# C# Examples - [3] File Class

You are a C# tutor. Explain the File class in C# for a developer with around 3 years of experience. Cover the following in detail:

Concepts:

What is the File class, its purpose, and when to use it.

Explain how it differs from using FileStream and StreamReader/StreamWriter.

Real-world scenarios where using the File class is more convenient.

Must-know methods and properties:

Methods like File.Create, File.Delete, File.Exists, File.Copy, File.Move, File.ReadAllText, File.WriteAllText, File.ReadAllLines, File.AppendAllText.

For each method, provide a small, simple C# code example with output.

Differences:

Compare FileStream vs StreamReader/StreamWriter vs File class in terms of:

Binary vs text operations

Ease of use

Performance

Flexibility

Real-world usage scenarios

Practical example:

Provide a complete, commented C# program demonstrating:

Creating, writing, reading, appending, and deleting a file using the File class

How the same operations would look using FileStream and StreamWriter/StreamReader

Include expected outputs for clarity.

Best practices and interview insights:

Show proper exception handling, efficient usage, and key points to remember for interviews or real projects.

Important: Include small, simple, working C# code examples and outputs for every method, property, and concept so it's easy to understand and practical for a 3-year experienced .NET developer.

Absolutely! Let's break this down systematically. Since you have around **3 years of C# experience**, I'll focus on practical insights, real-world scenarios, and comparisons with examples. I'll also include outputs for clarity.

---

## 1. Concepts: File Class in C#

The `File` class is part of the `System.IO` namespace. It provides **static methods** for creating, reading, writing, copying, deleting, and moving files. Unlike `FileStream` or `StreamReader/StreamWriter`, you don't need to manually manage streams most of the time— `File` simplifies common file operations.

**When to use `File` class:**

- Quick file operations like reading or writing small text files.
- When you don't need fine-grained control over buffering or file streams.
- For tasks like checking existence, copying, moving, deleting, or appending files.

**Key difference vs FileStream/StreamReader/StreamWriter:**

| Feature | FileStream | StreamReader/Writer | File class |
|---|---|---|---|
| Operation | Binary or text | Text only | Text or binary (simplified) |
| Ease of use | Low-level | Moderate | High-level |
| Performance | Best for large data | Good for text files | Good for small to medium files |
| Flexibility | High | Moderate | Low (but quick for common tasks) |
| Requires disposing | Yes | Yes | Mostly handled internally |

**Real-world scenarios for File class:**

- Logging small files (`ReadAllText`, `AppendAllText`).
- Copying config files (`Copy` and `Move`).
- Quick read/write operations in utilities, scripts, or setup apps.

---

## 2. Must-know Methods with Examples

### 1. File.Create

Creates a new file. Overwrites if exists.

csharp

```csharp
using System;
using System.IO;

class Program {
    static void Main() {
        string path = "example.txt";
        File.Create(path).Close(); // Create file and close stream
        Console.WriteLine(File.Exists(path) ? "File created" : "File not created");
    }
}
```

**Output:**

```arduino
File created
```

---

## 2. File.Delete

Deletes a file.

```csharp
File.Delete("example.txt");
Console.WriteLine(File.Exists("example.txt") ? "Exists" : "Deleted");
```

**Output:**

```nginx
Deleted
```

---

## 3. File.Exists

Checks if a file exists.

```csharp
bool exists = File.Exists("example.txt");
Console.WriteLine(exists ? "Exists" : "Does not exist");
```

---

## 4. File.Copy

Copies a file to another location.

```csharp
File.WriteAllText("source.txt", "Hello World");
File.Copy("source.txt", "dest.txt", true);
Console.WriteLine(File.ReadAllText("dest.txt"));
```

**Output:**

```nginx
Hello World
```

---

## 5. File.Move

Moves/renames a file.

```csharp
File.Move("dest.txt", "moved.txt");
Console.WriteLine(File.Exists("moved.txt") ? "File moved" : "Move failed");
```

**Output:**

```arduino
File moved
```

---

## 6. File.ReadAllText

Reads entire file content as string.

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

**Output:**

```nginx
Hello World
```

---

## 7. File.WriteAllText

Writes a string to a file, overwriting if exists.

```csharp
```

```
File.WriteAllText("source.txt", "New Content");
Console.WriteLine(File.ReadAllText("source.txt"));
```

**Output:**

```
css

New Content
```

---

## 8. File.ReadAllLines

Reads all lines into a string array.

```
csharp

File.WriteAllText("lines.txt", "Line1\nLine2\nLine3");
string[] lines = File.ReadAllLines("lines.txt");
foreach (var line in lines) Console.WriteLine(line);
```

**Output:**

```
nginx

Line1
Line2
Line3
```

---

## 9. File.AppendAllText

Appends text to an existing file.

```
csharp

File.AppendAllText("lines.txt", "\nLine4");
Console.WriteLine(File.ReadAllText("lines.txt"));
```

**Output:**

```
nginx
```

```
Line1
Line2
Line3
Line4
```

---

## 3. Comparison: FileStream vs StreamReader/StreamWriter vs File

| Feature | FileStream | StreamReader/Writer | File class |
|---|---|---|---|
| Type of operation | Binary & text | Text only | Text & binary |
| Usage complexity | Manual buffer handling | Easier, encoding handled | Very simple |
| Memory use | Efficient for large files | Efficient | Good for small-medium files |
| Real-world example | Large video/image | Config/text logs | Quick scripts, utilities |

---

## 4. Practical Example: File Operations Using All Three

csharp

```csharp
using System;
using System.IO;

class Program {
    static void Main() {
        string filePath = "test.txt";

        // ===== Using File class =====
        File.WriteAllText(filePath, "Hello File Class!");
        File.AppendAllText(filePath, "\nAppend using File class");
        Console.WriteLine("File class content:");
        Console.WriteLine(File.ReadAllText(filePath));

        // ===== Using StreamWriter =====
        using (StreamWriter sw = new StreamWriter("stream.txt")) {
```

```
            sw.WriteLine("Hello StreamWriter");
            sw.WriteLine("Another line");
        }
        Console.WriteLine("\nStreamWriter content:");
        using (StreamReader sr = new StreamReader("stream.txt")) {
            Console.WriteLine(sr.ReadToEnd());
        }

        // ===== Using FileStream =====
        using (FileStream fs = new FileStream("filestream.txt", FileMode.Create,
FileAccess.Write)) {
            byte[] data = System.Text.Encoding.UTF8.GetBytes("Hello FileStream");
            fs.Write(data, 0, data.Length);
        }
        using (FileStream fs = new FileStream("filestream.txt", FileMode.Open, FileAccess.Read))
{
            byte[] buffer = new byte[fs.Length];
            fs.Read(buffer, 0, buffer.Length);
            Console.WriteLine("\nFileStream content:");
            Console.WriteLine(System.Text.Encoding.UTF8.GetString(buffer));
        }

        // Cleanup
        File.Delete(filePath);
        File.Delete("stream.txt");
        File.Delete("filestream.txt");
    }
}
```

**Output:**

```
arduino

File class content:
Hello File Class!
Append using File class

StreamWriter content:
Hello StreamWriter
Another line
```

```
FileStream content:
Hello FileStream
```

---

## 5. Best Practices & Interview Insights

1. **Exception Handling:** Always use try-catch or `using` to properly close streams.
2. **File.Exists:** Check before deleting, copying, or moving.
3. **Use File class:** For simple file tasks; saves code and reduces errors.
4. **StreamReader/Writer:** Use when reading/writing large text files or line-by-line processing.
5. **FileStream:** Use for large files, binary operations, or custom buffering.

**Interview Tip:**
Be ready to explain **why you might choose `File` over `StreamWriter` or `FileStream`**, and vice versa. Usually, `File` is for simplicity; `FileStream` / `StreamReader/Writer` for flexibility and performance.

---

If you want, I can create a **quick visual summary table with all `File` class methods, their inputs/outputs, and equivalent StreamReader/Writer/FileStream code** in one sheet—perfect for reference and interviews.

Do you want me to do that next?