System.Threading.Thread Class

```
[ILASM]
.class public sealed Thread extends System.Object

[C#]
public sealed class Thread
```

Assembly Info:

- *Name:* mscorlib
- *Public Key:* [00 00 00 00 00 00 00 00 04 00 00 00 00 00 00 00]
- *Version:* 1.0.x.x
- *Attributes:*
  - CLSCompliantAttribute(true)

Summary

Represents a sequential thread of execution.

Inherits From: System.Object

Library: BCL

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

Description

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

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

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

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

Thread(System.Threading.ThreadStart) Constructor

```
[ILASM]
public rtspecialname specialname instance void .ctor(class
System.Threading.ThreadStart start)

[C#]
public Thread(ThreadStart start)
```

Summary

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

Parameters

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

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

Until **System.Threading.Thread.Start** is called, the thread's state includes **System.Threading.ThreadState.Unstarted**.

Exceptions

Exception	Condition
System.ArgumentNullException	*start* is **null**.

Thread.Abort(System.Object) Method

```
[ILASM]
.method public hidebysig instance void Abort(object
stateInfo)

[C#]
public void Abort(object stateInfo)
```

Summary

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

Parameters

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

Description

The object passed as the *stateInfo* parameter can be obtained by accessing the **System.Threading.ThreadAbortException.ExceptionState** property.

[*Note:* For details on aborting threads, see **System.Threading.Thread.Abort** ().]

Exceptions

Exception	Condition
System.Security.SecurityException	Caller does not have **System.Security.Permissions.SecurityPermissio ControlThread** security permission for this thread.

Permissions

Permission	Description
System.Security.SecurityPermission	Requires permission to control the thread to be ab See

	System.Security.Permissions.SecurityPermissi **ControlThread**.

1
2
3

Thread.Abort() Method

```
[ILASM]
.method public hidebysig instance void Abort()

[C#]
public void Abort()
```

Summary

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

Description

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

- unmanaged code
- a catch handler
- a finally clause
- a filter clause
- a type initializer

A thread abort proceeds as follows:

1. An abort begins at the earliest of the following times:

   a. when the thread transitions from unmanaged to managed code execution;

   b. when the thread finishes the outermost currently executing catch handler;

   c. immediately if the thread is running managed code outside of any catch handler, finally clause, filter clause or type initializer

2. Whenever an outermost catch handler finishes execution, the **System.Threading.ThreadAbortException** is rethrown unless the thread being aborted has called **System.Threading.Thread.ResetAbort** since the call to **System.Threading.Thread.Abort**.

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

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

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

After **System.Threading.Thread.Abort** is invoked on a thread, the state of the thread includes **System.Threading.ThreadState.AbortRequested**. After the thread has terminated as a result of a successful call to **System.Threading.Thread.Abort**, the state of the thread includes **System.Threading.ThreadState.Stopped** and **System.Threading.ThreadState.Aborted**.

[*Note:* With sufficient permissions, a thread that is the target of a **System.Threading.Thread.Abort** can cancel the abort using the **System.Threading.Thread.ResetAbort** method. For an example that demonstrates calling the **System.Threading.Thread.ResetAbort** method, see **System.Threading.ThreadAbortException**.]

Exceptions

Exception	Condition
System.Security.SecurityException	Caller does not have **System.Security.Permissions.SecurityPermissio ControlThread** security permission for the thread t aborted.

Permissions

Permission	Description
System.Security.SecurityPermission	Requires permission to control the thread to be ab See **System.Security.Permissions.SecurityPermissi ControlThread**.

Thread.Finalize() Method

```
[ILASM]
.method family hidebysig virtual void Finalize()

[C#]
~Thread()
```

Summary

Releases the resources held by this instance.

Description

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

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

Thread.GetDomain() Method

[ILASM]
```
.method public hidebysig static class System.AppDomain
GetDomain()
```

[C#]
```
public static AppDomain GetDomain()
```

Summary

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

Return Value

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

Thread.Join() Method

```
[ILASM]
.method public hidebysig instance void Join()

[C#]
public void Join()
```

Summary

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

Description

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

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

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

Exceptions

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

Thread.Join(System.Int32) Method

```
[ILASM]
.method public hidebysig instance bool Join(int32
millisecondsTimeout)

[C#]
public bool Join(int millisecondsTimeout)
```

Summary

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

Parameters

Parameter	Description
millisecondsTimeout	A **System.Int32** containing the number of milliseconds to wait for the thread to terminate.

Return Value

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

Description

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

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

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

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	The value of *millisecondsTimeout* 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
3

Thread.Join(System.TimeSpan) Method

```
[ILASM]
.method public hidebysig instance bool Join(valuetype
System.TimeSpan timeout)

[C#]
public bool Join(TimeSpan timeout)
```

Summary

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

Parameters

Parameter	Description
timeout	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.

Return Value

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

Description

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

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

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

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

Exceptions

Exception	Condition

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

1
2
3

Thread.MemoryBarrier () Method

```
[ILASM]
.method public hidebysig static void MemoryBarrier ()

[C#]
public static void MemoryBarrier ()
```

Summary

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

Thread.ResetAbort() Method

```
[ILASM]
.method public hidebysig static void ResetAbort()

[C#]
public static void ResetAbort()
```

Summary

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

Description

This method cannot be called by untrusted code.

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

Exceptions

Exception	Condition
System.Threading.ThreadStateException	**System.Threading.Thread.Abort** was not in current thread.
System.Security.SecurityException	Caller does not have **System.Security.Permissions.SecurityPer ControlThread** security permission for the cu

Example

For an example that demonstrates calling this method, see **System.Threading.ThreadAbortException**.

Permissions

Permission	Description
System.Security.SecurityPermission	Requires permission to control the current thread. **Svstem.Securitv.Permissions.SecuritvPermissi**

	ControlThread.

1
2
3

Thread.Sleep(System.Int32) Method

```
[ILASM]
.method public hidebysig static void Sleep(int32
millisecondsTimeout)

[C#]
public static void Sleep(int millisecondsTimeout)
```

Summary

Blocks the current thread for the specified number of milliseconds.

Parameters

Parameter	Description
millisecondsTimeout	A **System.Int32** 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 **System.Threading.Timeout.Infinite** to block the thread indefinitely.

Description

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

Exceptions

Exception	Condition
System.ArgumentOutOfRangeException	The value of *millisecondsTimeout* is negative and is not equal to **System.Threading.Timeout.Infinite**.

Thread.Sleep(System.TimeSpan) Method

```
[ILASM]
.method public hidebysig static void Sleep(valuetype
System.TimeSpan timeout)

[C#]
public static void Sleep(TimeSpan timeout)
```

Summary

Blocks the current thread for a specified time.

Parameters

Parameter	Description
timeout	A **System.TimeSpan** 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 **System.Threading.Timeout.Infinite** milliseconds to suspend the thread indefinitely.

Description

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

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

Exceptions

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

Thread.Start() Method

```
[ILASM]
.method public hidebysig instance void Start()

[C#]
public void Start()
```

Summary

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

Description

Calling **System.Threading.Thread.Start** removes the **System.Threading.ThreadState.Unstarted** state from the **System.Threading.Thread.ThreadState** of the thread.

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

Once the thread terminates, it cannot be restarted with another call to **System.Threading.Thread.Start**.

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.

Example

The following example demonstrates creating a thread and starting it.

[C#]

```
using System;
using System.Threading;
public class ThreadWork {
  public static void DoWork() {
    for (int i = 0; i<3;i++) {
```

```
            Console.WriteLine ("Working thread...");
            Thread.Sleep(100);
        }
      }
    }
    class ThreadTest{
      public static void Main() {
        ThreadStart myThreadDelegate = new
    ThreadStart(ThreadWork.DoWork);
        Thread myThread = new Thread(myThreadDelegate);
        myThread.Start();
        for (int i = 0; i<3; i++) {
            Console.WriteLine("In main.");
            Thread.Sleep(100);
        }
      }
    }
```

One possible set of output is

```
In main.


Working thread...


In main.


Working thread...


In main.
```

```
Working thread...
```

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

Thread.VolatileRead(System.Byte&) Method

```
[ILASM]
.method public hidebysig static byte VolatileRead (class
System.Byte& address)

[C#]
public static byte VolatileRead (ref byte address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Byte** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Int16&) Method

```
[ILASM]
.method public hidebysig static int16 VolatileRead (class
System.Int16& address)

[C#]
public static short VolatileRead (ref short address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Int16** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Int32&) Method

```
[ILASM]
.method public hidebysig static int32 VolatileRead (class
System.Int32& address)

[C#]
public static int VolatileRead (ref int address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Int32** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Int64&) Method

```
[ILASM]
.method public hidebysig static int64 VolatileRead (class
System.Int64& address)

[C#]
public static long VolatileRead (ref long address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Int64** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.SByte&) Method

```
[ILASM]
.method public hidebysig static sbyte VolatileRead (class
System.Sbyte& address)

[C#]
public static sbyte VolatileRead (ref sbyte address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.SByte** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.UInt16&) Method

```
[ILASM]
.method public hidebysig static uint16 VolatileRead (class
System.UInt16& address)

[C#]
public static ushort VolatileRead (ref ushort address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.UInt16** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.UInt32&) Method

```
[ILASM]
.method public hidebysig static uint32 VolatileRead (class
System.UInt32& address)

[C#]
public static uint VolatileRead (ref uint address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.UInt32** that specifies the address in memory from which to read.

Return Value

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

Description

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

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

Thread.VolatileRead(System.IntPtr&) Method

```
[ILASM]
.method public hidebysig static intptr VolatileRead (class
System.IntPtr& address)

[C#]
public static IntPtr VolatileRead (ref IntPtr address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.IntPtr** that specifies the address in memory from which to read.

Return Value

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

Description

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

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

Thread.VolatileRead(System.UIntPtr&) Method

```
[ILASM]
.method public hidebysig static uintPtr VolatileRead (class
System.UIntPtr& address)

[C#]
public static UIntPtr VolatileRead (ref UIntPtr address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.UIntPtr** that specifies the address in memory from which to read.

Return Value

A **System.UIntPtr** containing the value at the specified address after any pending writes.

Description

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

Thread.VolatileRead(System.UInt64&) Method

```
[ILASM]
.method public hidebysig static uint64 VolatileRead (class
System.UInt64& address)

[C#]
public static ulong VolatileRead (ref ulong address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.UInt64** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Single&) Method

```
[ILASM]
.method public hidebysig static float32 VolatileRead (class
System.Single& address)

[C#]
public static float VolatileRead (ref float address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Single** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Double&) Method

```
[ILASM]
.method public hidebysig static float64 VolatileRead (class
System.Double& address)

[C#]
public static double VolatileRead (ref double address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Double** that specifies the address in memory from which to read.

Return Value

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

Description

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

Thread.VolatileRead(System.Object&) Method

```
[ILASM]
.method public hidebysig static object VolatileRead (class
System.Object& address)

[C#]
public static object VolatileRead (ref object address)
```

Summary

Performs a volatile read from the specified address.

Parameters

Parameter	Description
address	A **System.Object** that specifies the address in memory from which to read.

Return Value

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

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Byte& address, byte value)

[C#]
public static void VolatileWrite (ref byte address, byte
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Byte** that specifies the address in memory at which to write.
value	A **System.Byte** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Int16& address, int16 value)

[C#]
public static void VolatileWrite (ref short address, short
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Int16** that specifies the address in memory at which to write.
value	A **System.Int16** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Int32& address, int32 value)

[C#]
public static void VolatileWrite (ref int address, int
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Int32** that specifies the address in memory at which to write.
value	A **System.Int32** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Int64& address, int64 value)

[C#]
public static void VolatileWrite (ref long address, long
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Int64** that specifies the address in memory at which to write.
value	A **System.Int64** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.SByte& address, sbyte value)

[C#]
public static void VolatileWrite (ref sbyte address, sbyte
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.SByte** that specifies the address in memory at which to write.
value	A **System.SByte** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.UInt16& address, uint16 value)

[C#]
public static void VolatileWrite (ref ushort address,
ushort value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.UInt16** that specifies the address in memory at which to write.
value	A **System.UInt16** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.UInt32& address, uint32 value)

[C#]
public static void VolatileWrite (ref uint address, uint
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.UInt32** that specifies the address in memory at which to write.
value	A **System.UInt32** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.UInt64& address, uint64 value)

[C#]
public static void VolatileWrite (ref ulong address, ulong
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.UInt64** that specifies the address in memory at which to write.
value	A **System.UInt64** that specifies the value to write.

Description

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

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.UIntPtr& address, UIntPtr value)

[C#]
public static void VolatileWrite (ref UIntPtr address,
UIntPtr value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.UIntPtr** that specifies the address in memory at which to write.
value	A **System.UIntPtr** that specifies the value to write.

Description

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

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

Thread.VolatileWrite(System.IntPtr&, System.IntPtr) Method

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.IntPtr& address, IntPtr value)

[C#]
public static void VolatileWrite (ref IntPtr address,
IntPtr value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.IntPtr** that specifies the address in memory at which to write.
value	A **System.IntPtr** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Single& address, float32 value)

[C#]
public static void VolatileWrite (ref float address, float
value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Single** that specifies the address in memory at which to write.
value	A **System.Single** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Double& address, float64 value)

[C#]
public static void VolatileWrite (ref double address,
double value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Double** that specifies the address in memory at which to write.
value	A **System.Double** that specifies the value to write.

Description

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

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

```
[ILASM]
.method public hidebysig static void VolatileWrite (class
System.Object& address, object value)

[C#]
public static void VolatileWrite (ref object address,
object value)
```

Summary

Performs a volatile write to the specified address.

Parameters

Parameter	Description
address	A **System.Object** that specifies the address in memory at which to write.
value	A **System.Object** that specifies the value to write.

Description

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

Thread.CurrentThread Property

[ILASM]
```
.property class System.Threading.Thread CurrentThread {
public hidebysig static specialname class
System.Threading.Thread get_CurrentThread() }
```

[C#]
```
public static Thread CurrentThread { get; }
```

Summary

Gets a **System.Threading.Thread** instance that represents the currently executing thread.

Property Value

An instance of **System.Threading.Thread** representing the current thread.

Description

This property is read-only.

Thread.IsAlive Property

```
[ILASM]
.property bool IsAlive { public hidebysig specialname
instance bool get_IsAlive() }

[C#]
public bool IsAlive { get; }
```

Summary

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

Property Value

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

Description

This property is read-only.

This property returns true unless the **System.Threading.Thread.ThreadState** of the thread contains **System.Threading.ThreadState.Unstarted**, or **System.Threading.ThreadState.Stopped**.

Thread.IsBackground Property

```
[ILASM]
.property bool IsBackground { public hidebysig specialname
instance bool get_IsBackground() public hidebysig
specialname instance void set_IsBackground(bool value) }

[C#]
public bool IsBackground { get; set; }
```

Summary

Gets or sets a **System.Boolean** value indicating whether a thread is a background thread.

Property Value

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

Description

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

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

Exceptions

Exception	Condition
System.Threading.ThreadStateException	The thread has reached the **System.Threading.ThreadState.Stopped** state.

Thread.Name Property

```
[ILASM]
.property string Name { public hidebysig specialname
instance string get_Name() public hidebysig specialname
instance void set_Name(string value) }

[C#]
public string Name { get; set; }
```

Summary

Gets or sets the name of the thread.

Property Value

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

Description

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

Exceptions

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

Thread.Priority Property

```
[ILASM]
.property valuetype System.Threading.ThreadPriority
Priority { public hidebysig specialname instance valuetype
System.Threading.ThreadPriority get_Priority() public
hidebysig specialname instance void set_Priority(valuetype
System.Threading.ThreadPriority value) }

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

Summary

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

Property Value

A **System.Threading.ThreadPriority** value.

Description

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

- **System.Threading.ThreadPriority.Highest**
- **System.Threading.ThreadPriority.AboveNormal**
- **System.Threading.ThreadPriority.Normal**
- **System.Threading.ThreadPriority.BelowNormal**
- **System.Threading.ThreadPriority.Lowest**

The default value is **System.Threading.ThreadPriority.Normal**.

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

Exceptions

Exception	Condition
System.Threading.ThreadStateException	The thread is in the **System.Threading.ThreadState.Stopped** state.
System.ArgumentException	The value specified for a set operation is not a valid

	System.Threading.ThreadPriority value.

1
2
3

Thread.ThreadState Property

```
[ILASM]
.property valuetype System.Threading.ThreadState
ThreadState { public hidebysig specialname instance
valuetype System.Threading.ThreadState get_ThreadState() }

[C#]
public ThreadState ThreadState { get; }
```

Summary

Gets a value containing the states of the current thread.

Property Value

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

Description

This property is read-only.

A thread is running if the value returned by this property does not include **System.Threading.ThreadState.Unstarted** and does not include **System.Threading.ThreadState.Stopped**.