d) Checking if the user has write permissions before writing.

Meta Description

"Master file manipulation in C#! Learn to create, append, and overwrite text files with user-friendly options. Explore error handling and best practices."