

1 System.IO.Stream Class

```
2 [ILAsm]  
3 .class public abstract serializable Stream extends  
4 System.MarshalByRefObject implements System.IDisposable  
  
5 [C#]  
6 public abstract class Stream: MarshalByRefObject, IDisposable
```

7 Assembly Info:

- 8 • *Name:* mscorlib
- 9 • *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00]
- 10 • *Version:* 2.0.x.x
- 11 • *Attributes:*
 - 12 ○ CLSCompliantAttribute(true)

13 Implements:

- 14 • **System.IDisposable**

15 Summary

16 Abstract base class for all stream implementations.

17 Inherits From: System.MarshalByRefObject

18
19 **Library:** BCL

20
21 **Thread Safety:** All public static members of this type are safe for multithreaded operations.
22 No instance members are guaranteed to be thread safe.

23 24 Description

25 Streams involve three fundamental operations:

- 26 • You can read from streams. Reading is the transfer of data from a stream into a data
27 structure, such as an array of bytes.

- 28 • You can write to streams. Writing is the transfer of data from a data structure into a
29 stream.

- 30 • Streams can support seeking. Seeking is the querying and modifying of the current
31 position within a stream. Seek capability depends on the kind of backing store a
32 stream has. For example, network streams have no unified concept of a current
33 position, and therefore typically do not support seeking.

1 All classes that represent streams inherit from the `System.IO.Stream` class. The
2 `System.IO.Stream` class and its subclasses provide a generic view of data sources and
3 repositories, isolating the programmer from the specific details of the operating system and
4 underlying devices.

5
6 Subclasses are required to provide implementations only for the synchronous read and write
7 methods. The asynchronous read and write methods are implemented via the synchronous
8 ones. [*Note:* The `System.IO.Stream` synchronous read and write methods are
9 `System.IO.Stream.Read` and `System.IO.Stream.Write`. The asynchronous read and write
10 methods are `System.IO.Stream.BeginRead`, `System.IO.Stream.EndRead`,
11 `System.IO.Stream.BeginWrite`, and `System.IO.Stream.EndWrite`.]

12
13
14

15 Depending on the underlying data source or repository, streams might support only some of
16 these capabilities. An application can query a stream for its capabilities by using the
17 `System.IO.Stream.CanRead`, `System.IO.Stream.CanWrite`, and
18 `System.IO.Stream.CanSeek` properties.

19

20 The `System.IO.Stream.Read` and `System.IO.Stream.Write` methods read and write data in
21 a variety of formats. For streams that support seeking, the `System.IO.Stream.Seek` and
22 `System.IO.Stream.SetLength` methods, and the `System.IO.Stream.Position` and
23 `System.IO.Stream.Length` properties can be used to query and modify the current position
24 and length of a stream.

25

26 Some stream implementations perform local buffering of the underlying data to improve
27 performance. For such streams, the `System.IO.Stream.Flush` method can be used to clear
28 any internal buffers and ensure that all data has been written to the underlying data source
29 or repository.

30

31 Calling `System.IO.Stream.Close` on a `System.IO.Stream` flushes any buffered data,
32 essentially calling `System.IO.Stream.Flush` for you. `System.IO.Stream.Close` also
33 releases operating system resources such as file handles, network connections, or memory
34 used for any internal buffering.

35

36 If you need a `System.IO.Stream` with no backing store (i.e., a bit bucket), use
37 `System.IO.Stream.Null`.

38

1 Stream() Constructor

```
2 [ILAsm]  
3 family rtspecialname specialname instance void .ctor()  
4 [C#]  
5 protected Stream()
```

6 Summary

7 Constructs a new instance of the `System.IO.Stream` class.

8

1 Stream.Null Field

```
2 [ILAsm]  
3 .field public static initOnly class System.IO.Stream Null  
  
4 [C#]  
5 public static readonly Stream Null
```

6 Summary

7 Returns a `System.IO.Stream` with no backing store.

8 Description

9 *[Note: `System.IO.Stream.Null` is used to redirect output to a stream that does not*
10 *consume any operating system resources. When the methods of `System.IO.Stream` that*
11 *provide writing are invoked on `System.IO.Stream.Null`, they simply return, and no*
12 *data is written. `System.IO.Stream.Null` also implements a `System.IO.Stream.Read`*
13 *method that returns zero without reading data.]*

14
15
16

1 Stream.BeginRead(System.Byte[], 2 System.Int32, System.Int32, 3 System.AsyncCallback, System.Object) 4 Method

```
5 [ILAsm]  
6 .method public hidebysig virtual class System.IAsyncResult BeginRead(class  
7 System.Byte[] buffer, int32 offset, int32 count, class  
8 System.AsyncCallback callback, object state)  
9  
10 [C#]  
11 public virtual IAsyncResult BeginRead(byte[] buffer, int offset, int  
count, AsyncCallback callback, object state)
```

12 Summary

13 Begins an asynchronous read operation.

14 Parameters

Parameter	Description
<i>buffer</i>	The <code>System.Byte</code> array to read the data into.
<i>offset</i>	A <code>System.Int32</code> that specifies the byte offset in <i>buffer</i> at which to begin writing data read from the stream.
<i>count</i>	A <code>System.Int32</code> that specifies the maximum number of bytes to read from the stream.
<i>callback</i>	A <code>System.AsyncCallback</code> delegate to be called when the read is complete, or null.
<i>state</i>	An application-defined object, or null.

15 16 Return Value

17 A `System.IAsyncResult` that contains information about the asynchronous read
18 operation, which could still be pending.

19 Description

20 This method starts an asynchronous read operation. To determine how many bytes were
21 read and release resources allocated by this method, call the
22 `System.IO.Stream.EndRead` method and specify the `System.IAsyncResult` object

1 returned by this method. [*Note:* The `System.IO.Stream.EndRead` method should be
2 called exactly once for each call to `System.IO.Stream.BeginRead`.]
3
4
5

6 If the *callback* parameter is not `null`, the method referenced by *callback* is invoked
7 when the asynchronous operation completes. The `System.IAsyncResult` object
8 returned by this method is passed as the argument to the method referenced by
9 *callback*.
10

11 The current position in the stream is updated when the asynchronous read or write is
12 issued, not when the I/O operation completes.
13

14 Multiple simultaneous asynchronous requests render the request completion order
15 unspecified.
16

17 The *state* parameter can be any object that the caller wishes to have available for the
18 duration of the asynchronous operation. This object is available via the
19 `System.IAsyncResult.AsyncState` property of the object returned by this method.
20

21 [*Note:* Use the `System.IO.Stream.CanRead` property to determine whether the current
22 instance supports reading.]
23
24

25 Behaviors

26 As described above.
27

28 Exceptions

Exception	Condition
System.NotSupportedException	The current <code>System.IO.Stream</code> does not support reading.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error occurred.

29

30

1 Stream.BeginWrite(System.Byte[], 2 System.Int32, System.Int32, 3 System.AsyncCallback, System.Object) 4 Method

```
5 [ILAsm]  
6 .method public hidebysig virtual class System.IAsyncResult  
7 BeginWrite(class System.Byte[] buffer, int32 offset, int32 count, class  
8 System.AsyncCallback callback, object state)  
  
9 [C#]  
10 public virtual IAsyncResult BeginWrite(byte[] buffer, int offset, int  
11 count, AsyncCallback callback, object state)
```

12 Summary

13 Begins an asynchronous write operation.

14 Parameters

Parameter	Description
<i>buffer</i>	The <code>System.Byte</code> array to be written to the current stream.
<i>offset</i>	A <code>System.Int32</code> that specifies the byte offset in <i>buffer</i> at which to begin copying bytes to the current stream.
<i>count</i>	A <code>System.Int32</code> that specifies the maximum number of bytes to be written to the current stream.
<i>callback</i>	A <code>System.AsyncCallback</code> delegate to be called when the write is complete, or null.
<i>state</i>	An application-defined object, or null.

15 16 Return Value

17 A `System.IAsyncResult` that represents the asynchronous write, which could still be
18 pending.

19 Description

20 Pass the `System.IAsyncResult` returned by this method to
21 `System.IO.Stream.EndWrite` to ensure that the write completes and frees resources
22 appropriately. If an error occurs during an asynchronous write, an exception will not be

1 thrown until `System.IO.Stream.EndWrite` is called with the `System.IAsyncResult`
2 returned by this method. [*Note:* If a failure is detected from the underlying OS (such as
3 if a floppy is ejected in the middle of the operation), the results of the write operation
4 are undefined.]

5
6
7
8 If the *callback* parameter is not `null`, the method referenced by *callback* is invoked
9 when the asynchronous operation completes. The `System.IAsyncResult` object
10 returned by this method is passed as the argument to the method referenced by
11 *callback*.

12
13 The *state* parameter can be any object that the caller wishes to have available for the
14 duration of the asynchronous operation. This object is available via the
15 `System.IAsyncResult.AsyncState` property of the object returned by this method.

16
17 If a stream is writable, writing at the end of it expands the stream.

18
19 The current position in the stream is updated when you issue the asynchronous read or
20 write, not when the I/O operation completes. Multiple simultaneous asynchronous
21 requests render the request completion order uncertain.

22
23 [*Note:* *buffer* should generally be greater than 64 KB.

24
25 Use the `System.IO.Stream.CanWrite` property to determine whether the current
26 instance supports writing.

27
28]

29 Behaviors

30 As described above.

32 Exceptions

Exception	Condition
<code>System.NotSupportedException</code>	The current <code>System.IO.Stream</code> does not support writing.
<code>System.ObjectDisposedException</code>	The stream is closed.
<code>System.IO.IOException</code>	An I/O error occurred.

33

34

1 Stream.Close() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual void Close()  
4 [C#]  
5 public virtual void Close()
```

6 Summary

7 Closes the current stream and releases any resources associated with the current
8 stream.

9 Description

10 Following a call to this method, a call to another operation on the same stream might
11 result in an exception (such as `System.ObjectDisposedException`, for example).
12 However, if the stream is already closed, a call to `System.IO.Stream.Close` throws no
13 exceptions.

14
15 [*Note:* If this method is called while an asynchronous read or write is pending for a
16 stream, the behavior of the stream is undefined.]
17
18

19 Behaviors

20 As described above.
21
22

1 Stream.CreateWaitHandle() Method

```
2 [ILAsm]  
3 .method family hidebysig virtual class System.Threading.WaitHandle  
4 CreateWaitHandle()  
  
5 [C#]  
6 protected virtual WaitHandle CreateWaitHandle()
```

7 Summary

8 Allocates a System.Threading.WaitHandle object.

9 Return Value

10 A reference to the allocated System.Threading.WaitHandle.

11 Description

12 When called for the first time this method creates a System.Threading.WaitHandle
13 object and returns it. On subsequent calls, the System.IO.Stream.CreateWaitHandle
14 method returns a reference to the same wait handle.

15
16 [Note: System.IO.Stream.CreateWaitHandle is useful if you implement the
17 asynchronous methods and require a way of blocking in System.IO.Stream.EndRead or
18 System.IO.Stream.EndWrite until the asynchronous operation is complete.]
19
20

21

1 Stream.EndRead(System.IAsyncResult)

2 Method

```
3 [IAsm]  
4 .method public hidebysig virtual int32 EndRead(class System.IAsyncResult  
5 asyncResult)  
  
6 [C#]  
7 public virtual int EndRead(IAsyncResult asyncResult)
```

8 Summary

9 Ends a pending asynchronous read request.

10 Parameters

Parameter	Description
<i>asyncResult</i>	The <code>System.IAsyncResult</code> object that references the pending asynchronous read request.

11 Return Value

13 A `System.Int32` that indicates the number of bytes read from the stream, between 0
14 and the number of bytes specified via the `System.IO.Stream.BeginRead` parameter
15 *count*. Streams only return 0 at the end of the stream, otherwise, they block until at
16 least 1 byte is available.

17 Description

18 `System.IO.Stream.EndRead` blocks until the I/O operation has completed.

19 Behaviors

20 As described above.

21 Exceptions

Exception	Condition
System.ArgumentNullException	<i>asyncResult</i> is null.
System.ArgumentException	<i>asyncResult</i> did not originate from a <code>System.IO.Stream.BeginRead</code> method on the current

	stream.
--	---------

1

2

1 Stream.EndWrite(System.IAsyncResult)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig virtual void EndWrite(class System.IAsyncResult  
5 asyncResult)  
  
6 [C#]  
7 public virtual void EndWrite(IAsyncResult asyncResult)
```

8 Summary

9 Ends an asynchronous write operation.

10 Parameters

Parameter	Description
<i>asyncResult</i>	A <code>System.IAsyncResult</code> that references the outstanding asynchronous I/O request.

11 Description

13 `System.IO.Stream.EndWrite` is required to be called exactly once for every
14 `System.IO.Stream.BeginWrite`. `System.IO.Stream.EndWrite` blocks until the write I/O
15 operation has completed.

16 Behaviors

17 As described above.

19 Exceptions

Exception	Condition
<code>System.ArgumentNullException</code>	The <i>asyncResult</i> parameter is null.
<code>System.ArgumentException</code>	<i>asyncResult</i> did not originate from a <code>System.IO.Stream.BeginWrite</code> method on the current stream.

20
21

1 Stream.Flush() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual abstract void Flush()  
4 [C#]  
5 public abstract void Flush()
```

6 Summary

7 Flashes the internal buffer.

8 Description

9 *[Note: Implementers should use this method to move any information from an*
10 *underlying buffer to its destination. The `System.IO.Stream.Flush` method should clear*
11 *the buffer, but the stream should not be closed. Depending upon the state of the object,*
12 *the current position within the stream might need to be modified (for example, if the*
13 *underlying stream supports seeking). For additional information see*
14 *`System.IO.Stream.CanSeek`.]*
15
16

17 Behaviors

18 As described above.

19

20 How and When to Override

21 Override `System.IO.Stream.Flush` on streams that implement a buffer.

22

23 Exceptions

Exception	Condition
System.IO.IOException	An I/O error occurs.
System.ObjectDisposedException	The stream is closed.

24

25

1 Stream.Read(System.Byte[], System.Int32, 2 System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig virtual abstract int32 Read(class System.Byte[]  
5 buffer, int32 offset, int32 count)  
  
6 [C#]  
7 public abstract int Read(byte[] buffer, int offset, int count)
```

8 Summary

9 Reads a sequence of bytes from the current stream and advances the position within the
10 stream by the number of bytes read.

11 Parameters

Parameter	Description
<i>buffer</i>	A <code>System.Byte</code> array. When this method returns, the elements between <i>offset</i> and $(offset + count - 1)$ are replaced by the bytes read from the current source.
<i>offset</i>	A <code>System.Int32</code> that specifies the zero based byte offset in <i>buffer</i> at which to begin storing the data read from the current stream.
<i>count</i>	A <code>System.Int32</code> that specifies the maximum number of bytes to be read from the current stream.

12

13 Return Value

14 A `System.Int32` that specifies the total number of bytes read into the buffer, or zero if
15 the end of the stream has been reached before any data can be read.

16 Description

17 [Note: Use the `System.IO.Stream.CanRead` property to determine whether the current
18 instance supports reading.]
19
20

21 Behaviors

22 As described above.
23

1 Exceptions

Exception	Condition
System.ArgumentException	<i>(offset + count - 1)</i> is greater than the length of <i>buffer</i> .
System.ArgumentNullException	<i>buffer</i> is null.
System.ArgumentOutOfRangeException	<i>offset</i> or <i>count</i> is less than zero.
System.NotSupportedException	The current stream does not support reading.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error occurred.

2

3

1 Stream.ReadByte() Method

```
2 [ILAsm]  
3 .method public hidebysig virtual int32 ReadByte()  
4 [C#]  
5 public virtual int ReadByte()
```

6 Summary

7 Reads a byte from the stream and advances the position within the stream by one byte.

8 Return Value

9 The unsigned byte cast to a `System.Int32`, or -1 if at the end of the stream.

10 Description

11 Behaviors

12 As described above.

13

14

15 [*Note:* Use the `System.IO.Stream.CanRead` property to determine whether the current
16 instance supports reading.]

17

18

19 Exceptions

Exception	Condition
<code>System.NotSupportedException</code>	The stream does not support reading.
<code>System.ObjectDisposedException</code>	The stream is closed.
<code>System.IO.IOException</code>	An I/O error has occurred.

20

21

1 Stream.Seek(System.Int64, 2 System.IO.SeekOrigin) Method

```
3 [ILAsm]  
4 .method public hidebysig virtual abstract int64 Seek(int64 offset,  
5 valuetype System.IO.SeekOrigin origin)  
  
6 [C#]  
7 public abstract long Seek(long offset, SeekOrigin origin)
```

8 Summary

9 Changes the position within the current stream by the given offset, which is relative to
10 the stated origin.

11 Parameters

Parameter	Description
<i>offset</i>	A System.Int64 that specifies the byte offset relative to origin.
<i>origin</i>	A System.IO.SeekOrigin value indicating the reference point used to obtain the new position.

12

13 Return Value

14 A System.Int64 that specifies the new position within the current stream.

15 Description

16 [Note: Use the System.IO.Stream.CanSeek property to determine whether the current
17 instance supports seeking.]
18
19

20 Behaviors

21 If *offset* is negative, the new position is required to precede the position specified by
22 *origin* by the number of bytes specified by *offset*. If *offset* is zero, the new position is
23 required to be the position specified by *origin*. If *offset* is positive, the new position is
24 required to follow the position specified by *origin* by the number of bytes specified by
25 *offset*.

26

27 How and When to Override

1 Classes derived from `System.IO.Stream` that support seeking are required to override
2 this method.

3 **Exceptions**

Exception	Condition
System.NotSupportedException	The stream does not support seeking, such as if the stream is constructed from a pipe or console output.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error has occurred.

4

5

1 Stream.SetLength(System.Int64) Method

```
2 [ILAsm]  
3 .method public hidebysig virtual abstract void SetLength(int64 value)  
4 [C#]  
5 public abstract void SetLength(long value)
```

6 Summary

7 Sets the length of the current stream.

8 Parameters

Parameter	Description
<i>value</i>	A System.Int64 that specifies the desired length of the current stream in bytes.

9

10 Description

11 [Note: Use the System.IO.Stream.CanWrite property to determine whether the current
12 instance supports writing, and the System.IO.Stream.CanSeek property to determine
13 whether seeking is supported.]
14
15

16 Behaviors

17 If the specified value is less than the current length of the stream, the stream is
18 truncated. If the specified value is larger than the current length of the stream, the
19 stream is expanded. If the stream is expanded, the contents of the stream between the
20 old and the new length are initialized to zeros.

21

22 Default

23 There is no default implementation.
24

25 How and When to Override

26 Classes derived from System.IO.Stream are required to support both writing and
27 seeking for System.IO.Stream.SetLength to work.

1

2 Exceptions

Exception	Condition
System.NotSupportedException	The stream does not support both writing and seeking, such as if the stream is constructed from a pipe or console output.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error occurred.

3

4

1 Stream.System.IDisposable.Dispose()

2 Method

```
3 [ILAsm]  
4 .method private final hidebysig virtual void System.IDisposable.Dispose()  
5 [C#]  
6 void IDisposable.Dispose()
```

7 Summary

8 Implemented to support the `System.IDisposable` interface. [Note: For more
9 information, see `System.IDisposable.Dispose`.]

10

Stream.Write(System.Byte[], System.Int32, System.Int32) Method

```
[ILAsm]
.method public hidebysig virtual abstract void Write(class System.Byte[]
buffer, int32 offset, int32 count)

[C#]
public abstract void Write(byte[] buffer, int offset, int count)
```

Summary

Writes a sequence of bytes to the current stream and advances the current position within the current stream by the number of bytes written.

Parameters

Parameter	Description
<i>buffer</i>	A System.Byte array containing the data to write.
<i>offset</i>	A System.Int32 that specifies the zero based byte offset in <i>buffer</i> at which to begin copying bytes to the current stream.
<i>count</i>	A System.Int32 that specifies the number of bytes to be written to the current stream.

Description

[Note: Use the System.IO.Stream.CanWrite property to determine whether the current instance supports writing.]

Behaviors

If the write operation is successful, the position within the stream advances by the number of bytes written. If an exception occurs, the position within the stream remains unchanged.

Exceptions

Exception	Condition
-----------	-----------

System.ArgumentException	<i>(offset + count)</i> is greater than the length of <i>buffer</i> .
System.ArgumentNullException	<i>buffer</i> is null.
System.ArgumentOutOfRangeException	<i>offset</i> or <i>count</i> is negative.
System.NotSupportedException	The stream does not support writing.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error occurred.

1

2

Stream.WriteByte(System.Byte) Method

```
[ILAsm]  
.method public hidebysig virtual void WriteByte(unsigned int8 value)  
  
[C#]  
public virtual void WriteByte(byte value)
```

Summary

Writes a `System.Byte` to the current position in the stream and advances the position within the stream by one byte.

Parameters

Parameter	Description
<i>value</i>	The <code>System.Byte</code> to write to the stream.

Description

[*Note:* Use the `System.IO.Stream.CanWrite` property to determine whether the current instance supports writing.]

Behaviors

As described above.

Exceptions

Exception	Condition
System.NotSupportedException	The stream does not support writing.
System.ObjectDisposedException	The stream is closed.
System.IO.IOException	An I/O error has occurred.

1 Stream.CanRead Property

```
2 [ILAsm]  
3 .property bool CanRead { public hidebysig virtual abstract specialname  
4 bool get_CanRead() }  
5 [C#]  
6 public abstract bool CanRead { get; }
```

7 Summary

8 Gets a `System.Boolean` value indicating whether the current stream supports reading.

9 Property Value

10 `true` if the stream supports reading; otherwise, `false`.

11 Description

12 If a class derived from `System.IO.Stream` does not support reading, the following
13 methods throw a `System.NotSupportedException`: `System.IO.Stream.BeginRead`,
14 `System.IO.Stream.Read` and `System.IO.Stream.ReadByte`.

15 Behaviors

16 As described above.

17

18

1 Stream.CanSeek Property

```
2 [ILAsm]  
3 .property bool CanSeek { public hidebysig virtual abstract specialname  
4 bool get_CanSeek() }  
  
5 [C#]  
6 public abstract bool CanSeek { get; }
```

7 Summary

8 Gets a `System.Boolean` value indicating whether the current stream supports seeking.

9 Property Value

10 `true` if the stream supports seeking; otherwise, `false`.

11 Description

12 If a class derived from `System.IO.Stream` does not support seeking, the following
13 methods throw a `System.NotSupportedException`: `System.IO.Stream.Length`,
14 `System.IO.Stream.SetLength`, `System.IO.Stream.Position`, or
15 `System.IO.Stream.Seek`.

16 Behaviors

17 As described above.

18

19

1 Stream.CanWrite Property

```
2 [ILAsm]  
3 .property bool CanWrite { public hidebysig virtual abstract specialname  
4 bool get_CanWrite() }  
  
5 [C#]  
6 public abstract bool CanWrite { get; }
```

7 Summary

8 Gets a `System.Boolean` value indicating whether the current stream supports writing.

9 Property Value

10 true if the stream supports writing; otherwise, false.

11 Description

12 If a class derived from `System.IO.Stream` does not support writing, the following
13 methods throw a `System.NotSupportedException`: `System.IO.Stream.Write`,
14 `System.IO.Stream.WriteByte`, and `System.IO.Stream.BeginWrite`.

15 Behaviors

16 As described above.

17

18

1 Stream.Length Property

```
2 [ILAsm]  
3 .property int64 Length { public hidebysig virtual abstract specialname  
4 int64 get_Length() }  
  
5 [C#]  
6 public abstract long Length { get; }
```

7 Summary

8 Gets the length in bytes of the stream.

9 Property Value

10 A `System.Int64` value representing the length of the stream in bytes.

11 Description

12 *[Note: Use the `System.IO.Stream.CanSeek` property to determine whether the current*
13 *instance supports seeking.]*
14
15

16 Behaviors

17 This property is read-only.
18

19 Exceptions

Exception	Condition
System.NotSupportedException	The stream does not support seeking.
System.ObjectDisposedException	The stream is closed.

20

21

1 Stream.Position Property

```
2 [ILAsm]
3 .property int64 Position { public hidebysig virtual abstract specialname
4 int64 get_Position() public hidebysig virtual abstract specialname void
5 set_Position(int64 value) }
6 [C#]
7 public abstract long Position { get; set; }
```

8 Summary

9 Gets or sets the position within the current stream.

10 Property Value

11 A `System.Int64` that specifies the current position within the stream.

12 Description

13 The stream is required to support seeking to get or set the position. [*Note:* Use the
14 `System.IO.Stream.CanSeek` property to determine whether the current instance
15 supports seeking.]

16
17
18
19 Classes that derive from `System.IO.Stream` are required to provide an implementation
20 of this property.

21 Behaviors

22 As described above.

23

24 Exceptions

Exception	Condition
<code>System.NotSupportedException</code>	The stream does not support seeking.
<code>System.ObjectDisposedException</code>	The stream is closed.
<code>System.IO.IOException</code>	An I/O error has occurred.

25

26