C# Examples - [1] FileStream in C#

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

Concepts:

What is a FileStream, its purpose, when to use it, and real-world scenarios.

Include stream types, reading/writing modes, buffering, and disposal patterns.

Must-know methods and properties:

Explain the commonly used methods (Read, Write, Seek, Flush, Close) and properties (CanRead, CanWrite, Length, Position).

For each method/property, provide a small, simple code example with output so it’s easy to understand.

Best practices:

Proper usage patterns, exception handling, using using statements, performance considerations.

Explain the difference between FileStream and other file handling approaches like the File class.

Practical example:

Provide a complete, commented C# program demonstrating:

Reading from a file

Writing to a file

Seeking positions

Checking capabilities (CanRead, CanWrite)

Proper disposal (using statement)

Include expected output for clarity.

Interview/real-world insight:

Highlight the key points a 3-year experienced .NET developer should know for interviews or real-world projects.

Important: For every concept or method/property, include small, simple, working C# code examples and their outputs to make understanding easy.

Absolutely! Let's dive deep into **FileStream** in C# with a focus for a developer with ~3 years experience. I'll cover concepts, must-know methods/properties, best practices, and give practical examples with outputs. Everything will be simple, clear, and interview-ready.

---

1. FileStream Concepts

What is a FileStream?

A `FileStream` is a class in `System.IO` namespace used for reading from and writing to files **as a stream of bytes**. Unlike the `File` class, which is higher-level, `FileStream` gives more control over file operations, such as buffering, seeking, and file access modes.

Purpose:

- Read and write binary data.
- Access large files efficiently.
- Control file pointer positions.
- Support synchronous and asynchronous operations.

When to use:

- Reading/writing large files in chunks.
- Binary file operations (images, videos, logs).

- Fine-grained control over file access (seek, flush, buffer).

Stream types:

- **FileStream**: File-based byte stream.
- **MemoryStream**: In-memory byte stream.
- **NetworkStream**: Stream over network sockets.

File access modes (FileMode, FileAccess, FileShare):

Enum	Purpose
FileMode	How to open the file (Create, Open, Append, etc.)
FileAccess	Permissions (Read, Write, ReadWrite)
FileShare	How other processes can access the file while it's open

Buffering:
`FileStream` uses an internal buffer to improve performance when reading/writing small amounts of data repeatedly.

Disposal pattern:
Always dispose `FileStream` using `using` statement to free OS resources and avoid file locks.

---

2. Must-Know Methods and Properties

Properties

1. **CanRead**

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

class Program {
    static void Main() {
        using (FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.ReadWrite)) {
```

```
        Console.WriteLine("CanRead: " + fs.CanRead);
    }
  }
}
```

Output:

```
vbnet

CanRead: True
```

2. **CanWrite**

```
csharp

using (FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.Read)) {
    Console.WriteLine("CanWrite: " + fs.CanWrite);
}
```

Output:

```
vbnet

CanWrite: False
```

3. **Length**

```
csharp

using (FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.ReadWrite)) {
    Console.WriteLine("File Length: " + fs.Length);
}
```

4. **Position**

```
csharp

using (FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.ReadWrite)) {
    Console.WriteLine("Current Position: " + fs.Position);
    fs.Position = 5; // move pointer to 5th byte
```

```
    Console.WriteLine("New Position: " + fs.Position);
}
```

Methods

1. Read(byte[] buffer, int offset, int count)

```csharp
using (FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.Read)) {
    byte[] buffer = new byte[fs.Length];
    int bytesRead = fs.Read(buffer, 0, buffer.Length);
    Console.WriteLine("Read bytes: " + bytesRead);
    Console.WriteLine("Data: " + System.Text.Encoding.UTF8.GetString(buffer));
}
```

2. Write(byte[] buffer, int offset, int count)

```csharp
using (FileStream fs = new FileStream("sample.txt", FileMode.Create, FileAccess.Write)) {
    byte[] data = System.Text.Encoding.UTF8.GetBytes("Hello FileStream!");
    fs.Write(data, 0, data.Length);
    Console.WriteLine("Data written successfully.");
}
```

3. Seek(long offset, SeekOrigin origin)

```csharp
using (FileStream fs = new FileStream("sample.txt", FileMode.Open, FileAccess.Read)) {
    fs.Seek(6, SeekOrigin.Begin); // move to 7th byte
    byte[] buffer = new byte[5];
    fs.Read(buffer, 0, buffer.Length);
    Console.WriteLine("Data after seek: " + System.Text.Encoding.UTF8.GetString(buffer));
}
```

4. Flush()

```csharp
```

```
using (FileStream fs = new FileStream("sample.txt", FileMode.Append, FileAccess.Write)) {
    byte[] data = System.Text.Encoding.UTF8.GetBytes(" Appending text.");
    fs.Write(data, 0, data.Length);
    fs.Flush(); // ensures data is written to disk
    Console.WriteLine("Data flushed to disk.");
}
```

5. **Close()**
   Automatically handled by `using` statement, but can be called manually:

```
csharp

FileStream fs = new FileStream("sample.txt", FileMode.OpenOrCreate, FileAccess.ReadWrite);
fs.Close();
```

---

3. Best Practices

- Use `using` **statements** to automatically dispose `FileStream`.
- Always check `CanRead` or `CanWrite` before performing operations.
- Use **buffering** to improve performance.
- Handle exceptions (`IOException`, `UnauthorizedAccessException`) when accessing files.
- Prefer `FileStream` over `File` class when you need:
  - Random access (`Seek`)
  - Binary operations
  - Large file handling

FileStream vs File class:

Feature	FileStream	File class
Control	High (seek, flush, buffer)	Limited
Data type	Bytes	Strings or byte arrays
Use case	Large or binary files	Simple read/write operations

---

4. Complete Practical Example

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

class FileStreamDemo {
    static void Main() {
        string filePath = "demo.txt";

        // Writing to file
        using (FileStream fs = new FileStream(filePath, FileMode.Create, FileAccess.Write)) {
            byte[] data = Encoding.UTF8.GetBytes("Hello FileStream Example!");
            fs.Write(data, 0, data.Length);
            fs.Flush(); // ensure data is written
            Console.WriteLine("Data written to file.");
        }

        // Reading from file
        using (FileStream fs = new FileStream(filePath, FileMode.Open, FileAccess.Read)) {
            Console.WriteLine($"CanRead: {fs.CanRead}, CanWrite: {fs.CanWrite}");
            byte[] buffer = new byte[fs.Length];
            fs.Read(buffer, 0, buffer.Length);
            Console.WriteLine("Read Data: " + Encoding.UTF8.GetString(buffer));

            // Seeking
            fs.Seek(6, SeekOrigin.Begin);
            byte[] seekBuffer = new byte[9];
            fs.Read(seekBuffer, 0, seekBuffer.Length);
            Console.WriteLine("Data after seek: " + Encoding.UTF8.GetString(seekBuffer));
        }
    }
}
```

Expected Output:

```yaml
Data written to file.
CanRead: True, CanWrite: False
```

```
Read Data: Hello FileStream Example!
Data after seek: FileStream
```

---

5. Interview / Real-World Insight

- **Know when to use FileStream vs File class.**
- **Understand buffering, seeking, and file access modes.**
- **Always dispose streams** to avoid file locks.
- **Understand common exceptions** like `IOException`.
- Can optimize performance for **large files or binary operations**.

✅ **Summary:**
A 3-year experienced C# developer should **master FileStream basics**, know **key methods/properties**, handle **resources properly**, and be able to **read/write files efficiently** in both real-world and interview scenarios.

If you want, I can create a **visual diagram showing FileStream workflow with reading, writing, seeking, and disposing**—this helps retention and interview prep.

Do you want me to create that diagram?