🎏 **Streams and I/O**

# **Stream Architecture**

A stream is a sequence of bytes that flows in or out of a program. In C#, the **System.IO** provides several types of streams for reading and writing data, such as file streams, memory streams, and network streams.

File Handling

* **FileStream**: Represents a stream that can read from or write to a file on disk.
* **MemoryStream**: Represents a stream that reads from or writes to a block of memory.
* **NetworkStream**: Represents a stream that reads from or writes to a network socket.
* **BufferedStream** : Wraps another stream and provides a buffer for optimizing read and write operations.

Backing Store Streams : Provide raw data

Decorator Streams : Provide transparent binary transformations

Stream Adapters : Typed methods for dealing in higher level types such as string and XML.

## **Using Streams**

Abstract stream class is base for all streams.

Recommended using async method for slow streams ( network I/O )

### **Reading and Writing**

CanWrite return false : streams is read-only

CanRead return false : streams is write-only

Read receives a data block from stream into array

| **byte[] data** = new **byte** [1000]*;* // **bytesRead** will always **end** up at 1000, unless the **stream** is // **itself** smaller in length: int **bytesRead** = 0*;* int chunkSize = 1*;* while (**bytesRead** < **data**.Length && chunkSize > 0)  **bytesRead** +=  chunkSize = s.Read (**data**, **bytesRead, data**.Length - **bytesRead);** |
| --- |

That is correct way to read 1000 byte stream

ReadBye : Read -1 to indicate the end of stream

## **Seeking**

Stream is seekable if CanSeek is true. Seeking is move the position indicator to a specific location within the stream.

Change position in fileStream will take a few microseconds. If doing million times, better using MemoryMappedFile.

## **Closing and Flushing**

Stream must be disposed or released, using Dispose or Close.

## **Timeouts**

Stream support R/W timeout if CanTimeout is true. Network streams support timeout, but file and memory do not.

Async ReadAsync and WriteAsync do not support timeouts, instead you can pass a cancellation token.

### **Thread Safety**

Streams is not thread-safe (can make error when 2 thread r/w same resource).

Must use Synchronized method to wrap any stream in thread-safe wrapper, so only one thread can do operation at a time.

## **Backing Store Streams**

## **FileStream**

### **Constructing a FileStream**

| FileStream fs1 = **File**.**OpenRead** ("readme.bin"); // **Read**-only FileStream fs2 = **File**.**OpenWrite** ("writeme.tmp"); // **Write**-only FileStream fs3 = **File**.Create ("readwrite.tmp"); // **Read**/write |
| --- |

**Create** method to open a file, if the file already exists, its existing content is truncated and the file is overwritten with zero bytes. This is different from the **OpenWrite** method, which leaves the existing content intact and positions the stream at the beginning of the file.

| **Method** | **Description** | **Return type** |
| --- | --- | --- |
| File.ReadAllText | Reads the entire contents of a file as a string | string |
| File.ReadAllLines | Reads the entire contents of a file as an array of strings, with one line per array element | string[] |
| File.ReadAllBytes | Reads the entire contents of a file as a byte array | byte[] |
| File.WriteAllText | Writes a string to a file, overwriting any existing contents | void |
| File.WriteAllLines | Writes an array of strings to a file, with one line per array element, overwriting any existing contents | void |
| File.WriteAllBytes | Writes a byte array to a file, overwriting any existing contents | void |
| File.AppendAllText | Appends a string to the end of a file, preserving any existing contents | void |

### **Specify a filename**

A filename can be either absolute (e.g., **c:\temp\test.txt**—or in Unix, /tmp/test.txt) or relative to the current directory (e.g., **test.txt or temp\test.txt**).

### **Specify a FileMode**

## **MemoryStream**

Backing store using array in MemoryStream. MemoryStream is still useful when need random access to a nonseekable stream.

## **PipeStream**

PipeStream : process able to communicate with another OS pipes protocol.

Two kinds of pipe :

* Anonymous Pipe : One way communication between parent and child process on the same computer
* Named Pipe (more flexible) : Two way communication between arbitrary processes on the same computer or different computers accross a network.

### **Named Pipes**

Parties communicate through a pipe of the same name. Roles : Server and Client.

**Server instantiates a NamedPipeServerStream and calls WaitForConnection**

| using **var** s = **new** **NamedPipeServerStream** ("pipedream"); **s**.**WaitForConnection**(); **s**.**WriteByte** (100); *// Send the value 100.* **Console**.**WriteLine** (s.**ReadByte**()); |
| --- |

**Client instantiates a NamedPipeClientStream and calls Connect**

| using **var** s = **new** **NamedPipeClientStream** ("pipedream"); **s**.**Connect**(); **Console**.**WriteLine** (s.**ReadByte**()); **s**.**WriteByte** (200); *// Send the value 200 back.* |
| --- |

### **Anonymous Pipes**

Anonymous Pipe is one way communication stream between a parent and child.

Server instantiates AnonymousPipeServerStream, and define Pipe Direction In or Out. And then calls GetClientHandleAsString to obtain an identifier of the pipe.

Child instantiates AnonymousPipeClientStream, define PipeDirection.

Server releases local handle generated on GetClientHandleAsString, by calling DisposeLocalCopyOfClientHandle.

Parent and Child communicate.

## **BufferedStream**

BufferedStream decorates or wraps stream with buffering capability and is one of a number of deorator stream types in .NET.

## **Stream Adapters**

Stream deals only in bytes to read or write data, and must plug in a adapter.

**Text adapters (for string and character data)**

TextReader, TextWriter

StreamReader, StreamWriter

StringReader, StringWriter

**Binary adapters (for primitive types such as int, bool, string, and float)**

BinaryReader, BinaryWriter

**XML adapters (covered in Chapter 11)**

XmlReader, XmlWriter