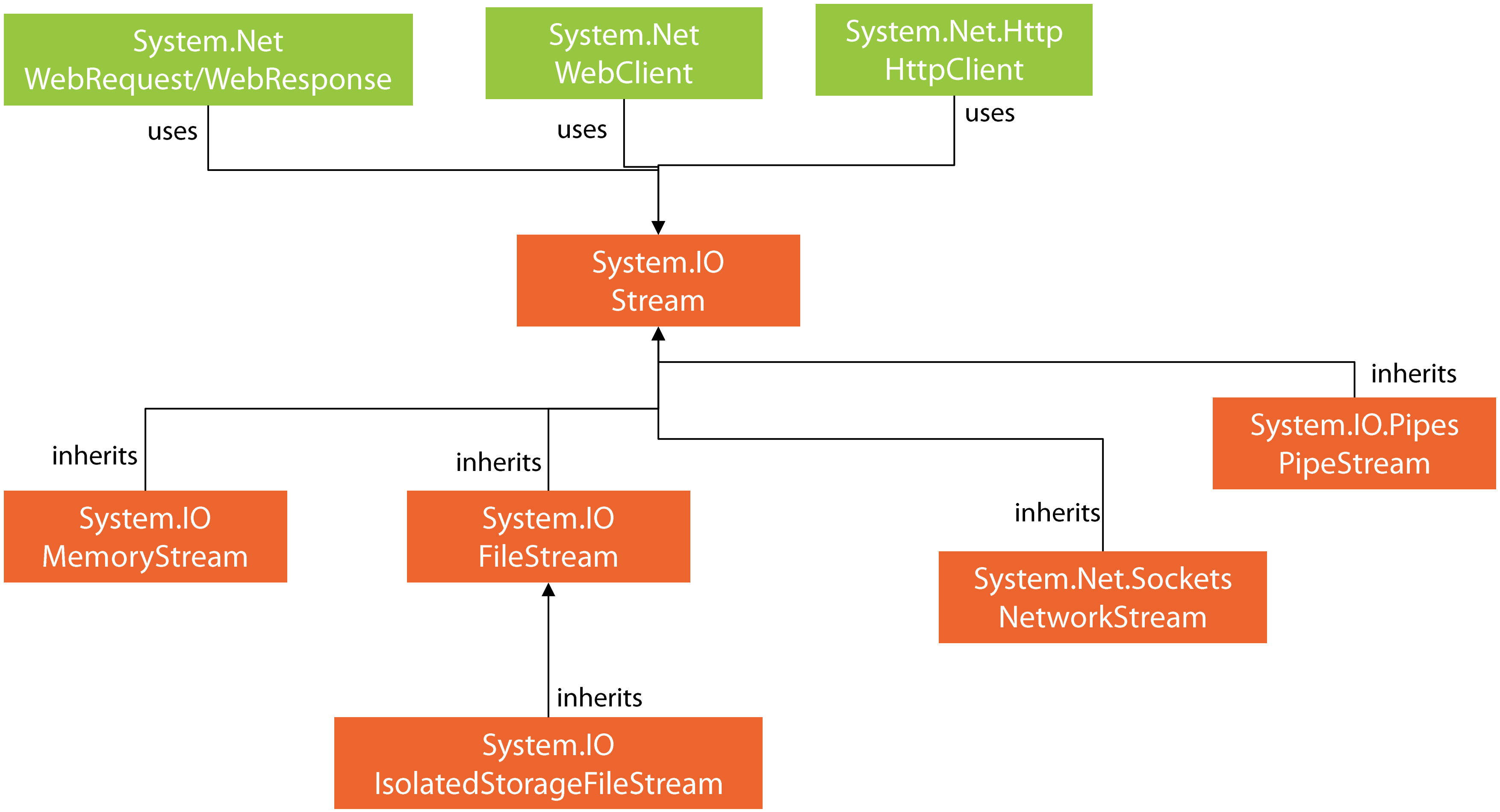
Working With FileStream

Mohamad Halabi
Microsoft Integration MVP
@mohamadhalabi

pluralsight
hardcore dev and IT training

Backing Store (Base) Streams



Instantiating a FileStream

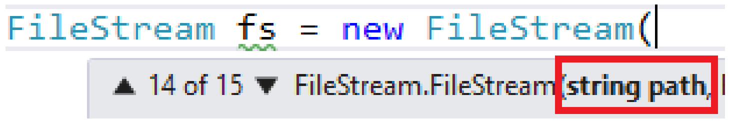
- **FileStream's backing store is file system**
  - So a file is required for instantiation
- **Two methods for instantiation:**
  - System.IO.File static methods
  - FileStream constructor overloads

Using File Class Methods

- **OpenRead: returns a read-only stream for an existing file**
  - Example: *FileStream fs = File.OpenRead(@"c:\myfiles\data.txt");*
- **If the file does not already exist:**
  - OpenWrite: creates the file and returns a <u>write-only stream</u>
    - Example: *FileStream fs = File.OpenWrite(@"c:\myfiles\data.txt");*
  - Create: creates the file and returns a <u>read/write stream</u>
    - Example: *FileStream fs = File.Create(@"c:\myfiles\data.txt");*
- **If the file already exists:**
  - Create: truncates existing content
  - OpenWrite: leaves existing content and sets Position to 0
    - You can explicitly advance the pointer to the end of stream

Using FileStream Constructors

- 15 constructor overloads
- Across these 15 overloads, two ways to point to the required file:
  - String file path for managed code

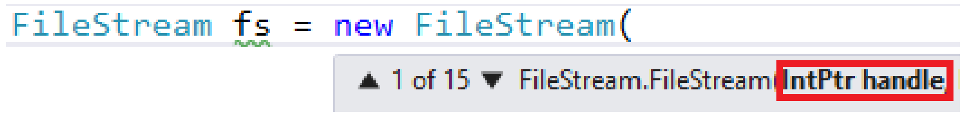


```
FileStream fs = new FileStream(
    ▲ 14 of 15 ▼ FileStream.FileStream(string path,
```

  - Operating system file handle for interoperability

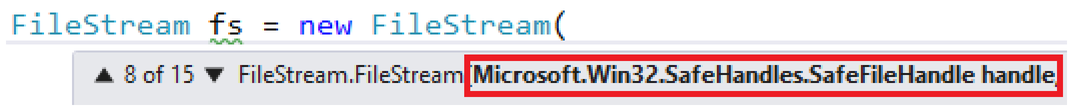
Obsolete in .NET 4.5

    - IntPtr:



```
FileStream fs = new FileStream(
    ▲ 1 of 15 ▼ FileStream.FileStream(IntPtr handle
```

    - SafeFileHandle:



```
FileStream fs = new FileStream(
    ▲ 8 of 15 ▼ FileStream.FileStream(Microsoft.Win32.SafeHandles.SafeFileHandle handle
```

More About Interoperability

- SafeHandle: http://msdn.microsoft.com/en-us/library/system.runtime.interopservices.safehandle.aspx
- FileStream.SafeFileHandle: http://msdn.microsoft.com/en-us/library/system.io.filestream.safefilehandle.aspx

Instantiating With a File Path String

- **Absolute value**
  - Ex: “c:\myfiles\data.txt”
- **Relative path to executing directory**
  - AppDomain.CurrentDomain.BaseDirectory returns application base directory
    - Ex: “C:\Users\mohamad\Desktop\Stream Course\Demos\FileStream\bin\Debug\”
  - FileStream fs = new FileStream(AppDomain.CurrentDomain.BaseDirectory + "data.txt“);
- **You can use the UNC path for network locations**
  - Ex: \\mohamadpc\Shares\data.txt
  - Ex: \\127.0.0.1\Shares\data.txt

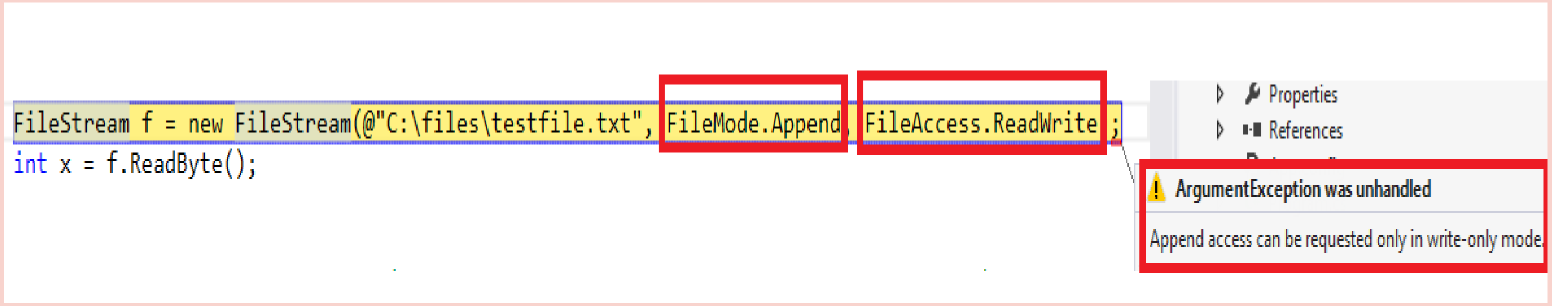
FileMode

- **Required enumerator for all managed code constructors (i.e. those that require string file path)**
  - FileStream fs = new FileStream(@"C:\files\data.txt", FileMode.Append,…
- **Determines how to open or create a file**

FileMode

Option	Explanation
CreateNew	Creates a new file If a file already exists, an exception is thrown
Create	Creates a new file If a file already exists, it’s overwritten
Open	Opens an existing file Sets Position to 0 If the file doesn’t exist, an exception is thrown
OpenOrCreate	Opens an existing file Creates a new file if it doesn’t already exist
Truncate	Opens a file Deletes file content (size = 0 bytes)

FileMode

Option	Explanation
Append	Opens a file and sets Position to end of file If file doesn't exist, it is created Only writing is allowed Only appending data is allowed Seek only works forwards 

```
FileStream f = new FileStream(@"C:\files\testfile.txt", FileMode.Append, FileAccess.ReadWrite);
int x = f.ReadByte();
```

ArgumentException was unhandled

Append access can be requested only in write-only mode.

FileAccess

- **By default FileStream will open a file in read/write access mode**
  - With the exception of FileMode.Append option
- **FileAccess enumeration sets file access to read, write, or read/write**

Option	Explanation
Read	File can only be read
Write	File can only be written to
Read/Write	Reading and writing operations are supported

- **FileAccess.Read and FileAccess.ReadWrite cannot be mixed with FileMode.Append**

FileShare

- **Files get locked by a FileStream until the stream is closed**
  - No other stream can access the file
- **FileShare enumeration can change this behavior**
  - Configures how the file can be shared with other streams

Option	Explanation
None	Sharing is not allowed. Default.
Delete	Subsequent streams can delete file
Inheritable	File handle can be inherited by child processes
Read	Streams can open file for reading only
Write	Streams can open file for writing only
ReadWrite	Streams can open file for reading and writing

FileShare

- **Be careful when you allow write sharing:**
  - This might affect file data and code quality
  - When two streams are allowed to share a file, unexpected results can happen

FileShare vs. Multithreading

- **Do not confuse FileShare with multithreading**
  - FileShare configures how multiple streams can access the same file
  - Multithreading is when multiple threads access the same stream

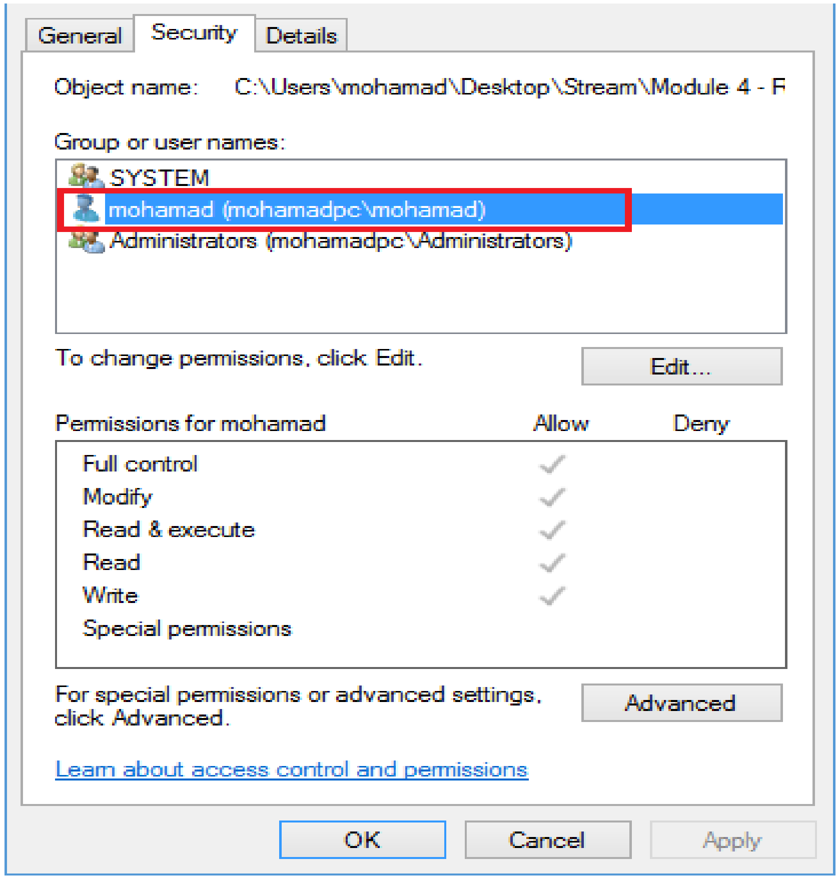
Access Control

- **Access control:**
  - What users have access
  - What permissions users have
- **This is different than FileAccess enumeration**
  - FileAccess controls FileStream open mode (read, write, read/write)
  - Access control sets access rules for system users
- **FileSecurity class in System.Security.AccessControl namespace**

Access Control

```
FileSecurity fs = new FileSecurity();
fs.AddAccessRule(new FileSystemAccessRule(@"mohamadpc\mohamad",
                                          FileSystemRights.FullControl,
                                          AccessControlType.Allow));

FileStream fstream = new FileStream(@"C:\files\data.txt", FileMode.Create,
    FileSystemRights.Write, FileShare.None, 8, FileOptions.Encrypted, fs);
```



Object name: C:\Users\mohamad\Desktop\Stream\Module 4 - F

Group or user names:

- SYSTEM
- mohamad (mohamadpc\mohamad)
- Administrators (mohamadpc\Administrators)

To change permissions, click Edit.

Permissions for mohamad	Allow	Deny
Full control	✓	
Modify	✓	
Read & execute	✓	
Read	✓	
Write	✓	
Special permissions		

For special permissions or advanced settings, click Advanced.

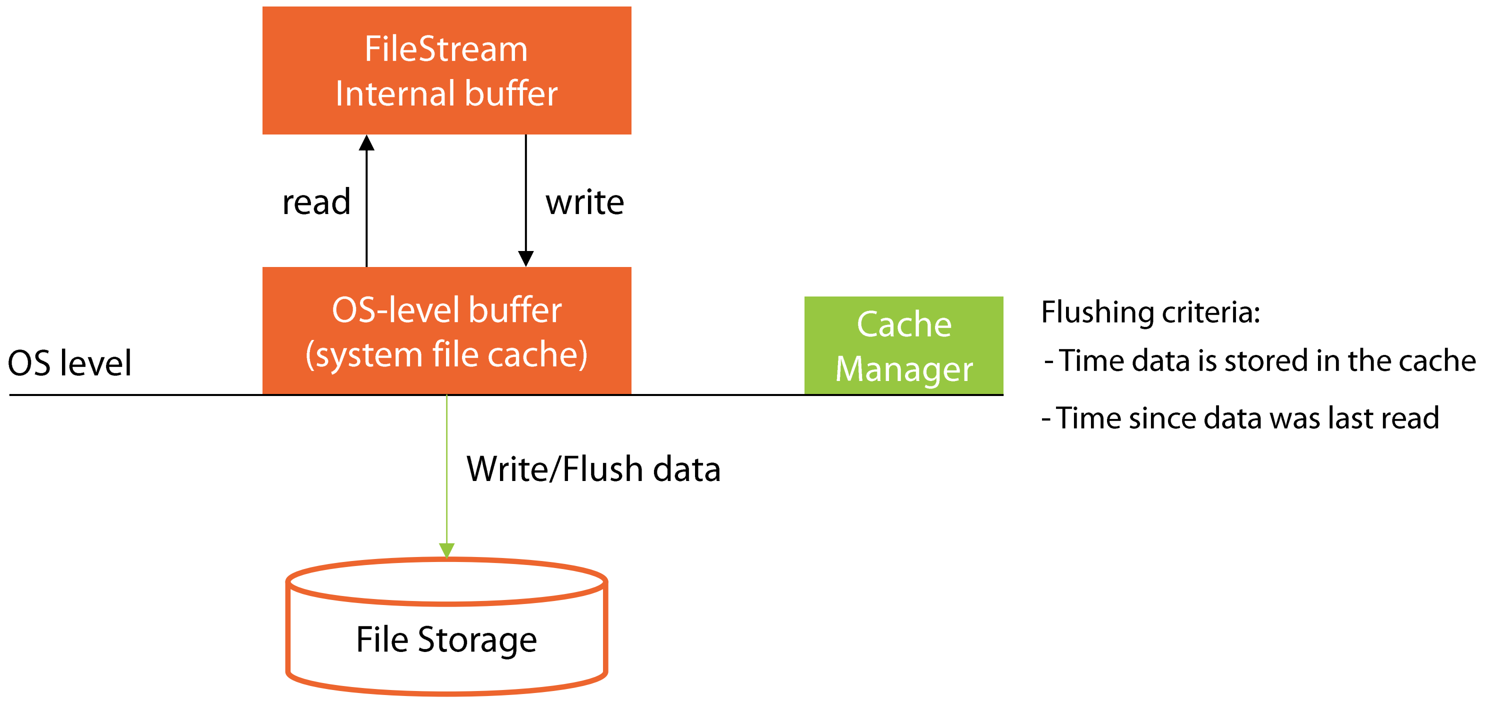
Learn about access control and permissions

Internal Buffer

- **Recall: streams can hold internal buffer to reduce I/O hits**
- **FileStream implements an internal buffer**
  - Default size of buffer is 4096 bytes
- **Buffer size can be set using constructor overloads**
  - Increasing buffer size will reduce I/O load but increase memory consumption
  - Reducing buffer size will increase I/O load but reduce memory consumption
- **4k size will be mostly sufficient**
- **Solid-state drives (SSD) handle I/O much faster than hard disk drives (HDD)**

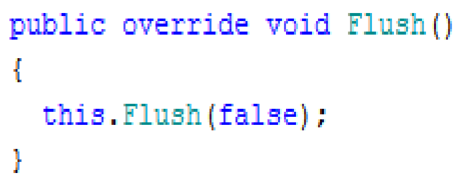
File Caching

- Recall that Flush writes internal buffer data into the backing store
- For FileStream, there is another layer of caching at OS level

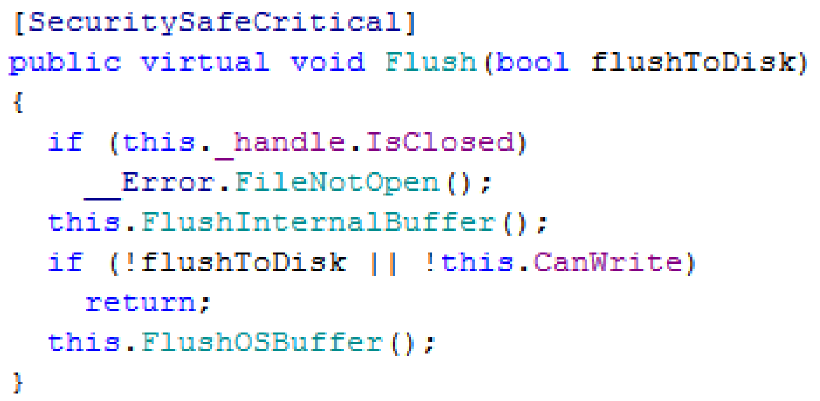

FileStream Internal buffer
read
write
OS-level buffer (system file cache)
OS level
Cache Manager
Flushing criteria:
- Time data is stored in the cache
- Time since data was last read
Write/Flush data
File Storage

Flushing a FileStream

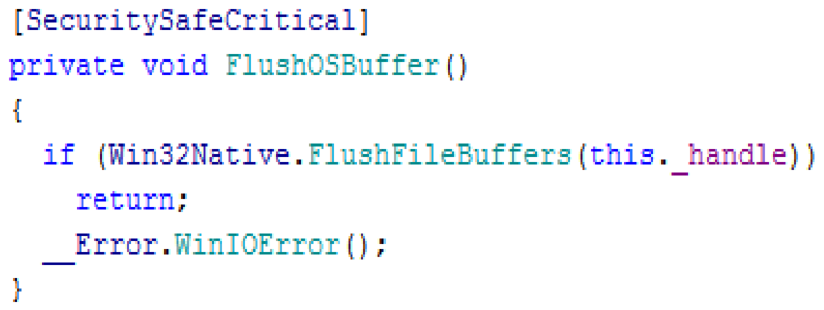
- Flush flushes FileStream's internal buffer but not the system cache
- FileStream.Flush(true) overload flushes the system cache



```
public override void Flush()
{
  this.Flush(false);
}
```



```
[SecuritySafeCritical]
public virtual void Flush(bool flushToDisk)
{
  if (this._handle.IsClosed)
    __Error.FileNotOpen();
  this.FlushInternalBuffer();
  if (!flushToDisk || !this.CanWrite)
    return;
  this.FlushOSBuffer();
}
```



```
[SecuritySafeCritical]
private void FlushOSBuffer()
{
  if (Win32Native.FlushFileBuffers(this._handle))
    return;
  __Error.WinIOError();
}
```

Turning Off System Caching

- FileOptions.WriteThrough turns off system caching
- Unless you have solid-state drives (SSD), disabling system cache can cause performance issues for frequent reads/writes
- For huge amount of data, system caching can also become a bottleneck
- Moral of the story: study your situation and do the tradeoffs

Optimize File Caching

- **FileOptions instructs the cache manager to optimize caching:**

SequentialScan	RandomAccess
File accessed sequentially moving the pointer forward	File accessed randomly
Cache manager optimizes caching for sequential access	Cache manager optimizes caching for random access
Using SequentialScan while performing random access prevents cache manager from optimizing caching	Using RandomAccess while performing sequential access prevents cache manager from optimizing caching
Example: developing a video player software	Example: developing a video editing software

If none is specified, cache manager will try to detect access pattern

Bitwise Combination

- FileOptions is decorated with FlagsAttribute, so it allows a bitwise combination
- Be careful not to combine incompatible flags

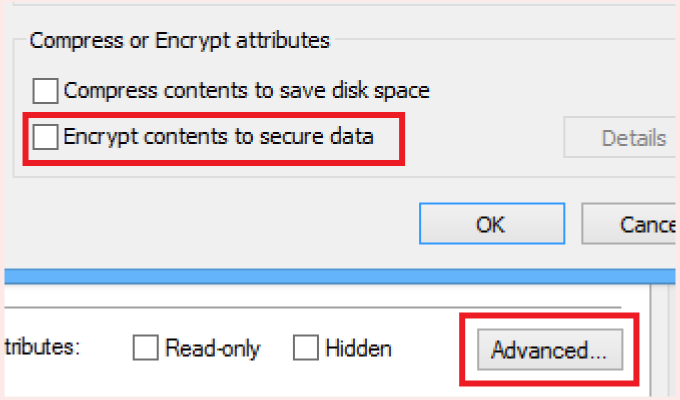
```
FileOptions.SequentialScan | FileOptions.RandomAccess
```

```
FileOptions.WriteThrough | FileOptions.RandomAccess
```

```
FileOptions.WriteThrough | FileOptions.SequentialScan
```

Other FileOptions

- **You have seen three values that relate to system caching:**
  - WriteThrough
  - SequentialScan
  - RandomAccess

Option	Explanation
DeleteOnClose	Deletes file once FileStream is closed. Useful for temporary files.
Encrypted	Encrypts file with Encrypting File System (EFS)  Compress or Encrypt attributes ☐ Compress contents to save disk space ☐ Encrypt contents to secure data tributes: ☐ Read-only ☐ Hidden
Asynchronous	Allows asynchronous access to the file

Summary

- **FileStream: stream implementation with file system backing store**
- **Two groups of constructors:**
  - Managed (string file path)
  - Interoperability (SafeFileHandle or (obsolete) IntPtr)
- **Constructor enumerations:**
  - FileMode: determines how to open or create a file
  - FileAccess: determines read, write, or read/write file access
  - FileShare: determines how subsequent streams can access current file
- **Access Control: determines users access and permissions (FileSecurity)**

Summary

- **File Caching: system-level cache**
    - FileOptions.SequentialScan and FileOptions.RandomAccess optimize cache manager caching
    - FileOptions.WriteThrough turns off system-level caching