Working with Files, Folders, Text, and Paths in C#

1. Overview

C# provides powerful classes under the `System.IO` namespace for interacting with the file system. This includes creating, reading, writing, copying, deleting, and navigating files and directories.

2. Namespaces and Classes

`System.IO` Namespace includes:

- **File** – static methods to work with files.
- **FileInfo** – instance methods for file operations.
- **Directory** – static methods for folder operations.
- **DirectoryInfo** – instance methods for directory operations.
- **Path** – helps in handling file and directory paths.
- **StreamReader/StreamWriter** – for reading and writing text.
- **FileStream** – for byte-level file operations.

3. Working with Files

Checking if File Exists

```
if (File.Exists("file.txt"))
{
    Console.WriteLine("File exists.");
}
```

Creating a File

```
File.Create("example.txt").Close(); // Closes to release the file
```

Deleting a File

```
File.Delete("example.txt");
```

Copying and Moving

```
File.Copy("source.txt", "copy.txt", overwrite: true);
File.Move("copy.txt", "moved.txt");
```

4. Reading and Writing Text

Writing to a Text File

```
File.WriteAllText("file.txt", "Hello, world!");
```

Reading from a Text File

```
string content = File.ReadAllText("file.txt");
Console.WriteLine(content);
```

Using `StreamWriter`

```
using (StreamWriter writer = new StreamWriter("file.txt", append: true))
{
    writer.WriteLine("More text!");
}
```

Using `StreamReader`

```
using (StreamReader reader = new StreamReader("file.txt"))
{
    string line;
    while ((line = reader.ReadLine()) != null)
    {
        Console.WriteLine(line);
    }
}
```

5. Working with Directories

Creating a Directory

```
Directory.CreateDirectory("MyFolder");
```

Deleting a Directory

```
Directory.Delete("MyFolder", recursive: true);
```

Listing Files and Folders

```
string[] files = Directory.GetFiles("MyFolder");
string[] folders = Directory.GetDirectories("MyFolder");
```

6. Path Manipulation with `Path` Class

Common Methods

```
string fileName = Path.GetFileName("C:\\folder\\file.txt");          // "file.txt"
string dirName = Path.GetDirectoryName("C:\\folder\\file.txt");      // "C:\\folder"
string extension = Path.GetExtension("file.txt");                    // ".txt"
string combined = Path.Combine("folder", "file.txt");               // "folder\\file.txt"
string fullPath = Path.GetFullPath("file.txt");
```

7. File Attributes and Info

Using `FileInfo`

```
FileInfo fileInfo = new FileInfo("file.txt");

Console.WriteLine(fileInfo.Length); // Size in bytes
Console.WriteLine(fileInfo.CreationTime);
```

Using `FileAttributes`

```
File.SetAttributes("file.txt", FileAttributes.ReadOnly);
var attributes = File.GetAttributes("file.txt");
Console.WriteLine(attributes);
```

8. Exception Handling

Always use `try-catch` blocks for file operations:

```
try
{
    File.WriteAllText("file.txt", "Data");
}
catch (IOException ex)
{
    Console.WriteLine("File error: " + ex.Message);
}
```

Tips & Best Practices

- Always close file streams (use `using` block).
- Check file/directory existence before operations.
- Use `Path.Combine` instead of manual path concatenation.
- Avoid hardcoded paths; use relative paths or configuration.

Sample Case Study

Task: Create a log file that stores every time a program is run.

```
string logPath = "log.txt";
string logMessage = $"Program run at {DateTime.Now}";

using (StreamWriter sw = new StreamWriter(logPath, true))
{
    sw.WriteLine(logMessage);
}
```

Q & A