C# Examples - [2] StreamWriter/Reader

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

Concepts:

What are StreamWriter and StreamReader, their purpose, and real-world scenarios where they are used.

Explain writing vs reading text files, buffering, encoding options, and disposal patterns.

Must-know methods and properties:

For StreamWriter: methods like Write, WriteLine, Flush, Close and properties like Encoding, BaseStream.

For StreamReader: methods like Read, ReadLine, ReadToEnd, Peek, Close and properties like BaseStream, EndOfStream.

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

Differences with FileStream:

Compare FileStream vs StreamWriter/StreamReader in terms of functionality, use cases, text vs binary, and performance considerations.

Practical example:

Provide a complete, commented C# program demonstrating:

Writing text to a file using StreamWriter

Reading text from a file using StreamReader

Using multiple methods (WriteLine, ReadLine, ReadToEnd)

Proper disposal using using statements

Include expected output for clarity.

Best practices and interview insights:

Show proper exception handling, efficient usage, and differences to remember for interviews or real-world projects.

Important: Include small, simple working code examples and outputs for every method, property, and concept so understanding is easy and practical.

Absolutely! Let’s dive deep into **StreamWriter** and **StreamReader** in C# with practical examples for a developer with ~3 years experience. I’ll cover concepts, must-know methods & properties, differences with `FileStream`, a complete working example, and interview/practical insights.

---

1. Concepts

StreamWriter

- **Purpose:** Write text to files efficiently.
- **Use-case:** Saving logs, writing configuration files, exporting CSVs, or any text output.
- **Key Features:**
  - Buffered writing (more efficient than writing one char at a time)
  - Supports encoding (UTF-8, ASCII, etc.)
  - Can append or overwrite files
  - Works specifically with text, not raw binary data

StreamReader

- **Purpose:** Read text from files efficiently.
- **Use-case:** Reading configuration, log files, or any text data.

- **Key Features:**
  - Buffered reading
  - Supports encoding
  - Provides convenient methods to read line by line or the entire content
  - Can peek at the next character without advancing the stream

Writing vs Reading Text Files

Aspect	StreamWriter	StreamReader
Purpose	Writing text	Reading text
Data type	Text	Text
Buffering	Writes in memory buffer, flushes to file	Reads into buffer from file for efficiency
Encoding	Supports ASCII, UTF-8, UTF-16	Supports ASCII, UTF-8, UTF-16
Disposal	Needs `Close()` or `using`	Needs `Close()` or `using`

Disposal Pattern

Always wrap streams in a `using` block or explicitly call `Close()` to release file handles:

```csharp
using (StreamWriter writer = new StreamWriter("file.txt"))
{
    writer.WriteLine("Hello");
} // Automatically calls Dispose() → closes file
```

---

2. Must-Know Methods & Properties

StreamWriter

Method/Property	Description	Example
Write()	Writes text without a newline	`writer.Write("Hello");`
WriteLine()	Writes text with a newline	`writer.WriteLine("Hello World");`
Flush()	Clears buffer and writes remaining data to file	`writer.Flush();`
Close()	Closes the stream	`writer.Close();`
Encoding	Gets encoding used by StreamWriter	`Console.WriteLine(writer.Encoding);`
BaseStream	Gets underlying stream	`Console.WriteLine(writer.BaseStream.Length);`

Example:

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

class Program
{
    static void Main()
    {
        using (StreamWriter writer = new StreamWriter("example.txt"))
        {
            writer.Write("Hello "); // no newline
            writer.WriteLine("World"); // adds newline
            writer.Flush(); // ensure data is written
            Console.WriteLine("Encoding: " + writer.Encoding);
            Console.WriteLine("BaseStream length: " + writer.BaseStream.Length);
        }
    }
}
```

Output (`example.txt`):

```vbnet
Hello World
Console:
Encoding: System.Text.UTF8Encoding
BaseStream length: 11
```

StreamReader

Method/Property	Description	Example
Read()	Reads next char as int	`int c = reader.Read();`
ReadLine()	Reads next line	`string line = reader.ReadLine();`
ReadToEnd()	Reads entire file	`string all = reader.ReadToEnd();`
Peek()	Returns next char without advancing	`int nextChar = reader.Peek();`
Close()	Closes stream	`reader.Close();`
BaseStream	Gets underlying stream	`reader.BaseStream.Position = 0;`
EndOfStream	Checks if reached end	`while(!reader.EndOfStream)`

Example:

```csharp

 using System;
using System.IO;

class Program
{
    static void Main()
    {
        using (StreamReader reader = new StreamReader("example.txt"))
        {
            Console.WriteLine("Peek next char: " + (char)reader.Peek());
            Console.WriteLine("Read one char: " + (char)reader.Read());
            Console.WriteLine("Read one line: " + reader.ReadLine());
            reader.BaseStream.Position = 0; // rewind
            Console.WriteLine("Read entire file:\n" + reader.ReadToEnd());
        }
    }
}
```

Output:

```sql
Peek next char: H
Read one char: H
Read one line: ello World
Read entire file:
Hello World
```

---

3. FileStream vs StreamWriter/StreamReader

Aspect	FileStream	StreamWriter/StreamReader
Type	Binary stream	Text-oriented
Purpose	Read/write bytes	Read/write text
Ease of Use	Requires manual encoding for text	Handles text encoding automatically
Buffering	Minimal, you need manual buffering	Buffered internally by default
Use Cases	Images, PDFs, any binary	Text files, logs, CSV, configs
Performance	Slightly faster for raw binary	Efficient for text because of buffering

Takeaway: Use **FileStream** for binary, **StreamReader/Writer** for text.

---

4. Complete Practical Example

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

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

        // Writing text using StreamWriter
        using (StreamWriter writer = new StreamWriter(filePath))
        {
            writer.WriteLine("Line 1: Hello StreamWriter!");
            writer.WriteLine("Line 2: Writing another line.");
            writer.Flush(); // optional, will auto-flush at Dispose
        }

        // Reading text using StreamReader
        using (StreamReader reader = new StreamReader(filePath))
        {
            Console.WriteLine("Reading line by line:");
            while (!reader.EndOfStream)
            {
                Console.WriteLine(reader.ReadLine());
            }

            // Rewind to beginning
            reader.BaseStream.Position = 0;
            Console.WriteLine("\nReading entire file with ReadToEnd:");
            Console.WriteLine(reader.ReadToEnd());
        }
    }
}
```

Expected Output:

```yaml

Reading line by line:
Line 1: Hello StreamWriter!
Line 2: Writing another line.

Reading entire file with ReadToEnd:
Line 1: Hello StreamWriter!
Line 2: Writing another line.
```

---

5. Best Practices & Interview Insights

1. **Always use `using` blocks** to automatically release resources.
2. **Use `StreamWriter` / `StreamReader` for text**; `FileStream` for binary.
3. **Buffering matters**: default buffering is usually efficient; manual flush only when necessary.
4. **Encoding**: specify encoding when needed (`new StreamWriter(filePath, false, Encoding.UTF8)`).
5. **Avoid mixing FileStream + StreamReader unless you need special control.**
6. **Interview Tips:**
    - Know the difference: FileStream vs StreamReader/Writer.
    - Know methods like `Peek`, `EndOfStream`.
    - Understand disposal pattern and exceptions (file locks).

If you want, I can also make a **side-by-side table of all methods of StreamWriter and StreamReader with small one-line example + output**—super handy for quick interview prep.

Do you want me to make that cheat sheet?