MemoryStream and Memory-Mapped Files

Mohamad Halabi
Microsoft Integration MVP
@mohamadhalabi

MemoryStream

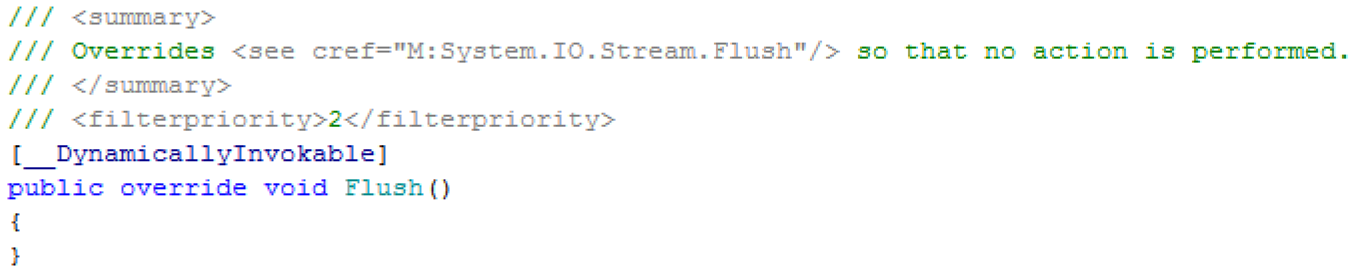
- **Simple and commonly used**
- **Backing store: in-memory buffer**
  - Very fast: no need for I/O read/write operations
  - Caution: not suitable for large data size
- **Defies an important streaming advantage:**
  - No reading data in chunks (all data is already in memory!)
- **So, should we ever use MemoryStream?**
  - Definitely a good choice for non-persistent small-size data

Usage Scenarios

- **Scenario 1: random access to data chunk from a non-seekable stream**
  - Read data chunk and store in an array
  - Wrap a MemoryStream around the array
  - MemoryStream is seekable
- **Scenario 2: random access to data from I/O call; ex: web service or database**
  - Wrap data in MemoryStream for fast random access

Flush

- **Flush has no implementation in MemoryStream**



```
/// <summary>
/// Overrides <see cref="M:System.IO.Stream.Flush"/> so that no action is performed.
/// </summary>
/// <filterpriority>2</filterpriority>
[__DynamicallyInvokable]
public override void Flush()
{
}
```

Why Are Memory-Mapped Files Discussed?

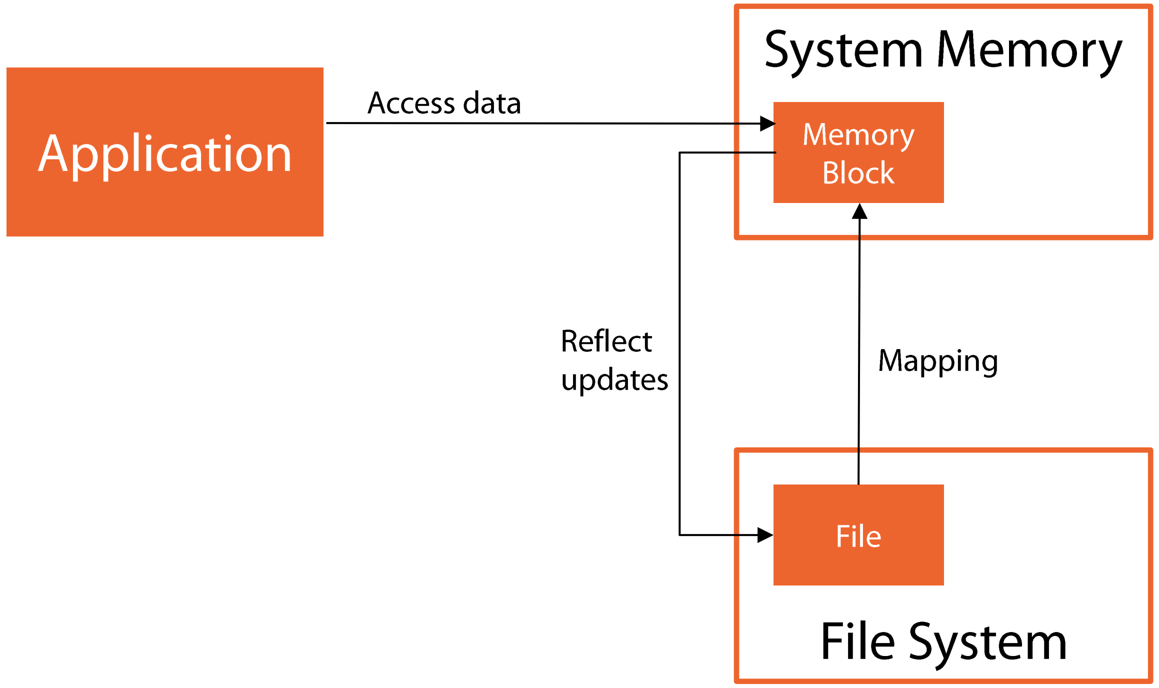
- **Memory-mapped files are <u>not</u> stream types**
- **Provide similar features to files and pipes**
  - So memory-mapped files are discussed to understand design alternatives

Memory-Mapped Files

- **Types found in System.IO.MemoryMappedFiles namespace**
- **Two key features:**

Feature 1	Feature 2
Fast random access to files	Shared memory between processes on same machine
FileStream allows random access to files	Pipes provide a shared memory between processes
Memory-mapped files vs. FileStream	Memory-mapped files vs. pipes and PipeStream

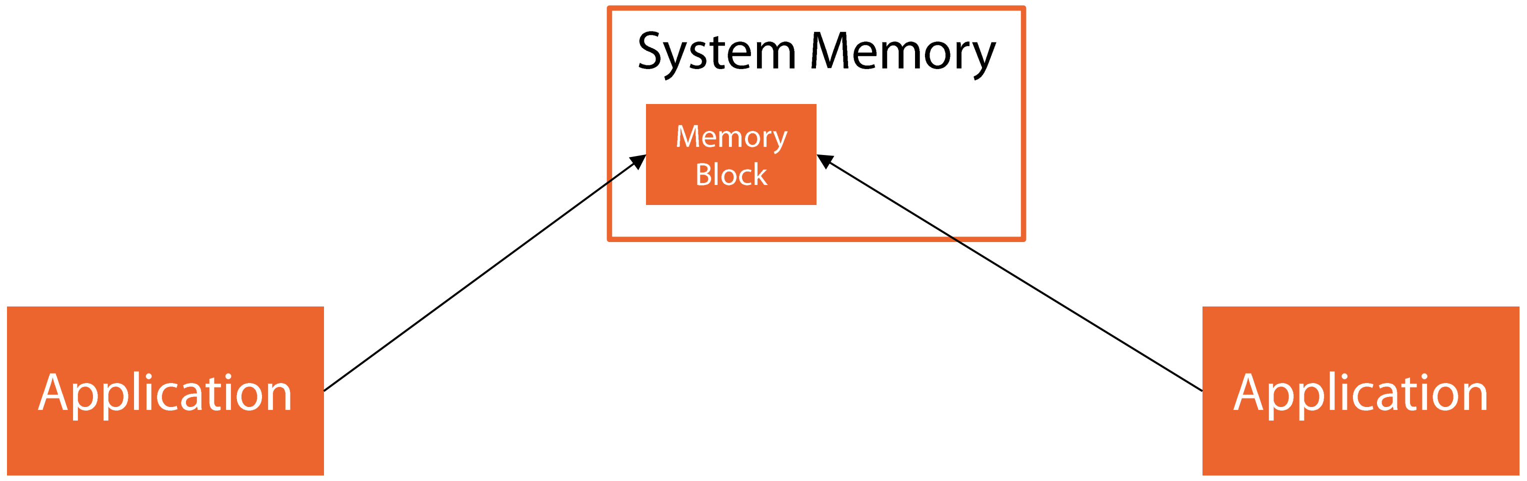
File Access


Application
Access data
System Memory
Memory Block
Reflect updates
Mapping
File
File System

Memory-Mapped Files vs. FileStream

FileStream	Memory-Mapped Files
Optimized for sequential file access	Better performance for random file access
I/O required to access data	Faster memory access
Not thread-safe because of pointer moves while writing and reading	Data can be read in a multi-threaded fashion

Memory Sharing



- Despite the name memory-mapped "file", there is no system file

Memory-Mapped Files vs. Pipes

Memory-Mapped Files	Pipes
Allow same-machine communication only	Allow cross-machine communication
Not stream-based. Shared memory block	Stream-based
Most efficient for single machine communication	

Using the API

- Use MemoryMappedFile's CreateNew instead of CreateFromFile

```
using (MemoryMappedFile mmf = MemoryMappedFile.CreateNew("MemoryLocation1", 1000))
using (MemoryMappedViewAccessor accessor = mmf.CreateViewAccessor())
{
    accessor.Write(0, 100);
    Console.ReadLine();
}

// the second process
using (MemoryMappedFile mmf = MemoryMappedFile.OpenExisting("MemoryLocation1"))
using (MemoryMappedViewAccessor accessor = mmf.CreateViewAccessor())
    Console.WriteLine(accessor.ReadInt32(0)); // returns 100
```

Summary

- **MemoryStream is often overused**
  - Suitable for small-size data
  - Not suitable for large-size data
- **Memory-mapped files are not stream types**
  - Fast random file access
  - Fast access to shared memory data between processes
  - Preferred over FileStream and Pipes (in terms of performance)