Mutexes - .NET

By BillWagner

Archived: 2026-04-05 23:11:32 UTC

You can use a Mutex object to provide exclusive access to a resource. The Mutex class uses more system resources than the Monitor class, but it can be marshalled across application domain boundaries, it can be used with multiple waits, and it can be used to synchronize threads in different processes. For a comparison of managed synchronization mechanisms, see Overview of Synchronization Primitives.

For code examples, see the reference documentation for the Mutex constructors.

A thread calls the WaitOne method of a mutex to request ownership. The call blocks until the mutex is available, or until the optional timeout interval elapses. The state of a mutex is signaled if no thread owns it.

A thread releases a mutex by calling its ReleaseMutex method. Mutexes have thread affinity; that is, the mutex can be released only by the thread that owns it. If a thread releases a mutex it does not own, an ApplicationException is thrown in the thread.

Because the Mutex class derives from WaitHandle, you can also call the static WaitAll or WaitAny methods of WaitHandle to request ownership of a Mutex in combination with other wait handles.

If a thread owns a Mutex, that thread can specify the same Mutex in repeated wait-request calls without blocking its execution; however, it must release the Mutex as many times to release ownership.

If a thread terminates without releasing a Mutex, the mutex is said to be abandoned. This often indicates a serious programming error because the resource the mutex is protecting might be left in an inconsistent state. An AbandonedMutexException is thrown in the next thread that acquires the mutex.

In the case of a system-wide mutex, an abandoned mutex might indicate that an application has been terminated abruptly (for example, by using Windows Task Manager).

Mutexes are of two types: local mutexes and named system mutexes. If you create a Mutex object using a constructor that accepts a name, it is associated with an operating-system object of that name. Named system mutexes are visible throughout the operating system and can be used to synchronize the activities of processes. You can create multiple Mutex objects that represent the same named system mutex, and you can use the OpenExisting method to open an existing named system mutex.

A local mutex exists only within your process. It can be used by any thread in your process that has a reference to the local Mutex object. Each Mutex object is a separate local mutex.

.NET Framework provides the ability to query and set Windows access control security for named system objects. Protecting system mutexes from the moment of creation is recommended because system objects are global and therefore can be locked by code other than your own.

For information on access control security for mutexes, see the MutexSecurity and MutexAccessRule classes, the MutexRights enumeration, the GetAccessControl, SetAccessControl, and OpenExisting methods of the Mutex class, and the Mutex(Boolean, String, Boolean, MutexSecurity) constructor.

Note

Access control security for system mutexes is only available with .NET Framework, it's not available with .NET Core or .NET 5+.

- System.Threading.Mutex
- Mutex
- System.Security.AccessControl.MutexSecurity
- System.Security.AccessControl.MutexAccessRule
- System.Threading.Monitor
- Threading objects and features
- Threads and threading
- Threading
