

1 System.Threading.Thread Class

```
2 [ILAsm]  
3 .class public sealed Thread extends System.Object  
  
4 [C#]  
5 public sealed class Thread
```

6 Assembly Info:

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

12 Summary

13 Represents a sequential thread of execution.

14 Inherits From: System.Object

15

16 **Library:** BCL

17

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

20

21 Description

22 A process can create and execute one or more threads to execute a portion of the
23 program code associated with the process. A `System.Threading.ThreadStart` delegate
24 is used to specify the program code executed by a thread.

25

26 Some operating systems might not utilize the concepts of threads or preemptive
27 scheduling. Also, the concept of "thread priority" might not exist at all or its meaning
28 might vary, depending on the underlying operating system. Implementers of the
29 `System.Threading.Thread` type are required to describe their threading policies,
30 including what thread priority means, how many threading priority levels exist, and
31 whether scheduling is preemptive.

32

33 For the duration of its existence, a thread is always in one or more of the states defined
34 by `System.Threading.ThreadState`. A scheduling priority level, as defined by
35 `System.Threading.ThreadPriority`, can be requested for a thread, but it might not be
36 honored by the operating system.

37

38 If an unhandled exception is thrown in the code executed by a thread created by an
39 application, a `System.AppDomain.UnhandledException` event is raised
40 (`System.UnhandledExceptionEventArgs.IsTerminating` is set to false), and the
41 thread is terminated; the current process is not terminated.

42

1 Thread(System.Threading.ThreadStart)

2 Constructor

```
3 [ILAsm]  
4 public rtspecialname specialname instance void .ctor(class  
5 System.Threading.ThreadStart start)  
  
6 [C#]  
7 public Thread(ThreadStart start)
```

8 Summary

9 Constructs and initializes a new instance of the System.Threading.Thread class.

10 Parameters

Parameter	Description
<i>start</i>	A System.Threading.ThreadStart delegate that references the methods to be invoked when the new thread begins executing.

11

12 [*Note:* To schedule the thread for execution, call System.Threading.Thread.Start.]

13

14

15

16 Until System.Threading.Thread.Start is called, the thread's state includes

17 System.Threading.ThreadState.Unstarted.

18 Exceptions

Exception	Condition
System.ArgumentNullException	<i>start</i> is null.

19

20

1 Thread.Abort(System.Object) Method

```
2 [ILAsm]  
3 .method public hidebysig instance void Abort(object stateInfo)  
4 [C#]  
5 public void Abort(object stateInfo)
```

6 Summary

7 Raises a `System.Threading.ThreadAbortException` in the thread on which it is invoked
8 to begin the process of terminating the thread. In all but the most extraordinary
9 situations, calling this method will terminate the thread.

10 Parameters

Parameter	Description
<i>stateInfo</i>	A <code>System.Object</code> that contains application-specific information, such as state, which can be used by the thread being aborted.

11 Description

13 The object passed as the *stateInfo* parameter can be obtained by accessing the
14 `System.Threading.ThreadAbortException.ExceptionState` property.
15
16 [*Note:* For details on aborting threads, see `System.Threading.Thread.Abort ()`.]
17
18

19 Exceptions

Exception	Condition
System.Security.SecurityException	Caller does not have <code>System.Security.Permissions.SecurityPermissionFlag.ControlThread</code> security permission for this thread.

20 Permissions

Permission	Description
System.Security.SecurityPermission	Requires permission to control the thread to be aborted. See <code>System.Security.Permissions.SecurityPermissionFlag</code> .

	ControlThread.
--	----------------

1

2

1 Thread.Abort() Method

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

6 Summary

7 Raises a `System.Threading.ThreadAbortException` in the thread on which it is invoked
8 to begin the process of terminating the thread. In all but the most extraordinary
9 situations, calling this method will terminate the thread.

10 Description

11 When this method is invoked on a thread, the system throws a
12 `System.Threading.ThreadAbortException` in the thread to abort it. Invoking
13 `System.Threading.Thread.Abort` on a thread is similar to arranging for the target
14 thread to throw a `System.Threading.ThreadAbortException`. Because, unlike other
15 exceptions, a `System.Threading.ThreadAbortException` is sent to another thread, the
16 exception might be delayed. A `System.Threading.ThreadAbortException` is required to
17 be delayed if and while the target thread is executing any of the following:

- 18 • unmanaged code
- 19 • a catch handler
- 20 • a finally clause
- 21 • a filter clause
- 22 • a type initializer

23 A thread abort proceeds as follows:

- 24 1. An abort begins at the earliest of the following times:
 - 25 a. when the thread transitions from unmanaged to managed code execution;
 - 26 b. when the thread finishes the outermost currently executing catch handler;
 - 27 c. immediately if the thread is running managed code outside of any catch handler,
 - 28 finally clause, filter clause or type initializer
 - 29
 - 30
 - 31
- 32 2. Whenever an outermost catch handler finishes execution, the
33 `System.Threading.ThreadAbortException` is rethrown unless the thread being
34 aborted has called `System.Threading.Thread.ResetAbort` since the call to
35 `System.Threading.Thread.Abort`.

1 3. When all finally blocks have been called and the thread is about to transition to any
2 unmanaged code which executed before the first entry to managed code,
3 `System.Threading.Thread.ResetAbort` is called so that a return to managed code
4 will consider the abort to have been successfully processed.

5 Unexecuted `finally` blocks are executed before the thread is aborted; this includes any
6 finally block that is executing when the exception is thrown. The thread is not guaranteed to
7 abort immediately, or at all. This situation can occur if a thread does an unbounded amount
8 of computation in the finally blocks that are called as part of the abort procedure, thereby
9 indefinitely delaying the abort. To ensure a thread has aborted, invoke
10 `System.Threading.Thread.Join` on the thread after calling
11 `System.Threading.Thread.Abort`.

12
13 If `System.Threading.Thread.Abort` is called on a thread that has not been started, the
14 thread aborts when `System.Threading.Thread.Start` is called. If the target thread is
15 blocked or sleeping in managed code and is not inside any of the code blocks that are
16 required to delay an abort, the thread is resumed and immediately aborted.

17
18 After `System.Threading.Thread.Abort` is invoked on a thread, the state of the thread
19 includes `System.Threading.ThreadState.AbortRequested`. After the thread has
20 terminated as a result of a successful call to `System.Threading.Thread.Abort`, the state of
21 the thread includes `System.Threading.ThreadState.Stopped` and
22 `System.Threading.ThreadState.Aborted`.

23
24 [Note: With sufficient permissions, a thread that is the target of a
25 `System.Threading.Thread.Abort` can cancel the abort using the
26 `System.Threading.Thread.ResetAbort` method. For an example that demonstrates calling
27 the `System.Threading.Thread.ResetAbort` method, see
28 `System.Threading.ThreadAbortException`.]
29
30

31 Exceptions

Exception	Condition
System.Security.SecurityException	Caller does not have <code>System.Security.Permissions.SecurityPermissionFlag.ControlThread</code> security permission for the thread to be aborted.

32

33 Permissions

Permission	Description
------------	-------------

System.Security.SecurityPermission

Requires permission to control the thread to be aborted.
See
System.Security.Permissions.SecurityPermissionFlag.
ControlThread.

1

2

1 Thread.Finalize() Method

```
2 [ILAsm]  
3 .method family hidebysig virtual void Finalize()  
4 [C#]  
5 ~Thread()
```

6 Summary

7 Releases the resources held by this instance.

8 Description

9 [Note: Application code does not call this method; it is automatically invoked during
10 garbage collection.]
11
12

13

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 Thread.GetDomain() Method

```
4 [ILAsm]  
5 .method public hidebysig static class System.AppDomain GetDomain()  
6 [C#]  
7 public static AppDomain GetDomain()
```

8 Summary

9 Returns an object representing the application domain in which the current thread is
10 executing.

11 Return Value

12 A System.AppDomain object that represents the current application domain.

13

1 Thread.Join() Method

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

6 Summary

7 Blocks the calling thread until the thread on which this method is invoked terminates.

8 Description

9 *[Note: Use this method to ensure a thread has terminated. The caller will block*
10 *indefinitely if the thread does not terminate.]*

11
12
13
14 `System.Threading.Thread.Join` cannot be invoked on a thread that is in the
15 `System.Threading.ThreadState.Unstarted` state.

16
17 This method changes the state of the calling thread to include
18 `System.Threading.ThreadState.WaitSleepJoin`.

19 Exceptions

Exception	Condition
System.Threading.ThreadStateException	The caller attempted to join a thread that is in the <code>System.Threading.ThreadState.Unstarted</code> state.

20

21

1 Thread.Join(System.TimeSpan) Method

```
2 [ILAsm]  
3 .method public hidebysig instance bool Join(valuetype System.TimeSpan  
4 timeout)  
5 [C#]  
6 public bool Join(TimeSpan timeout)
```

7 Summary

8 Blocks the calling thread until the thread on which this method is invoked terminates or
9 the specified time elapses.

10 Parameters

Parameter	Description
<i>timeout</i>	A System.TimeSpan set to the amount of time to wait for the thread to terminate. Specify System.Threading.Timeout.Infinite milliseconds to wait indefinitely.

11

12 Return Value

13 true if the thread has terminated; false if the thread has not terminated after the
14 amount of time specified by *timeout* has elapsed.

15 Description

16 This method converts *timeout* to milliseconds, tests the validity of the converted value,
17 and calls System.Threading.Thread.Join(System.Int32).
18

19 [Note: If System.Threading.Timeout.Infinite milliseconds is specified for *timeout*,
20 this method behaves identically to Join (), except for the return value.]
21
22
23

24 Join cannot be invoked on a thread that is in the
25 System.Threading.ThreadState.Unstarted state.
26

27 This method changes the state of the current thread to include
28 System.Threading.ThreadState.WaitSleepJoin.

29 Exceptions

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

System.ArgumentOutOfRangeException	The value of <i>timeout</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> milliseconds, or is greater than <code>System.Int32.MaxValue</code> milliseconds.
System.Threading.ThreadStateException	The caller attempted to join a thread that is in the <code>System.Threading.ThreadState.Unstarted</code> state.

1

2

1 Thread.Join(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig instance bool Join(int32 millisecondsTimeout)  
4 [C#]  
5 public bool Join(int millisecondsTimeout)
```

6 Summary

7 Blocks the calling thread until the thread on which this method is invoked terminates or
8 the specified time elapses.

9 Parameters

Parameter	Description
<i>millisecondsTimeout</i>	A System.Int32 containing the number of milliseconds to wait for the thread to terminate.

10

11 Return Value

12 true if the thread has terminated; false if the thread has not terminated after
13 *millisecondsTimeout* has elapsed.

14 Description

15 [Note: If System.Threading.Timeout.Infinite is specified for *millisecondsTimeout*,
16 this method behaves identically to Join (), except for the return value.]

17

18

19

20 Join cannot be invoked on a thread that is in the
21 System.Threading.ThreadState.Unstarted state.

22

23 This method changes the state of the calling thread to include
24 System.Threading.ThreadState.WaitSleepJoin.

25 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	The value of <i>millisecondsTimeout</i> is negative and is not equal to System.Threading.Timeout.Infinite.

System.Threading.ThreadStateException

The caller attempted to join a thread that is in the `System.Threading.ThreadState.Unstarted` state.

1

2

1 Thread.MemoryBarrier() Method

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

6 Summary

7 Guarantees that all subsequent loads or stores from the current thread will not access
8 memory until after all previous loads and stores from the current thread have
9 completed, as observed from this or other threads.

10

1 Thread.ResetAbort() Method

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

6 Summary

7 Cancels a `System.Threading.Thread.Abort` requested for the current thread.

8 Description

9 This method cannot be called by untrusted code.

10
11 When a call is made to `System.Threading.Thread.Abort` to destroy a thread, the
12 system throws a `System.Threading.ThreadAbortException`.
13 `System.Threading.ThreadAbortException` is a special exception that can be caught by
14 application code, but is rethrown at the end of the catch block unless `ResetAbort` is
15 called. `ResetAbort` cancels the request to abort, and prevents the
16 `ThreadAbortException` from terminating the thread.

17 Exceptions

Exception	Condition
<code>System.Threading.ThreadStateException</code>	<code>System.Threading.Thread.Abort</code> was not invoked on the current thread.
<code>System.Security.SecurityException</code>	Caller does not have <code>System.Security.Permissions.SecurityPermissionFlag.ControlThread</code> security permission for the current thread.

18 19 Example

20 For an example that demonstrates calling this method, see
21 `System.Threading.ThreadAbortException`.

22 Permissions

Permission	Description
<code>System.Security.SecurityPermission</code>	Requires permission to control the current thread. See <code>System.Security.Permissions.SecurityPermissionFlag.ControlThread</code> .

1

2

1 Thread.Sleep(System.Int32) Method

```
2 [ILAsm]  
3 .method public hidebysig static void Sleep(int32 millisecondsTimeout)  
4 [C#]  
5 public static void Sleep(int millisecondsTimeout)
```

6 Summary

7 Blocks the current thread for the specified number of milliseconds.

8 Parameters

Parameter	Description
<i>millisecondsTimeout</i>	A <code>System.Int32</code> containing the number of milliseconds for which the thread is blocked. Specify zero to indicate that this thread should be suspended temporarily to allow other waiting threads to execute. Specify <code>System.Threading.Timeout.Infinite</code> to block the thread indefinitely.

9 10 Description

11 The thread will not be scheduled for execution by the operating system for the amount
12 of time specified. This method changes the state of the thread to include
13 `System.Threading.ThreadState.WaitSleepJoin`.

14 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	The value of <i>millisecondsTimeout</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> .

15
16

1 Thread.Sleep(System.TimeSpan) Method

```
2 [ILAsm]  
3 .method public hidebysig static void Sleep(valuetype System.TimeSpan  
4 timeout)  
5 [C#]  
6 public static void Sleep(TimeSpan timeout)
```

7 Summary

8 Blocks the current thread for a specified time.

9 Parameters

Parameter	Description
<i>timeout</i>	A <code>System.TimeSpan</code> set to the amount of time for which the current thread will be blocked. Specify zero to indicate that this thread should be suspended temporarily to allow other waiting threads to execute. Specify <code>System.Threading.Timeout.Infinite</code> milliseconds to suspend the thread indefinitely.

10

11 Description

12 This method converts *timeout* to milliseconds, tests the validity of the converted value,
13 and calls `System.Threading.Thread.Sleep(System.Int32)`.

14

15 The thread will not be scheduled for execution by the operating system for the amount
16 of time specified. This method changes the state of the thread to include
17 `System.Threading.ThreadState.WaitSleepJoin`.

18 Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	The value of <i>timeout</i> is negative and is not equal to <code>System.Threading.Timeout.Infinite</code> milliseconds, or is greater than <code>System.Int32.MaxValue</code> milliseconds.

19

20

1 Thread.Start() Method

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

6 Summary

7 Causes the operating system to consider the thread ready to be scheduled for execution.

8 Description

9 Calling `System.Threading.Thread.Start` removes the
10 `System.Threading.ThreadState.Unstarted` state from the
11 `System.Threading.Thread.ThreadState` of the thread.

12
13 Once a thread is started, the operating system can schedule it for execution. When the
14 thread begins executing, the `System.Threading.ThreadStart` delegate supplied to the
15 constructor for the thread invokes its methods.

16
17 Once the thread terminates, it cannot be restarted with another call to
18 `System.Threading.Thread.Start`.

19 Exceptions

Exception	Condition
System.OutOfMemoryException	There is not enough memory available to start the thread.
System.NullReferenceException	This method was invoked on a null thread reference.
System.Threading.ThreadStateException	The thread has already been started.

20

21 Example

22 The following example demonstrates creating a thread and starting it.

```
23 [C#]  
24  
25 using System;  
26 using System.Threading;  
27 public class ThreadWork {  
28     public static void DoWork() {  
29         for (int i = 0; i<3;i++) {
```

```
1         Console.WriteLine ("Working thread...");
2         Thread.Sleep(100);
3     }
4 }
5 }
6 class ThreadTest{
7     public static void Main() {
8         ThreadStart myThreadDelegate = new ThreadStart(ThreadWork.DoWork);
9         Thread myThread = new Thread(myThreadDelegate);
10        myThread.Start();
11        for (int i = 0; i<3; i++) {
12            Console.WriteLine("In main.");
13            Thread.Sleep(100);
14        }
15    }
16 }
17
```

18 One possible set of output is

```
19
20 In main.
21
22
23 Working thread...
24
25
26 In main.
27
28
29 Working thread...
30
31
32 In main.
33
34
35 Working thread...
36
37
```

38 Note that the sequence of the output statements is not guaranteed to be identical across
39 systems.

40

1 Thread.VolatileRead(System.Object&)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static object VolatileRead (object& address)  
5 [C#]  
6 public static object VolatileRead (ref object address)
```

7 Summary

8 Performs a volatile read from the specified address.

9 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Object</code> that specifies the address in memory from which to read.

10

11 Return Value

12 A `System.Object` containing the value at the specified address after any pending writes.

13 Description

14 The value at the given address is atomically loaded with acquire semantics, meaning
15 that the read is guaranteed to occur prior to any references to memory that occur after
16 the execution of this method in the current thread. It is recommended that
17 `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]
25
26

27

1 Thread.VolatileRead(System.Double& 2 Method

```
3 [ILAsm]  
4 .method public hidebysig static float64 VolatileRead (float64& address)  
5 [C#]  
6 public static double VolatileRead (ref double address)
```

7 Summary

8 Performs a volatile read from the specified address.

9 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Double</code> that specifies the address in memory from which to read.

10 11 Return Value

12 A `System.Double` containing the value at the specified address after any pending writes.

13 Description

14 The value at the given address is atomically loaded with acquire semantics, meaning
15 that the read is guaranteed to occur prior to any references to memory that occur after
16 the execution of this method in the current thread. It is recommended that
17 `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]
25
26
27

1 Thread.VolatileRead(System.Single&) Method

```
2 [ILAsm]  
3 .method public hidebysig static float32 VolatileRead (float32& address)  
4 [C#]  
5 public static float VolatileRead (ref float address)
```

6 Summary

7 Performs a volatile read from the specified address.

8 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Single</code> that specifies the address in memory from which to read.

9 Return Value

11 A `System.Single` containing the value at the specified address after any pending writes.

12 Description

13 The value at the given address is atomically loaded with acquire semantics, meaning
14 that the read is guaranteed to occur prior to any references to memory that occur after
15 the execution of this method in the current thread. It is recommended that
16 `System.Threading.Thread.VolatileRead` and
17 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
18 affects only this single access; other accesses to the same location are required to also
19 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
20 semantics are to be preserved. This method has exactly the same semantics as using
21 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
22 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
23 Partition I of the CLI Specification.]
24
25

26

1 Thread.VolatileRead(System.UInt64&)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static unsigned int64 VolatileRead (unsigned  
5 int64& address)  
  
6 [C#]  
7 public static ulong VolatileRead (ref ulong address)
```

8 Summary

9 Performs a volatile read from the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt64</code> that specifies the address in memory from which to read.

11 Return Value

13 A `System.UInt64` containing the value at the specified address after any pending writes.

14 Description

15 The value at the given address is atomically loaded with acquire semantics, meaning
16 that the read is guaranteed to occur prior to any references to memory that occur after
17 the execution of this method in the current thread. It is recommended that
18 `System.Threading.Thread.VolatileRead` and
19 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
20 affects only this single access; other accesses to the same location are required to also
21 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
22 semantics are to be preserved. This method has exactly the same semantics as using
23 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
24 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
25 Partition I of the CLI Specification.]
26
27

28

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 Thread.VolatileRead(System.UIntPtr&) 4 Method

```
5 [ILAsm]  
6 .method public hidebysig static uintPtr VolatileRead (class  
7 System.UIntPtr& address)  
8 [C#]  
9 public static UIntPtr VolatileRead (ref UIntPtr address)
```

10 Summary

11 Performs a volatile read from the specified address.

12 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UIntPtr</code> that specifies the address in memory from which to read.

14 Return Value

15 A `System.UIntPtr` containing the value at the specified address after any pending
16 writes.

17 Description

18 The value at the given address is atomically loaded with acquire semantics, meaning
19 that the read is guaranteed to occur prior to any references to memory that occur after
20 the execution of this method in the current thread. It is recommended that
21 `System.Threading.Thread.VolatileRead` and
22 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
23 affects only this single access; other accesses to the same location are required to also
24 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
25 semantics are to be preserved. This method has exactly the same semantics as using
26 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
27 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
28 Partition I of the CLI Specification.]
29
30

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 Thread.VolatileRead(System.IntPtr&) Method

```
4 [ILAsm]  
5 .method public hidebysig static intptr VolatileRead (class System.IntPtr&  
6 address)  
  
7 [C#]  
8 public static IntPtr VolatileRead (ref IntPtr address)
```

9 Summary

10 Performs a volatile read from the specified address.

11 Parameters

Parameter	Description
<i>address</i>	A reference to a System.IntPtr that specifies the address in memory from which to read.

13 Return Value

14 A System.IntPtr containing the value at the specified address after any pending writes.

15 Description

16 The value at the given address is atomically loaded with acquire semantics, meaning
17 that the read is guaranteed to occur prior to any references to memory that occur after
18 the execution of this method in the current thread. It is recommended that
19 System.Threading.Thread.VolatileRead and
20 System.Threading.Thread.VolatileWrite be used in conjunction. Calling this method
21 affects only this single access; other accesses to the same location are required to also
22 be made using this method or System.Threading.Thread.VolatileWrite if the volatile
23 semantics are to be preserved. This method has exactly the same semantics as using
24 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
25 types, not just those 32 bits or smaller in size. [Note: For additional information, see
26 Partition I of the CLI Specification.]
27
28

1 Thread.VolatileRead(System.UInt32&)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static unsigned int32 VolatileRead (unsigned  
5 int32& address)  
  
6 [C#]  
7 public static uint VolatileRead (ref uint address)
```

8 Summary

9 Performs a volatile read from the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt32</code> that specifies the address in memory from which to read.

11 Return Value

12 A `System.UInt32` containing the value at the specified address after any pending writes.

14 Description

15 The value at the given address is atomically loaded with acquire semantics, meaning
16 that the read is guaranteed to occur prior to any references to memory that occur after
17 the execution of this method in the current thread. It is recommended that
18 `System.Threading.Thread.VolatileRead` and
19 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
20 affects only this single access; other accesses to the same location are required to also
21 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
22 semantics are to be preserved. This method has exactly the same semantics as using
23 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
24 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
25 Partition I of the CLI Specification.]
26
27

28

1 Thread.VolatileRead(System.UInt16&)

2 Method

```
3 [ILAsm]  
4 .method public hidebysig static unsigned int16 VolatileRead (unsigned  
5 int16& address)  
  
6 [C#]  
7 public static ushort VolatileRead (ref ushort address)
```

8 Summary

9 Performs a volatile read from the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt16</code> that specifies the address in memory from which to read.

11 Return Value

12 A `System.UInt16` containing the value at the specified address after any pending writes.

14 Description

15 The value at the given address is atomically loaded with acquire semantics, meaning
16 that the read is guaranteed to occur prior to any references to memory that occur after
17 the execution of this method in the current thread. It is recommended that
18 `System.Threading.Thread.VolatileRead` and
19 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
20 affects only this single access; other accesses to the same location are required to also
21 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
22 semantics are to be preserved. This method has exactly the same semantics as using
23 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
24 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
25 Partition I of the CLI Specification.]
26
27

28

1 Thread.VolatileRead(System.SByte&) Method

```
2 [ILAsm]  
3 .method public hidebysig static sbyte VolatileRead (class System.Sbyte&  
4 address)  
  
5 [C#]  
6 public static sbyte VolatileRead (ref sbyte address)
```

7 Summary

8 Performs a volatile read from the specified address.

9 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.SByte</code> that specifies the address in memory from which to read.

10

11 Return Value

12 A `System.SByte` containing the value at the specified address after any pending writes.

13 Description

14 The value at the given address is atomically loaded with acquire semantics, meaning
15 that the read is guaranteed to occur prior to any references to memory that occur after
16 the execution of this method in the current thread. It is recommended that
17 `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]
25
26

27

1 Thread.VolatileRead(System.Int64&) Method

```
2 [ILAsm]  
3 .method public hidebysig static int64 VolatileRead (int64& address)  
4 [C#]  
5 public static long VolatileRead (ref long address)
```

6 Summary

7 Performs a volatile read from the specified address.

8 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int64</code> that specifies the address in memory from which to read.

9 Return Value

11 A `System.Int64` containing the value at the specified address after any pending writes.

12 Description

13 The value at the given address is atomically loaded with acquire semantics, meaning
14 that the read is guaranteed to occur prior to any references to memory that occur after
15 the execution of this method in the current thread. It is recommended that
16 `System.Threading.Thread.VolatileRead` and
17 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
18 affects only this single access; other accesses to the same location are required to also
19 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
20 semantics are to be preserved. This method has exactly the same semantics as using
21 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
22 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
23 Partition I of the CLI Specification.]
24
25

26

1 Thread.VolatileRead(System.Int32&) Method

```
2 [ILAsm]  
3 .method public hidebysig static int32 VolatileRead (int32& address)  
4 [C#]  
5 public static int VolatileRead (ref int address)
```

6 Summary

7 Performs a volatile read from the specified address.

8 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int32</code> that specifies the address in memory from which to read.

9 Return Value

11 A `System.Int32` containing the value at the specified address after any pending writes.

12 Description

13 The value at the given address is atomically loaded with acquire semantics, meaning
14 that the read is guaranteed to occur prior to any references to memory that occur after
15 the execution of this method in the current thread. It is recommended that
16 `System.Threading.Thread.VolatileRead` and
17 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
18 affects only this single access; other accesses to the same location are required to also
19 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
20 semantics are to be preserved. This method has exactly the same semantics as using
21 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
22 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
23 Partition I of the CLI Specification.]
24
25

1 Thread.VolatileRead(System.Int16&) Method

```
2 [ILAsm]  
3 .method public hidebysig static int16 VolatileRead (int16& address)  
4 [C#]  
5 public static short VolatileRead (ref short address)
```

6 Summary

7 Performs a volatile read from the specified address.

8 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int16</code> that specifies the address in memory from which to read.

9 Return Value

10 A `System.Int16` containing the value at the specified address after any pending writes.

12 Description

13 The value at the given address is atomically loaded with acquire semantics, meaning
14 that the read is guaranteed to occur prior to any references to memory that occur after
15 the execution of this method in the current thread. It is recommended that
16 `System.Threading.Thread.VolatileRead` and
17 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
18 affects only this single access; other accesses to the same location are required to also
19 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
20 semantics are to be preserved. This method has exactly the same semantics as using
21 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
22 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
23 Partition I of the CLI Specification.]
24
25

26

1 Thread.VolatileRead(System.Byte&) Method

```
2 [ILAsm]  
3 .method public hidebysig static byte VolatileRead (class System.Byte&  
4 address)  
  
5 [C#]  
6 public static byte VolatileRead (ref byte address)
```

7 Summary

8 Performs a volatile read from the specified address.

9 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Byte</code> that specifies the address in memory from which to read.

10

11 Return Value

12 A `System.Byte` containing the value at the specified address after any pending writes.

13 Description

14 The value at the given address is atomically loaded with acquire semantics, meaning
15 that the read is guaranteed to occur prior to any references to memory that occur after
16 the execution of this method in the current thread. It is recommended that
17 `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileWrite` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the load CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]
25
26

27

1 Thread.VolatileWrite(System.UInt32&, System.UInt32) Method

```
3 [ILAsm]
4 .method public hidebysig static void VolatileWrite (unsigned int32&
5 address, unsigned int32 value)
6
7 [C#]
8 public static void VolatileWrite (ref uint address, uint value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt32</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.UInt32</code> that specifies the value to write.

11 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23
24
25

1 Thread.VolatileWrite(System.UInt64&, System.UInt64) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (unsigned int64&  
5 address, unsigned int64 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref ulong address, ulong value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt64</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.UInt64</code> that specifies the value to write.

11 12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]
23
24

25

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 Thread.VolatileWrite(System.UIntPtr&, 4 System.UIntPtr) Method

```
5 [ILAsm]  
6 .method public hidebysig static void VolatileWrite (class System.UIntPtr&  
7 address, UIntPtr value)  
8 [C#]  
9 public static void VolatileWrite (ref UIntPtr address, UIntPtr value)
```

10 Summary

11 Performs a volatile write to the specified address.

12 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UIntPtr</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.UIntPtr</code> that specifies the value to write.

14 Description

15 The value is written atomically to the specified address with release semantics, meaning
16 that the write is guaranteed to happen after any references to memory that occur prior
17 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]

1 **The following member must be implemented if the RuntimeInfrastructure library is**
2 **present in the implementation.**

3 Thread.VolatileWrite(System.IntPtr& 4 System.IntPtr) Method

```
5 [ILAsm]  
6 .method public hidebysig static void VolatileWrite (class System.IntPtr&  
7 address, IntPtr value)  
8 [C#]  
9 public static void VolatileWrite (ref IntPtr address, IntPtr value)
```

10 Summary

11 Performs a volatile write to the specified address.

12 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.IntPtr</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.IntPtr</code> that specifies the value to write.

14 Description

15 The value is written atomically to the specified address with release semantics, meaning
16 that the write is guaranteed to happen after any references to memory that occur prior
17 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
18 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
19 affects only this single access; other accesses to the same location are required to also
20 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
21 semantics are to be preserved. This method has exactly the same semantics as using
22 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
23 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
24 Partition I of the CLI Specification.]
25
26
27

1 Thread.VolatileWrite(System.Single&, System.Single) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (float32& address,  
5 float32 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref float address, float value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Single</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Single</code> that specifies the value to write.

11 12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23
24
25

1 Thread.VolatileWrite(System.Double&, System.Double) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (float64& address,  
5 float64 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref double address, double value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Double</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Double</code> that specifies the value to write.

11 12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23
24
25

1 Thread.VolatileWrite(System.Object&, 2 System.Object) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (object& address,  
5 object value)  
  
6 [C#]  
7 public static void VolatileWrite (ref object address, object value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Object</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Object</code> that specifies the value to write.

11

12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23

24

25

1 Thread.VolatileWrite(System.UInt16&, System.UInt16) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (unsigned int16&  
5 address, unsigned int16 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref ushort address, ushort value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.UInt16</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.UInt16</code> that specifies the value to write.

11

12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23

24

25

1 Thread.VolatileWrite(System.SByte&, System.SByte) Method

```
3 [ILAsm]
4 .method public hidebysig static void VolatileWrite (class System.SByte&
5 address, sbyte value)
6
7 [C#]
8 public static void VolatileWrite (ref sbyte address, sbyte value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.SByte</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.SByte</code> that specifies the value to write.

11 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23
24
25

1 Thread.VolatileWrite(System.Int64&, System.Int64) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (int64& address, int64  
5 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref long address, long value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int64</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Int64</code> that specifies the value to write.

11 12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23
24
25

1 Thread.VolatileWrite(System.Int32&, System.Int32) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (int32& address, int32  
5 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref int address, int value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int32</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Int32</code> that specifies the value to write.

11

12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23

24

25

1 Thread.VolatileWrite(System.Int16&, System.Int16) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (int16& address, int16  
5 value)  
  
6 [C#]  
7 public static void VolatileWrite (ref short address, short value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Int16</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Int16</code> that specifies the value to write.

11

12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]

23

24

25

1 Thread.VolatileWrite(System.Byte&, System.Byte) Method

```
3 [ILAsm]  
4 .method public hidebysig static void VolatileWrite (class System.Byte&  
5 address, byte value)  
  
6 [C#]  
7 public static void VolatileWrite (ref byte address, byte value)
```

8 Summary

9 Performs a volatile write to the specified address.

10 Parameters

Parameter	Description
<i>address</i>	A reference to a <code>System.Byte</code> that specifies the address in memory at which to write.
<i>value</i>	A <code>System.Byte</code> that specifies the value to write.

11 12 Description

13 The value is written atomically to the specified address with release semantics, meaning
14 that the write is guaranteed to happen after any references to memory that occur prior
15 to the execution. It is recommended that `System.Threading.Thread.VolatileRead` and
16 `System.Threading.Thread.VolatileWrite` be used in conjunction. Calling this method
17 affects only this single access; other accesses to the same location are required to also
18 be made using this method or `System.Threading.Thread.VolatileRead` if the volatile
19 semantics are to be preserved. This method has exactly the same semantics as using
20 the volatile prefix on the store CIL instruction, except that atomicity is provided for all
21 types, not just those 32 bits or smaller in size. [*Note:* For additional information, see
22 Partition I of the CLI Specification.]
23
24

25

1 Thread.CurrentThread Property

```
2 [ILAsm]  
3 .property class System.Threading.Thread CurrentThread { public hidebysig  
4 static specialname class System.Threading.Thread get_CurrentThread() }  
  
5 [C#]  
6 public static Thread CurrentThread { get; }
```

7 Summary

8 Gets a `System.Threading.Thread` instance that represents the currently executing
9 thread.

10 Property Value

11 An instance of `System.Threading.Thread` representing the current thread.

12 Description

13 This property is read-only.

14

1 Thread.IsAlive Property

```
2 [ILAsm]  
3 .property bool IsAlive { public hidebysig specialname instance bool  
4 get_IsAlive() }  
  
5 [C#]  
6 public bool IsAlive { get; }
```

7 Summary

8 Gets a `System.Boolean` value indicating the execution status of the current thread.

9 Property Value

10 `true` if this thread has been started, and has not terminated; otherwise, `false`.

11 Description

12 This property is read-only.

13

1 Thread.IsBackground Property

```
2 [ILAsm]
3 .property bool IsBackground { public hidebysig specialname instance bool
4 get_IsBackground() public hidebysig specialname instance void
5 set_IsBackground(bool value) }
6
7 [C#]
8 public bool IsBackground { get; set; }
```

8 Summary

9 Gets or sets a `System.Boolean` value indicating whether a thread is a background
10 thread.

11 Property Value

12 `true` if the thread is or is to become a background thread; otherwise, `false`.

13 Description

14 The default value of this property is `false`. The property value can be changed before
15 the thread is started and before it terminates.

16
17 [*Note:* A thread is either a background thread or a foreground thread. Background
18 threads are identical to foreground threads except for the fact that background threads
19 do not prevent a process from terminating. Once all foreground threads belonging to a
20 process have terminated, the execution engine ends the process by invoking
21 `System.Threading.Thread.Abort` on any background threads that are still alive.]
22
23

24 Exceptions

Exception	Condition
System.Threading.ThreadStateException	The thread has reached the <code>System.Threading.ThreadState.Stopped</code> state.

25

26

1 Thread.Name Property

```
2 [ILAsm]  
3 .property string Name { public hidebysig specialname instance string  
4 get_Name() public hidebysig specialname instance void set_Name(string  
5 value) }  
6 [C#]  
7 public string Name { get; set; }
```

8 Summary

9 Gets or sets the name of the thread.

10 Property Value

11 A System.String containing the name of the thread, or null if no name was set.

12 Description

13 This property is write-once. Once this property has been set to a non-null value,
14 attempts to set this property to a new value cause an exception.

15 Exceptions

Exception	Condition
System.InvalidOperationException	A set operation was requested, and the Name property has already been set.

16

17

1 Thread.Priority Property

```
2 [ILAsm]
3 .property valuetype System.Threading.ThreadPriority Priority { public
4 hidebysig specialname instance valuetype System.Threading.ThreadPriority
5 get_Priority() public hidebysig specialname instance void
6 set_Priority(valuetype System.Threading.ThreadPriority value) }

7 [C#]
8 public ThreadPriority Priority { get; set; }
```

9 Summary

10 Gets or sets a value indicating the scheduling priority of a thread.

11 Property Value

12 A `System.Threading.ThreadPriority` value.

13 Description

14 A thread can be assigned any one of the following priority values:

- 15 • `System.Threading.ThreadPriority.Highest`
- 16 • `System.Threading.ThreadPriority.AboveNormal`
- 17 • `System.Threading.ThreadPriority.Normal`
- 18 • `System.Threading.ThreadPriority.BelowNormal`
- 19 • `System.Threading.ThreadPriority.Lowest`

20 The default value is `System.Threading.ThreadPriority.Normal`.

21

22 Operating systems are not required to honor the priority of a thread.

23 Exceptions

Exception	Condition
System.Threading.ThreadStateException	The thread is in the <code>System.Threading.ThreadState.Stopped</code> state.
System.ArgumentException	The value specified for a set operation is not a valid <code>System.Threading.ThreadPriority</code>

	value.
--	--------

1

2

1 Thread.ThreadState Property

```
2 [ILAsm]  
3 .property valuetype System.Threading.ThreadState ThreadState { public  
4 hidebysig specialname instance valuetype System.Threading.ThreadState  
5 get_ThreadState() }  
6 [C#]  
7 public ThreadState ThreadState { get; }
```

8 Summary

9 Gets a value containing the states of the current thread.

10 Property Value

11 A combination of one or more `System.Threading.ThreadState` values, which indicate
12 the state of the current thread.

13 Description

14 This property is read-only.

15
16 A thread is running if the value returned by this property does not include
17 `System.Threading.ThreadState.Unstarted` and does not include
18 `System.Threading.ThreadState.Stopped`.

19