



OpenSceneGraph Version 2.9.6

# **OpenThreads::**

## **Reference Manual**



# Contents

---

<b>1</b>	<b>Main Page</b>	<b>1</b>
<b>2</b>	<b>Directory Documentation</b>	<b>3</b>
2.1	src/OpenThreads/common/ Directory Reference	3
2.2	include/ Directory Reference	4
2.3	src/OpenThreads/ Directory Reference	5
2.4	include/OpenThreads/ Directory Reference	6
2.5	src/OpenThreads/pthreads/ Directory Reference	7
2.6	src/OpenThreads/sproc/ Directory Reference	8
2.7	src/ Directory Reference	9
2.8	src/OpenThreads/win32/ Directory Reference	10
<b>3</b>	<b>Namespace Documentation</b>	<b>11</b>
3.1	OpenThreads Namespace Reference	11
3.1.1	Function Documentation	12
3.1.1.1	cooperativeWait	12
3.1.1.2	GetNumberOfProcessors	12
3.1.1.3	SetProcessorAffinityOfCurrentThread	12
3.2	Producer Namespace Reference	13
<b>4</b>	<b>Class Documentation</b>	<b>15</b>
4.1	Atomic Class Reference	15
4.1.1	Detailed Description	15
4.1.2	Constructor & Destructor Documentation	15
4.1.2.1	Atomic	15
4.1.3	Member Function Documentation	15
4.1.3.1	AND	15
4.1.3.2	exchange	15
4.1.3.3	operator unsigned	15
4.1.3.4	operator++	15
4.1.3.5	operator--	15
4.1.3.6	OR	15
4.1.3.7	XOR	15
4.2	AtomicPtr Class Reference	16
4.2.1	Detailed Description	16
4.2.2	Constructor & Destructor Documentation	16
4.2.2.1	AtomicPtr	16
4.2.2.2	~AtomicPtr	16

4.2.3	Member Function Documentation	16
4.2.3.1	assign	16
4.2.3.2	get	16
4.3	Barrier Class Reference	17
4.3.1	Detailed Description	17
4.3.2	Constructor & Destructor Documentation	17
4.3.2.1	Barrier	17
4.3.2.2	~Barrier	17
4.3.3	Member Function Documentation	17
4.3.3.1	block	17
4.3.3.2	invalidate	17
4.3.3.3	numThreadsCurrentlyBlocked	17
4.3.3.4	release	17
4.3.3.5	reset	17
4.4	Block Class Reference	18
4.4.1	Detailed Description	18
4.4.2	Constructor & Destructor Documentation	18
4.4.2.1	Block	18
4.4.2.2	~Block	18
4.4.3	Member Function Documentation	18
4.4.3.1	block	18
4.4.3.2	block	18
4.4.3.3	release	18
4.4.3.4	reset	18
4.4.3.5	set	18
4.4.4	Member Data Documentation	18
4.4.4.1	_cond	18
4.4.4.2	_mut	18
4.4.4.3	_released	18
4.5	BlockCount Class Reference	20
4.5.1	Detailed Description	20
4.5.2	Constructor & Destructor Documentation	21
4.5.2.1	BlockCount	21
4.5.2.2	~BlockCount	21
4.5.3	Member Function Documentation	21
4.5.3.1	block	21
4.5.3.2	completed	21
4.5.3.3	getBlockCount	21
4.5.3.4	getCurrentCount	21
4.5.3.5	release	21
4.5.3.6	reset	21

4.5.3.7	setBlockCount	21
4.5.4	Member Data Documentation	21
4.5.4.1	_blockCount	21
4.5.4.2	_cond	21
4.5.4.3	_currentCount	21
4.5.4.4	_mut	21
4.6	Condition Class Reference	22
4.6.1	Constructor & Destructor Documentation	22
4.6.1.1	Condition	22
4.6.1.2	~Condition	22
4.6.2	Member Function Documentation	22
4.6.2.1	broadcast	22
4.6.2.2	wait	22
4.6.3	Member Data Documentation	22
4.6.3.1	_cond	22
4.7	Condition Class Reference	23
4.7.1	Detailed Description	23
4.7.2	Constructor & Destructor Documentation	23
4.7.2.1	Condition	23
4.7.2.2	~Condition	23
4.7.3	Member Function Documentation	23
4.7.3.1	broadcast	23
4.7.3.2	signal	23
4.7.3.3	wait	23
4.7.3.4	wait	23
4.8	HandleHolder Class Reference	24
4.8.1	Constructor & Destructor Documentation	24
4.8.1.1	HandleHolder	24
4.8.1.2	HandleHolder	24
4.8.1.3	~HandleHolder	24
4.8.2	Member Function Documentation	24
4.8.2.1	get	24
4.8.2.2	operator bool	24
4.8.2.3	set	24
4.9	Mutex Class Reference	25
4.9.1	Detailed Description	25
4.9.2	Constructor & Destructor Documentation	25
4.9.2.1	Mutex	25
4.9.2.2	~Mutex	25
4.9.3	Member Function Documentation	25
4.9.3.1	lock	25

4.9.3.2	trylock	26
4.9.3.3	unlock	26
4.9.4	Friends And Related Function Documentation	26
4.9.4.1	Condition	26
4.10	PThreadBarrierPrivateData Class Reference	27
4.10.1	Friends And Related Function Documentation	27
4.10.1.1	Barrier	27
4.11	PThreadConditionPrivateData Class Reference	28
4.11.1	Friends And Related Function Documentation	28
4.11.1.1	Condition	28
4.12	PThreadMutexPrivateData Class Reference	29
4.12.1	Friends And Related Function Documentation	29
4.12.1.1	Condition	29
4.12.1.2	Mutex	29
4.13	PThreadPrivateData Class Reference	30
4.13.1	Friends And Related Function Documentation	30
4.13.1.1	Thread	30
4.13.1.2	ThreadPrivateActions	30
4.14	ReadWriteMutex Class Reference	31
4.14.1	Constructor & Destructor Documentation	31
4.14.1.1	ReadWriteMutex	31
4.14.1.2	~ReadWriteMutex	31
4.14.1.3	ReadWriteMutex	31
4.14.2	Member Function Documentation	31
4.14.2.1	operator=	31
4.14.2.2	readLock	31
4.14.2.3	readUnlock	31
4.14.2.4	writeLock	31
4.14.2.5	writeUnlock	31
4.14.3	Member Data Documentation	31
4.14.3.1	_readCount	31
4.14.3.2	_readCountMutex	31
4.14.3.3	_readWriteMutex	31
4.15	ReentrantMutex Class Reference	33
4.15.1	Constructor & Destructor Documentation	33
4.15.1.1	ReentrantMutex	33
4.15.1.2	~ReentrantMutex	33
4.15.2	Member Function Documentation	33
4.15.2.1	lock	33
4.15.2.2	trylock	33
4.15.2.3	unlock	34

4.16	ReverseScopedLock< M > Class Template Reference	35
4.16.1	Constructor & Destructor Documentation	35
4.16.1.1	ReverseScopedLock	35
4.16.1.2	~ReverseScopedLock	35
4.17	ReverseScopedPointerLock< M > Class Template Reference	36
4.17.1	Constructor & Destructor Documentation	36
4.17.1.1	ReverseScopedPointerLock	36
4.17.1.2	~ReverseScopedPointerLock	36
4.18	ScopedLock< M > Class Template Reference	37
4.18.1	Constructor & Destructor Documentation	37
4.18.1.1	ScopedLock	37
4.18.1.2	~ScopedLock	37
4.19	ScopedPointerLock< M > Class Template Reference	38
4.19.1	Constructor & Destructor Documentation	38
4.19.1.1	ScopedPointerLock	38
4.19.1.2	~ScopedPointerLock	38
4.20	ScopedReadLock Class Reference	39
4.20.1	Constructor & Destructor Documentation	39
4.20.1.1	ScopedReadLock	39
4.20.1.2	~ScopedReadLock	39
4.20.2	Member Function Documentation	39
4.20.2.1	operator=	39
4.20.3	Member Data Documentation	39
4.20.3.1	_mutex	39
4.21	ScopedWriteLock Class Reference	40
4.21.1	Constructor & Destructor Documentation	40
4.21.1.1	ScopedWriteLock	40
4.21.1.2	~ScopedWriteLock	40
4.21.2	Member Function Documentation	40
4.21.2.1	operator=	40
4.21.3	Member Data Documentation	40
4.21.3.1	_mutex	40
4.22	SemaLink Class Reference	41
4.22.1	Friends And Related Function Documentation	41
4.22.1.1	Condition	41
4.22.1.2	ConditionDebug	41
4.22.1.3	SprocConditionPrivatedata	41
4.23	SharedArena Class Reference	42
4.23.1	Friends And Related Function Documentation	42
4.23.1.1	Barrier	42
4.23.1.2	Condition	42

4.23.1.3	Mutex	42
4.24	SprocBarrierPrivateData Class Reference	43
4.24.1	Friends And Related Function Documentation	43
4.24.1.1	Barrier	43
4.25	SprocConditionPrivateData Class Reference	44
4.25.1	Friends And Related Function Documentation	44
4.25.1.1	Condition	44
4.26	SprocMutexPrivateData Class Reference	45
4.26.1	Friends And Related Function Documentation	45
4.26.1.1	Mutex	45
4.26.1.2	SprocThreadPrivateActions	45
4.27	SprocThreadPrivateData Class Reference	46
4.27.1	Friends And Related Function Documentation	46
4.27.1.1	Thread	46
4.27.1.2	ThreadPrivateActions	46
4.28	Thread Class Reference	47
4.28.1	Detailed Description	49
4.28.2	Member Enumeration Documentation	49
4.28.2.1	ThreadPolicy	49
4.28.2.2	ThreadPriority	49
4.28.3	Constructor & Destructor Documentation	49
4.28.3.1	Thread	49
4.28.3.2	~Thread	49
4.28.4	Member Function Documentation	49
4.28.4.1	cancel	49
4.28.4.2	cancelCleanup	49
4.28.4.3	CurrentThread	49
4.28.4.4	detach	50
4.28.4.5	GetConcurrency	50
4.28.4.6	getImplementation	50
4.28.4.7	GetMasterPriority	50
4.28.4.8	getProcessId	50
4.28.4.9	getSchedulePolicy	50
4.28.4.10	getSchedulePriority	50
4.28.4.11	getStackSize	50
4.28.4.12	getThreadId	50
4.28.4.13	Init	50
4.28.4.14	isRunning	50
4.28.4.15	join	50
4.28.4.16	microSleep	50
4.28.4.17	printSchedulingInfo	50

4.28.4.18	run	51
4.28.4.19	setCancelModeAsynchronous	51
4.28.4.20	setCancelModeDeferred	51
4.28.4.21	setCancelModeDisable	51
4.28.4.22	SetConcurrency	51
4.28.4.23	setProcessorAffinity	51
4.28.4.24	setSchedulePolicy	51
4.28.4.25	setSchedulePriority	51
4.28.4.26	setStackSize	51
4.28.4.27	start	52
4.28.4.28	startThread	52
4.28.4.29	testCancel	52
4.28.4.30	YieldCurrentThread	52
4.28.5	Friends And Related Function Documentation	52
4.28.5.1	ThreadPrivateActions	52
4.29	ThreadPrivateActions Class Reference	53
4.29.1	Member Function Documentation	53
4.29.1.1	PopCancelFunction	53
4.29.1.2	PushCancelFunction	53
4.29.1.3	ThreadCancelTest	53
4.29.2	Friends And Related Function Documentation	53
4.29.2.1	Thread	53
4.30	TlsHolder Struct Reference	54
4.30.1	Constructor & Destructor Documentation	54
4.30.1.1	TlsHolder	54
4.30.1.2	~TlsHolder	54
4.30.2	Member Function Documentation	54
4.30.2.1	getld	54
4.31	Win32BarrierPrivateData Class Reference	55
4.31.1	Friends And Related Function Documentation	55
4.31.1.1	Barrier	55
4.32	Win32ConditionPrivateData Class Reference	56
4.32.1	Constructor & Destructor Documentation	57
4.32.1.1	Win32ConditionPrivateData	57
4.32.1.2	~Win32ConditionPrivateData	57
4.32.2	Member Function Documentation	57
4.32.2.1	broadcast	57
4.32.2.2	signal	57
4.32.2.3	wait	57
4.32.3	Friends And Related Function Documentation	57
4.32.3.1	Condition	57

4.32.4	Member Data Documentation	57
4.32.4.1	sema_	57
4.32.4.2	waiters_	57
4.32.4.3	waiters_done_	57
4.32.4.4	was_broadcast_	57
4.33	Win32MutexPrivateData Class Reference	58
4.33.1	Friends And Related Function Documentation	58
4.33.1.1	Condition	58
4.33.1.2	Mutex	58
4.34	Win32ThreadCanceled Struct Reference	59
4.35	Win32ThreadPrivateData Class Reference	60
4.35.1	Friends And Related Function Documentation	60
4.35.1.1	Thread	60
4.35.1.2	ThreadPrivateActions	60
4.35.2	Member Data Documentation	60
4.35.2.1	cancelEvent	60
4.35.2.2	TLS	60
<b>5</b>	<b>File Documentation</b>	<b>61</b>
5.1	Atomic File Reference	61
5.1.1	Define Documentation	61
5.1.1.1	_OPENTHREADS_ATOMIC_INLINE	61
5.2	Atomic.cpp File Reference	62
5.3	Barrier File Reference	63
5.4	Block File Reference	64
5.5	Condition File Reference	65
5.6	Exports File Reference	66
5.6.1	Define Documentation	66
5.6.1.1	OPENTHREAD_EXPORT_DIRECTIVE	66
5.7	HandleHolder.h File Reference	67
5.7.1	Define Documentation	67
5.7.1.1	WIN32_LEAN_AND_MEAN	67
5.8	mainpage.h File Reference	68
5.8.1	Detailed Description	68
5.9	Mutex File Reference	69
5.10	PThreadBarrierPrivateData.h File Reference	70
5.11	PThreadConditionPrivateData.h File Reference	71
5.12	PThreadMutexPrivateData.h File Reference	72
5.13	PThreadPrivateData.h File Reference	73
5.14	ReadWriteMutex File Reference	74
5.15	ReentrantMutex File Reference	75

5.16	ScopedLock File Reference	76
5.17	SharedArena.h File Reference	77
5.17.1	Define Documentation	77
5.17.1.1	OT_USESHAREDONLY	77
5.18	SprocBarrierPrivateData.h File Reference	78
5.19	SprocConditionPrivateData.h File Reference	79
5.20	SprocMutexPrivateData.h File Reference	80
5.21	SprocThreadPrivateActions.h File Reference	81
5.22	SprocThreadPrivateData.h File Reference	82
5.23	Thread File Reference	83
5.24	Version File Reference	84
5.24.1	Define Documentation	84
5.24.1.1	OPENTHREADS_MAJOR_VERSION	84
5.24.1.2	OPENTHREADS_MINOR_VERSION	84
5.24.1.3	OPENTHREADS_PATCH_VERSION	84
5.24.1.4	OPENTHREADS_SOVERSION	84
5.24.1.5	OPENTHREADS_VERSION	84
5.24.2	Function Documentation	84
5.24.2.1	OpenThreadsGetLibraryName	84
5.24.2.2	OpenThreadsGetSOVersion	85
5.24.2.3	OpenThreadsGetVersion	85
5.25	Version.cpp File Reference	86
5.25.1	Function Documentation	86
5.25.1.1	OpenThreadsGetLibraryName	86
5.25.1.2	OpenThreadsGetSOVersion	86
5.25.1.3	OpenThreadsGetVersion	86
5.26	Win32BarrierPrivateData.h File Reference	87
5.26.1	Define Documentation	87
5.26.1.1	_WIN32_WINNT	87
5.26.1.2	WIN32_LEAN_AND_MEAN	87
5.27	WIN32Condition.cpp File Reference	88
5.28	Win32Condition.h File Reference	89
5.28.1	Define Documentation	89
5.28.1.1	PRODUCER_CONDITION	89
5.29	Win32ConditionPrivateData.h File Reference	90
5.29.1	Define Documentation	90
5.29.1.1	_WIN32_WINNT	90
5.29.1.2	InterlockedGet	90
5.29.1.3	WIN32_LEAN_AND_MEAN	90
5.30	Win32Mutex.cpp File Reference	91
5.31	Win32MutexPrivateData.h File Reference	92

---

5.31.1	Define Documentation . . . . .	92
5.31.1.1	_WIN32_WINNT . . . . .	92
5.31.1.2	USE_CRITICAL_SECTION . . . . .	92
5.31.1.3	WIN32_LEAN_AND_MEAN . . . . .	92
5.32	Win32Thread.cpp File Reference . . . . .	93
5.33	Win32ThreadBarrier.cpp File Reference . . . . .	94
5.34	Win32ThreadPrivateData.h File Reference . . . . .	95
5.34.1	Define Documentation . . . . .	95
5.34.1.1	_WIN32_WINNT . . . . .	95
5.34.1.2	WIN32_LEAN_AND_MEAN . . . . .	95

## Main Page

---

### OpenThreads

The **OpenThreads** library is intended to provide a minimal & complete Object-Oriented (OO) thread interface for C++ programmers. It is loosely modeled on the Java thread API, and the POSIX Threads standards. The architecture of the library is designed around "swappable" thread models which are defined at compile-time in a shared object library.

**Note:** The **OpenThreads** library comes shipped with the OSG installation files.

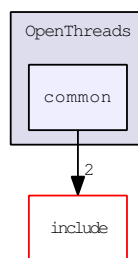
The documentation you are looking at can be downloaded from [www.3draum.ch](http://www.3draum.ch).



## Directory Documentation

---

### 2.1 src/OpenThreads/common/ Directory Reference



#### Files

- file [Atomic.cpp](#)
- file [Version.cpp](#)

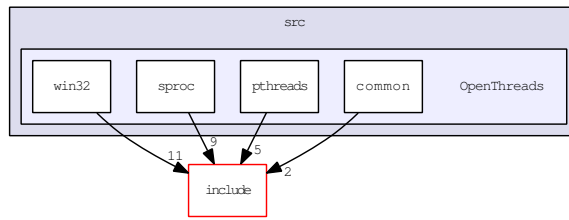
## 2.2 include/ Directory Reference



### Directories

- directory [OpenThreads](#)

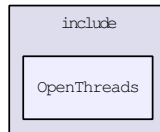
## 2.3 src/OpenThreads/ Directory Reference



### Directories

- directory [common](#)
- directory [pthreads](#)
- directory [sproc](#)
- directory [win32](#)

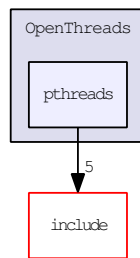
## 2.4 include/OpenThreads/ Directory Reference



### Files

- file [Atomic](#)
- file [Barrier](#)
- file [Block](#)
- file [Condition](#)
- file [Exports](#)
- file [mainpage.h](#)
- file [Mutex](#)
- file [ReadWriteMutex](#)
- file [ReentrantMutex](#)
- file [ScopedLock](#)
- file [Thread](#)
- file [Version](#)

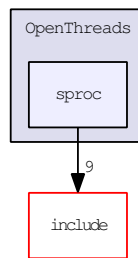
## 2.5 src/OpenThreads/pthreads/ Directory Reference



### Files

- file [PThreadBarrierPrivateData.h](#)
- file [PThreadConditionPrivateData.h](#)
- file [PThreadMutexPrivateData.h](#)
- file [PThreadPrivateData.h](#)

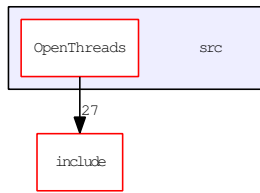
## 2.6 src/OpenThreads/sproc/ Directory Reference



### Files

- file [SharedArena.h](#)
- file [SprocBarrierPrivateData.h](#)
- file [SprocConditionPrivateData.h](#)
- file [SprocMutexPrivateData.h](#)
- file [SprocThreadPrivateActions.h](#)
- file [SprocThreadPrivateData.h](#)

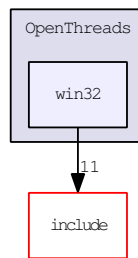
## 2.7 src/ Directory Reference



### Directories

- directory [OpenThreads](#)

## 2.8 src/OpenThreads/win32/ Directory Reference



### Files

- file [HandleHolder.h](#)
- file [Win32BarrierPrivateData.h](#)
- file [WIN32Condition.cpp](#)
- file [Win32Condition.h](#)
- file [Win32ConditionPrivateData.h](#)
- file [Win32Mutex.cpp](#)
- file [Win32MutexPrivateData.h](#)
- file [Win32Thread.cpp](#)
- file [Win32ThreadBarrier.cpp](#)
- file [Win32ThreadPrivateData.h](#)

## Namespace Documentation

---

### 3.1 OpenThreads Namespace Reference

#### Classes

- class [Atomic](#)  
*This class provides an atomic increment and decrement operation.*
- class [AtomicPtr](#)  
*This class provides an atomic pointer assignment using cas operations.*
- class [Barrier](#)  
*This class provides an object-oriented thread barrier interface.*
- class [Block](#)  
*Block is a block that can be used to halt a thread that is waiting another thread to release it.*
- class [BlockCount](#)  
*BlockCount is a block that can be used to halt a thread that is waiting for a specified number of operations to be completed.*
- class [Condition](#)  
*This class provides an object-oriented thread condition interface.*
- class [HandleHolder](#)
- class [Mutex](#)  
*This class provides an object-oriented thread mutex interface.*
- class [PThreadBarrierPrivateData](#)
- class [PThreadConditionPrivateData](#)
- class [PThreadMutexPrivateData](#)
- class [PThreadPrivateData](#)
- class [ReadWriteMutex](#)
- class [ReentrantMutex](#)
- class [ReverseScopedLock](#)
- class [ReverseScopedPointerLock](#)
- class [ScopedLock](#)
- class [ScopedPointerLock](#)
- class [ScopedReadLock](#)
- class [ScopedWriteLock](#)
- class [SemaLink](#)
- class [SharedArena](#)
- class [SprocBarrierPrivateData](#)
- class [SprocConditionPrivateData](#)

- class [SprocMutexPrivateData](#)
- class [SprocThreadPrivateData](#)
- class [Thread](#)

*This class provides an object-oriented thread interface.*

- class [ThreadPrivateActions](#)
- class [Win32BarrierPrivateData](#)
- class [Win32ConditionPrivateData](#)
- class [Win32MutexPrivateData](#)
- class [Win32ThreadPrivateData](#)

## Functions

- DWORD [cooperativeWait](#) (HANDLE *waitHandle*, unsigned long *timeout*)
- OPENTHREAD\_EXPORT\_DIRECTIVE int [GetNumberOfProcessors](#) ()  
*Get the number of processors.*
- OPENTHREAD\_EXPORT\_DIRECTIVE int [SetProcessorAffinityOfCurrentThread](#) (unsigned int *cpunum*)  
*Set the processor affinity of current thread.*

### 3.1.1 Function Documentation

#### 3.1.1.1 DWORD [cooperativeWait](#) (HANDLE *waitHandle*, unsigned long *timeout*)

#### 3.1.1.2 int [GetNumberOfProcessors](#) ()

Get the number of processors. Note, systems where no support exists for querying the number of processors, 1 is returned.

#### 3.1.1.3 int [SetProcessorAffinityOfCurrentThread](#) (unsigned int *cpunum*)

Set the processor affinity of current thread. Note, systems where no support exists no affinity will be set, and -1 will be returned.

## 3.2 Producer Namespace Reference

### Classes

- class [Condition](#)



## Class Documentation

---

### 4.1 Atomic Class Reference

This class provides an atomic increment and decrement operation.

#### Public Member Functions

- [Atomic](#) (unsigned value=0)
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [AND](#) (unsigned value)
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [exchange](#) (unsigned value=0)
- `_OPENTHREADS_ATOMIC_INLINE` [operator unsigned](#) () const
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [operator++](#) ()
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [operator--](#) ()
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [OR](#) (unsigned value)
- `_OPENTHREADS_ATOMIC_INLINE` unsigned [XOR](#) (unsigned value)

#### 4.1.1 Detailed Description

This class provides an atomic increment and decrement operation.

#### 4.1.2 Constructor & Destructor Documentation

4.1.2.1 [Atomic](#) (unsigned *value* = 0) [`inline`]

#### 4.1.3 Member Function Documentation

4.1.3.1 `_OPENTHREADS_ATOMIC_INLINE` unsigned [AND](#) (unsigned *value*)

4.1.3.2 `_OPENTHREADS_ATOMIC_INLINE` unsigned [exchange](#) (unsigned *value* = 0)

4.1.3.3 `_OPENTHREADS_ATOMIC_INLINE` [operator unsigned](#) () const

4.1.3.4 `_OPENTHREADS_ATOMIC_INLINE` unsigned [operator++](#) ()

4.1.3.5 `_OPENTHREADS_ATOMIC_INLINE` unsigned [operator--](#) ()

4.1.3.6 `_OPENTHREADS_ATOMIC_INLINE` unsigned [OR](#) (unsigned *value*)

4.1.3.7 `_OPENTHREADS_ATOMIC_INLINE` unsigned [XOR](#) (unsigned *value*)

The documentation for this class was generated from the following file:

- [Atomic](#)

## 4.2 AtomicPtr Class Reference

This class provides an atomic pointer assignment using cas operations.

### Public Member Functions

- [AtomicPtr](#) (void \*ptr=0)
- [~AtomicPtr](#) ()
- `_OPENTHREADS_ATOMIC_INLINE` bool [assign](#) (void \*ptrNew, const void \*const ptrOld)
- `_OPENTHREADS_ATOMIC_INLINE` void \* [get](#) () const

### 4.2.1 Detailed Description

This class provides an atomic pointer assignment using cas operations.

### 4.2.2 Constructor & Destructor Documentation

**4.2.2.1** `AtomicPtr (void * ptr = 0) [inline]`

**4.2.2.2** `~AtomicPtr () [inline]`

### 4.2.3 Member Function Documentation

**4.2.3.1** `_OPENTHREADS_ATOMIC_INLINE` bool [assign](#) (void \* *ptrNew*, const void \*const *ptrOld*)

**4.2.3.2** `_OPENTHREADS_ATOMIC_INLINE` void \* [get](#) () const

The documentation for this class was generated from the following file:

- [Atomic](#)

## 4.3 Barrier Class Reference

This class provides an object-oriented thread barrier interface.

### Public Member Functions

- [Barrier](#) (int numThreads=0)  
*Constructor.*
- virtual [~Barrier](#) ()  
*Destructor.*
- virtual void [block](#) (unsigned int numThreads=0)  
*Block until numThreads threads have entered the barrier.*
- void [invalidate](#) ()
- virtual int [numThreadsCurrentlyBlocked](#) ()  
*Return the number of threads currently blocked in the barrier, Return -1 if error.*
- virtual void [release](#) ()  
*Release the barrier, now.*
- virtual void [reset](#) ()  
*Reset the barrier to it's original state.*

### 4.3.1 Detailed Description

This class provides an object-oriented thread barrier interface. Warning It is unwise to use the construct "Barrier barrier" in the global namespace on sgi's. The object "barrier" will conflict with the c-library sproc function "barrier" and unpredictable results may occur. You have been warned.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 [Barrier](#) (int *numThreads* = 0)

Constructor.

#### 4.3.2.2 [~Barrier](#) () [virtual]

Destructor.

### 4.3.3 Member Function Documentation

#### 4.3.3.1 void [block](#) (unsigned int *numThreads* = 0) [virtual]

[Block](#) until numThreads threads have entered the barrier.

#### 4.3.3.2 void [invalidate](#) ()

#### 4.3.3.3 int [numThreadsCurrentlyBlocked](#) () [virtual]

Return the number of threads currently blocked in the barrier, Return -1 if error.

#### 4.3.3.4 void [release](#) () [virtual]

Release the barrier, now.

#### 4.3.3.5 void [reset](#) () [virtual]

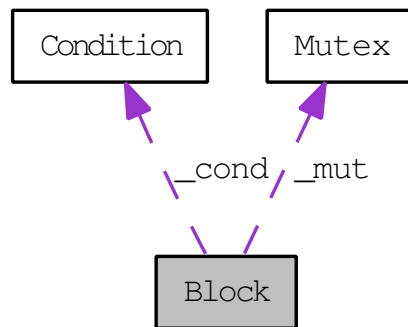
Reset the barrier to it's original state.

The documentation for this class was generated from the following files:

- [Barrier](#)
- [Win32ThreadBarrier.cpp](#)

## 4.4 Block Class Reference

[Block](#) is a block that can be used to halt a thread that is waiting another thread to release it. Collaboration diagram for Block:



### Public Member Functions

- [Block](#) ()
- [~Block](#) ()
- bool [block](#) (unsigned long timeout)
- bool [block](#) ()
- void [release](#) ()
- void [reset](#) ()
- void [set](#) (bool doRelease)

### Protected Attributes

- [Condition \\_cond](#)
- [Mutex \\_mut](#)
- bool [\\_released](#)

#### 4.4.1 Detailed Description

[Block](#) is a block that can be used to halt a thread that is waiting another thread to release it.

#### 4.4.2 Constructor & Destructor Documentation

4.4.2.1 [Block](#) () [inline]

4.4.2.2 [~Block](#) () [inline]

#### 4.4.3 Member Function Documentation

4.4.3.1 bool [block](#) (unsigned long *timeout*) [inline]

4.4.3.2 bool [block](#) () [inline]

4.4.3.3 void [release](#) () [inline]

4.4.3.4 void [reset](#) () [inline]

4.4.3.5 void [set](#) (bool *doRelease*) [inline]

#### 4.4.4 Member Data Documentation

4.4.4.1 [Condition \\_cond](#) [protected]

4.4.4.2 [Mutex \\_mut](#) [protected]

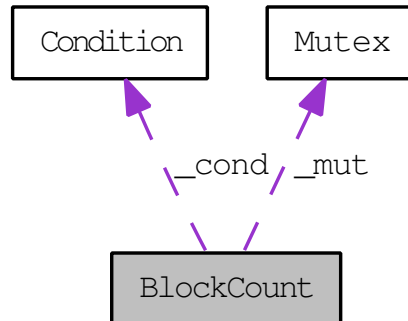
4.4.4.3 [bool \\_released](#) [protected]

The documentation for this class was generated from the following file:

- [Block](#)

## 4.5 BlockCount Class Reference

[BlockCount](#) is a block that can be used to halt a thread that is waiting for a specified number of operations to be completed. Collaboration diagram for BlockCount:



### Public Member Functions

- [BlockCount](#) (unsigned int blockCount)
- [~BlockCount](#) ()
- void [block](#) ()
- void [completed](#) ()
- unsigned int [getBlockCount](#) () const
- unsigned int [getCurrentCount](#) () const
- void [release](#) ()
- void [reset](#) ()
- void [setBlockCount](#) (unsigned int blockCount)

### Protected Attributes

- unsigned int [\\_blockCount](#)
- [OpenThreads::Condition](#) [\\_cond](#)
- unsigned int [\\_currentCount](#)
- [OpenThreads::Mutex](#) [\\_mut](#)

#### 4.5.1 Detailed Description

[BlockCount](#) is a block that can be used to halt a thread that is waiting for a specified number of operations to be completed.

## 4.5.2 Constructor & Destructor Documentation

4.5.2.1 `BlockCount (unsigned int blockCount)` [`inline`]

4.5.2.2 `~BlockCount ()` [`inline`]

## 4.5.3 Member Function Documentation

4.5.3.1 `void block ()` [`inline`]

4.5.3.2 `void completed ()` [`inline`]

4.5.3.3 `unsigned int getBlockCount () const` [`inline`]

4.5.3.4 `unsigned int getCurrentCount () const` [`inline`]

4.5.3.5 `void release ()` [`inline`]

4.5.3.6 `void reset ()` [`inline`]

4.5.3.7 `void setBlockCount (unsigned int blockCount)` [`inline`]

## 4.5.4 Member Data Documentation

4.5.4.1 `unsigned int _blockCount` [`protected`]

4.5.4.2 `OpenThreads::Condition _cond` [`protected`]

4.5.4.3 `unsigned int _currentCount` [`protected`]

4.5.4.4 `OpenThreads::Mutex _mut` [`protected`]

The documentation for this class was generated from the following file:

- [Block](#)

## 4.6 Condition Class Reference

```
#include <OpenThreads/win32/Win32Condition.h>
```

### Public Member Functions

- [Condition](#) (long *max*)  
*number of waiters.*
- [~Condition](#) ()
- int [broadcast](#) ()
- int [wait](#) (Mutex &*external\_mutex*)

### Protected Attributes

- pthread\_cond\_t [\\_cond](#)

#### 4.6.1 Constructor & Destructor Documentation

##### 4.6.1.1 [Condition](#) (long *max*) [[inline](#)]

*number of waiters.*

##### 4.6.1.2 [~Condition](#) () [[inline](#)]

#### 4.6.2 Member Function Documentation

##### 4.6.2.1 int [broadcast](#) () [[inline](#)]

##### 4.6.2.2 int [wait](#) (Mutex & *external\_mutex*) [[inline](#)]

#### 4.6.3 Member Data Documentation

##### 4.6.3.1 pthread\_cond\_t [\\_cond](#) [[protected](#)]

The documentation for this class was generated from the following file:

- [Win32Condition.h](#)

## 4.7 Condition Class Reference

This class provides an object-oriented thread condition interface.

### Public Member Functions

- [Condition \(\)](#)  
*Constructor.*
- virtual [~Condition \(\)](#)  
*Destructor.*
- virtual int [broadcast \(\)](#)  
*Wake all threads waiting on this condition.*
- virtual int [signal \(\)](#)  
*Signal a SINGLE thread to wake if it's waiting.*
- virtual int [wait \(Mutex \\*mutex, unsigned long int ms\)](#)  
*Wait on a mutex for a given amount of time (ms).*
- virtual int [wait \(Mutex \\*mutex\)](#)  
*Wait on a mutex.*

### 4.7.1 Detailed Description

This class provides an object-oriented thread condition interface.

### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Condition ()

Constructor.

#### 4.7.2.2 ~Condition () [virtual]

Destructor.

### 4.7.3 Member Function Documentation

#### 4.7.3.1 int broadcast () [virtual]

Wake all threads waiting on this condition. Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.7.3.2 int signal () [virtual]

Signal a SINGLE thread to wake if it's waiting. Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.7.3.3 virtual int wait (Mutex \* *mutex*, unsigned long int *ms*) [virtual]

Wait on a mutex for a given amount of time (ms). Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.7.3.4 int wait (Mutex \* *mutex*) [virtual]

Wait on a mutex.

The documentation for this class was generated from the following files:

- [Condition](#)
- [WIN32Condition.cpp](#)

## 4.8 HandleHolder Class Reference

```
#include <OpenThreads/win32/HandleHolder.h>
```

### Public Member Functions

- [HandleHolder](#) (HANDLE h)
- [HandleHolder](#) ()
- [~HandleHolder](#) ()
- const HANDLE & [get](#) () const
- [operator bool](#) ()
- void [set](#) (HANDLE h)

### 4.8.1 Constructor & Destructor Documentation

4.8.1.1 [HandleHolder](#) () [inline]

4.8.1.2 [HandleHolder](#) (HANDLE h) [inline, explicit]

4.8.1.3 [~HandleHolder](#) () [inline]

### 4.8.2 Member Function Documentation

4.8.2.1 const HANDLE& [get](#) () const [inline]

4.8.2.2 [operator bool](#) () [inline]

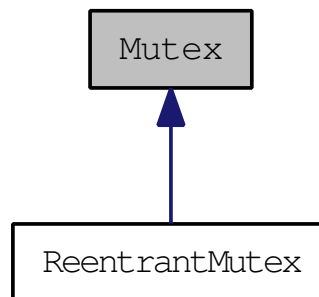
4.8.2.3 void [set](#) (HANDLE h) [inline]

The documentation for this class was generated from the following file:

- [HandleHolder.h](#)

## 4.9 Mutex Class Reference

This class provides an object-oriented thread mutex interface. Inheritance diagram for Mutex:



### Public Member Functions

- [Mutex \(\)](#)  
*Constructor.*
- virtual [~Mutex \(\)](#)  
*Destructor.*
- virtual int [lock \(\)](#)  
*Lock the mutex.*
- virtual int [trylock \(\)](#)  
*Test if mutex can be locked.*
- virtual int [unlock \(\)](#)  
*Unlock the mutex.*

### Friends

- class [Condition](#)

#### 4.9.1 Detailed Description

This class provides an object-oriented thread mutex interface.

#### 4.9.2 Constructor & Destructor Documentation

##### 4.9.2.1 [Mutex \(\)](#)

Constructor.

##### 4.9.2.2 [~Mutex \(\)](#) [virtual]

Destructor.

#### 4.9.3 Member Function Documentation

##### 4.9.3.1 [int lock \(\)](#) [virtual]

Lock the mutex. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented in [ReentrantMutex](#).

**4.9.3.2 int trylock () [virtual]**

Test if mutex can be locked. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented in [ReentrantMutex](#).

**4.9.3.3 int unlock () [virtual]**

Unlock the mutex. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented in [ReentrantMutex](#).

**4.9.4 Friends And Related Function Documentation****4.9.4.1 friend class Condition [friend]**

The documentation for this class was generated from the following files:

- [Mutex](#)
- [Win32Mutex.cpp](#)

## 4.10 PThreadBarrierPrivateData Class Reference

```
#include <OpenThreads/pthreads/PThreadBarrierPrivateData.h>
```

### Friends

- class [Barrier](#)

### 4.10.1 Friends And Related Function Documentation

#### 4.10.1.1 friend class Barrier [friend]

The documentation for this class was generated from the following file:

- [PThreadBarrierPrivateData.h](#)

## 4.11 PThreadConditionPrivateData Class Reference

```
#include <OpenThreads/pthreads/PThreadConditionPrivateData.h>
```

### Friends

- class [Condition](#)

### 4.11.1 Friends And Related Function Documentation

#### 4.11.1.1 friend class Condition [friend]

The documentation for this class was generated from the following file:

- [PThreadConditionPrivateData.h](#)

## 4.12 PThreadMutexPrivateData Class Reference

```
#include <OpenThreads/pthreads/PThreadMutexPrivateData.h>
```

### Friends

- class [Condition](#)
- class [Mutex](#)

### 4.12.1 Friends And Related Function Documentation

**4.12.1.1 friend class Condition [friend]**

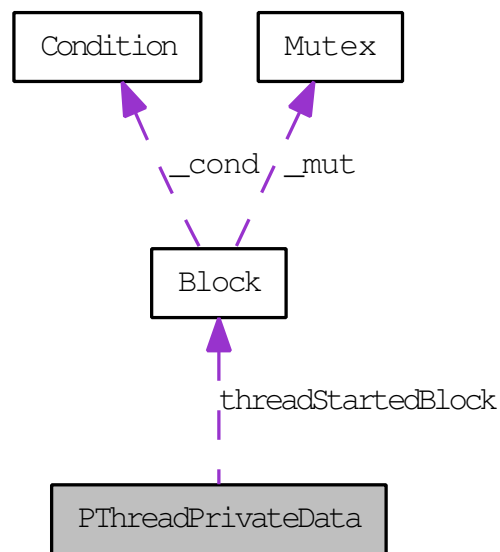
**4.12.1.2 friend class Mutex [friend]**

The documentation for this class was generated from the following file:

- [PThreadMutexPrivateData.h](#)

## 4.13 PThreadPrivateData Class Reference

#include <OpenThreads/pthreads/PThreadPrivateData.h> Collaboration diagram for PThreadPrivateData:



### Friends

- class [Thread](#)
- class [ThreadPrivateActions](#)

### 4.13.1 Friends And Related Function Documentation

4.13.1.1 `friend class Thread` [friend]

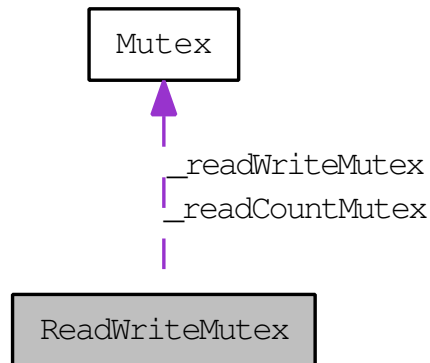
4.13.1.2 `friend class ThreadPrivateActions` [friend]

The documentation for this class was generated from the following file:

- [PThreadPrivateData.h](#)

## 4.14 ReadWriteMutex Class Reference

Collaboration diagram for ReadWriteMutex:



### Public Member Functions

- [ReadWriteMutex \(\)](#)
- virtual [~ReadWriteMutex \(\)](#)
- virtual int [readLock \(\)](#)
- virtual int [readUnlock \(\)](#)
- virtual int [writeLock \(\)](#)
- virtual int [writeUnlock \(\)](#)

### Protected Member Functions

- [ReadWriteMutex \(const ReadWriteMutex &\)](#)
- [ReadWriteMutex & operator= \(const ReadWriteMutex &\)](#)

### Protected Attributes

- unsigned int [\\_readCount](#)
- [OpenThreads::Mutex \\_readCountMutex](#)
- [OpenThreads::Mutex \\_readWriteMutex](#)

### 4.14.1 Constructor & Destructor Documentation

4.14.1.1 [ReadWriteMutex \(\)](#) [inline]

4.14.1.2 [virtual ~ReadWriteMutex \(\)](#) [inline, virtual]

4.14.1.3 [ReadWriteMutex \(const ReadWriteMutex &\)](#) [inline, protected]

### 4.14.2 Member Function Documentation

4.14.2.1 [ReadWriteMutex& operator= \(const ReadWriteMutex &\)](#) [inline, protected]

4.14.2.2 [virtual int readLock \(\)](#) [inline, virtual]

4.14.2.3 [virtual int readUnlock \(\)](#) [inline, virtual]

4.14.2.4 [virtual int writeLock \(\)](#) [inline, virtual]

4.14.2.5 [virtual int writeUnlock \(\)](#) [inline, virtual]

### 4.14.3 Member Data Documentation

4.14.3.1 [unsigned int \\_readCount](#) [protected]

4.14.3.2 [OpenThreads::Mutex \\_readCountMutex](#) [protected]

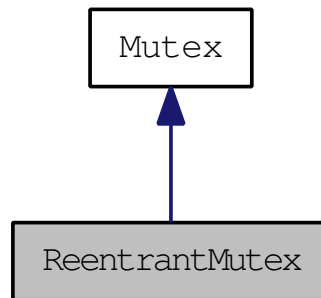
4.14.3.3 [OpenThreads::Mutex \\_readWriteMutex](#) [protected]

The documentation for this class was generated from the following file:

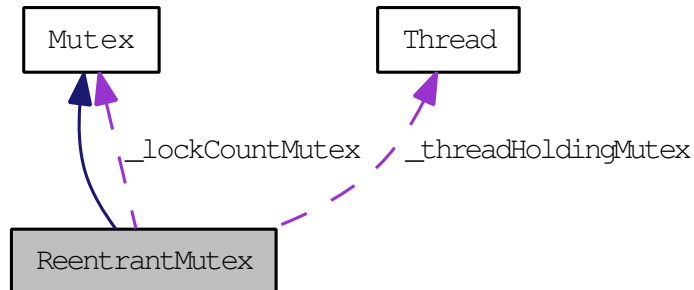
- [ReadWriteMutex](#)

## 4.15 ReentrantMutex Class Reference

Inheritance diagram for ReentrantMutex:



Collaboration diagram for ReentrantMutex:



### Public Member Functions

- [ReentrantMutex \(\)](#)
- virtual [~ReentrantMutex \(\)](#)
- virtual int [lock \(\)](#)  
*Lock the mutex.*
- virtual int [trylock \(\)](#)  
*Test if mutex can be locked.*
- virtual int [unlock \(\)](#)  
*Unlock the mutex.*

### 4.15.1 Constructor & Destructor Documentation

**4.15.1.1** [ReentrantMutex \(\)](#) [`inline`]

**4.15.1.2** [virtual ~ReentrantMutex \(\)](#) [`inline`, `virtual`]

### 4.15.2 Member Function Documentation

**4.15.2.1** [virtual int lock \(\)](#) [`inline`, `virtual`]

Lock the mutex. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented from [Mutex](#).

**4.15.2.2** [virtual int trylock \(\)](#) [`inline`, `virtual`]

Test if mutex can be locked. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented from [Mutex](#).

**4.15.2.3 virtual int unlock () [inline, virtual]**

Unlock the mutex. Returns 0 if normal, -1 if errno set, errno code otherwise.

Reimplemented from [Mutex](#).

The documentation for this class was generated from the following file:

- [ReentrantMutex](#)

## 4.16 ReverseScopedLock< M > Class Template Reference

### Public Member Functions

- [ReverseScopedLock](#) (M &m)
- [~ReverseScopedLock](#) ()

`template<class M> class OpenThreads::ReverseScopedLock< M >`

### 4.16.1 Constructor & Destructor Documentation

4.16.1.1 [ReverseScopedLock](#) (M & *m*) [`inline`, `explicit`]

4.16.1.2 [~ReverseScopedLock](#) () [`inline`]

The documentation for this class was generated from the following file:

- [ScopedLock](#)

## 4.17 ReverseScopedPointerLock< M > Class Template Reference

### Public Member Functions

- [ReverseScopedPointerLock](#) (M \*m)
- [~ReverseScopedPointerLock](#) ()

`template<class M> class OpenThreads::ReverseScopedPointerLock< M >`

### 4.17.1 Constructor & Destructor Documentation

4.17.1.1 [ReverseScopedPointerLock](#) (M \* *m*) [`inline`, `explicit`]

4.17.1.2 [~ReverseScopedPointerLock](#) () [`inline`]

The documentation for this class was generated from the following file:

- [ScopedLock](#)

## 4.18 ScopedLock< M > Class Template Reference

### Public Member Functions

- [ScopedLock](#) (M &m)
- [~ScopedLock](#) ()

```
template<class M> class OpenThreads::ScopedLock< M >
```

### 4.18.1 Constructor & Destructor Documentation

4.18.1.1 [ScopedLock](#) (M & *m*) [inline, explicit]

4.18.1.2 [~ScopedLock](#) () [inline]

The documentation for this class was generated from the following file:

- [ScopedLock](#)

## 4.19 ScopedPointerLock< M > Class Template Reference

### Public Member Functions

- [ScopedPointerLock](#) (M \*m)
- [~ScopedPointerLock](#) ()

`template<class M> class OpenThreads::ScopedPointerLock< M >`

### 4.19.1 Constructor & Destructor Documentation

4.19.1.1 [ScopedPointerLock](#) (M \* *m*) [`inline`, `explicit`]

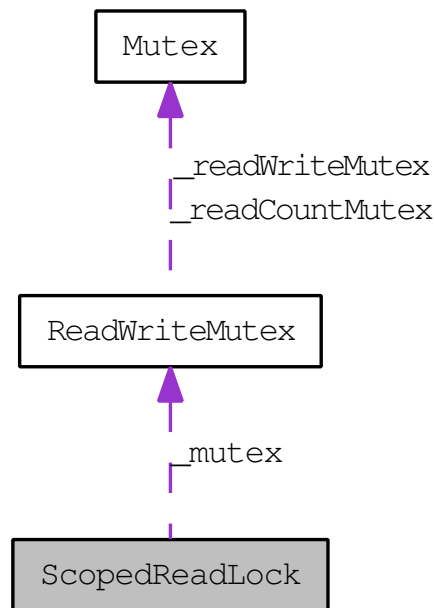
4.19.1.2 [~ScopedPointerLock](#) () [`inline`]

The documentation for this class was generated from the following file:

- [ScopedLock](#)

## 4.20 ScopedReadLock Class Reference

Collaboration diagram for ScopedReadLock:



### Public Member Functions

- [ScopedReadLock \(ReadWriteMutex &mutex\)](#)
- [~ScopedReadLock \(\)](#)

### Protected Member Functions

- [ScopedReadLock & operator= \(const ScopedReadLock &\)](#)

### Protected Attributes

- [ReadWriteMutex & \\_mutex](#)

### 4.20.1 Constructor & Destructor Documentation

4.20.1.1 [ScopedReadLock \(ReadWriteMutex & mutex\)](#) [inline]

4.20.1.2 [~ScopedReadLock \(\)](#) [inline]

### 4.20.2 Member Function Documentation

4.20.2.1 [ScopedReadLock& operator= \(const ScopedReadLock &\)](#) [inline, protected]

### 4.20.3 Member Data Documentation

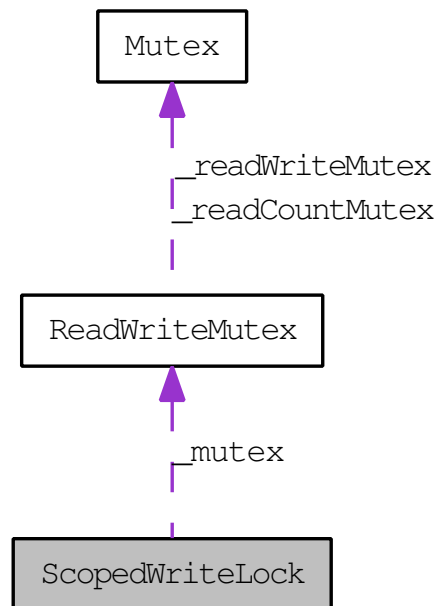
4.20.3.1 [ReadWriteMutex& \\_mutex](#) [protected]

The documentation for this class was generated from the following file:

- [ReadWriteMutex](#)

## 4.21 ScopedWriteLock Class Reference

Collaboration diagram for ScopedWriteLock:



### Public Member Functions

- [ScopedWriteLock \(ReadWriteMutex &mutex\)](#)
- [~ScopedWriteLock \(\)](#)

### Protected Member Functions

- [ScopedWriteLock & operator= \(const ScopedWriteLock &\)](#)

### Protected Attributes

- [ReadWriteMutex & \\_mutex](#)

#### 4.21.1 Constructor & Destructor Documentation

4.21.1.1 [ScopedWriteLock \(ReadWriteMutex & mutex\)](#) [inline]

4.21.1.2 [~ScopedWriteLock \(\)](#) [inline]

#### 4.21.2 Member Function Documentation

4.21.2.1 [ScopedWriteLock& operator= \(const ScopedWriteLock &\)](#) [inline, protected]

#### 4.21.3 Member Data Documentation

4.21.3.1 [ReadWriteMutex& \\_mutex](#) [protected]

The documentation for this class was generated from the following file:

- [ReadWriteMutex](#)

## 4.22 SemaLink Class Reference

#include <OpenThreads/sproc/SprocConditionPrivateData.h> Collaboration diagram for SemaLink:



### Friends

- class [Condition](#)
- class [ConditionDebug](#)
- class [SprocConditionPrivatedata](#)

### 4.22.1 Friends And Related Function Documentation

4.22.1.1 friend class [Condition](#) [friend]

4.22.1.2 friend class [ConditionDebug](#) [friend]

4.22.1.3 friend class [SprocConditionPrivatedata](#) [friend]

The documentation for this class was generated from the following file:

- [SprocConditionPrivateData.h](#)

## 4.23 SharedArena Class Reference

```
#include <OpenThreads/sproc/SharedArena.h>
```

### Friends

- class [Barrier](#)
- class [Condition](#)
- class [Mutex](#)

### 4.23.1 Friends And Related Function Documentation

**4.23.1.1 friend class Barrier [friend]**

**4.23.1.2 friend class Condition [friend]**

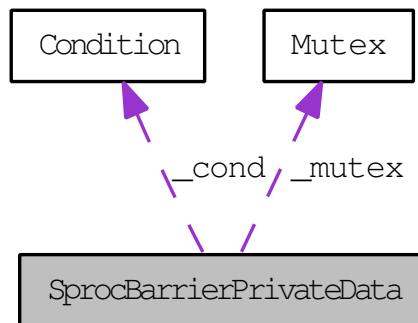
**4.23.1.3 friend class Mutex [friend]**

The documentation for this class was generated from the following file:

- [SharedArena.h](#)

## 4.24 SprocBarrierPrivateData Class Reference

`#include <OpenThreads/sproc/SprocBarrierPrivateData.h>` Collaboration diagram for SprocBarrierPrivateData:



### Friends

- class [Barrier](#)

### 4.24.1 Friends And Related Function Documentation

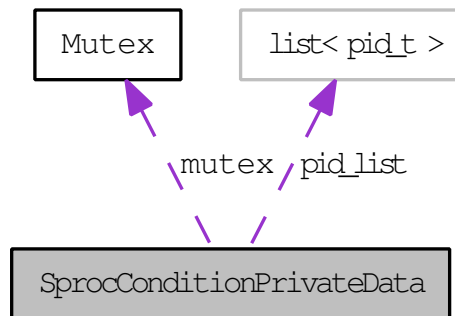
#### 4.24.1.1 friend class Barrier [friend]

The documentation for this class was generated from the following file:

- [SprocBarrierPrivateData.h](#)

## 4.25 SprocConditionPrivateData Class Reference

`#include <OpenThreads/sproc/SprocConditionPrivateData.h>` Collaboration diagram for SprocConditionPrivateData:



### Friends

- class [Condition](#)

### 4.25.1 Friends And Related Function Documentation

#### 4.25.1.1 friend class `Condition` [friend]

The documentation for this class was generated from the following file:

- [SprocConditionPrivateData.h](#)

## 4.26 SprocMutexPrivateData Class Reference

```
#include <OpenThreads/sproc/SprocMutexPrivateData.h>
```

### Friends

- class [Mutex](#)
- class [SprocThreadPrivateActions](#)

### 4.26.1 Friends And Related Function Documentation

**4.26.1.1 friend class `Mutex` [friend]**

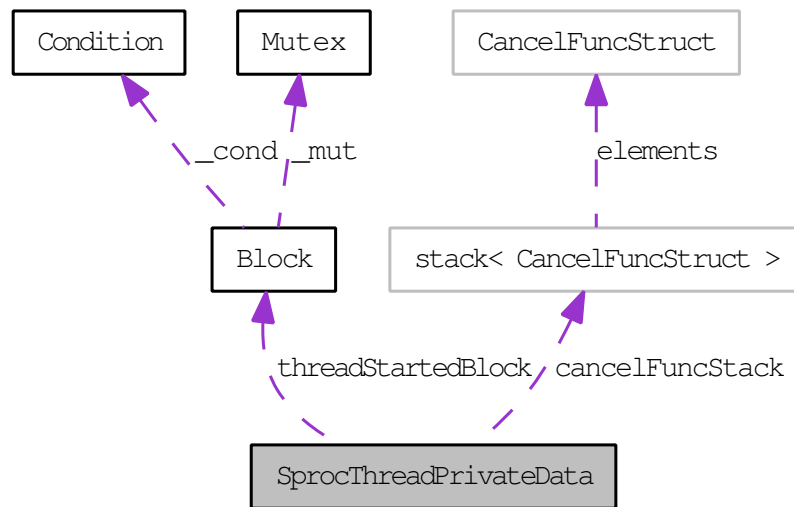
**4.26.1.2 friend class `SprocThreadPrivateActions` [friend]**

The documentation for this class was generated from the following file:

- [SprocMutexPrivateData.h](#)

## 4.27 SprocThreadPrivateData Class Reference

`#include <OpenThreads/sproc/SprocThreadPrivateData.h>` Collaboration diagram for SprocThreadPrivateData:



### Classes

- struct **CancelFuncStruct**

### Friends

- class [Thread](#)
- class [ThreadPrivateActions](#)

### 4.27.1 Friends And Related Function Documentation

4.27.1.1 **friend class Thread** [friend]

4.27.1.2 **friend class ThreadPrivateActions** [friend]

The documentation for this class was generated from the following file:

- [SprocThreadPrivateData.h](#)

## 4.28 Thread Class Reference

This class provides an object-oriented thread interface.

### Public Types

- enum `ThreadPolicy` { `THREAD_SCHEDULE_FIFO`, `THREAD_SCHEDULE_ROUND_ROBIN`, `THREAD_SCHEDULE_TIME_SHARE`, `THREAD_SCHEDULE_DEFAULT` }  
*Enumerated Type for thread scheduling policy.*
- enum `ThreadPriority` { `THREAD_PRIORITY_MAX`, `THREAD_PRIORITY_HIGH`, `THREAD_PRIORITY_NOMINAL`, `THREAD_PRIORITY_LOW`, `THREAD_PRIORITY_MIN`, `THREAD_PRIORITY_DEFAULT` }  
*Enumerated Type for thread priority.*

### Public Member Functions

- `Thread` ()  
*Constructor.*
- virtual `~Thread` ()  
*Destructor.*
- virtual int `cancel` ()  
*Cancel the thread.*
- virtual void `cancelCleanup` ()  
*Thread's cancel cleanup routine, called upon `cancel()`, after the cancelation has taken place, but before the thread exits completely.*
- int `detach` ()  
*Detach the thread from the calling process.*
- void \* `getImplementation` ()
- size\_t `getProcessId` ()  
*Get the thread's process id.*
- int `getSchedulePolicy` ()  
*Get the thread's policy (if able).*
- int `getSchedulePriority` ()  
*Get the thread's schedule priority (if able).*
- size\_t `getStackSize` ()  
*Get the thread's desired stack size.*
- int `getThreadId` ()  
*Get a unique thread id.*
- bool `isRunning` ()  
*Query the thread's running status.*
- int `join` ()  
*Join the calling process with the thread.*

- void `printSchedulingInfo ()`  
*Print the thread's scheduling information to stdout.*
- virtual void `run ()=0`  
*Thread's run method.*
- int `setCancelModeAsynchronous ()`  
*Mark the thread to cancel asynchronously on `Thread::cancel()`.*
- int `setCancelModeDeferred ()`  
*Mark the thread to cancel at the earliest convenience on `Thread::cancel()` (This is the default).*
- int `setCancelModeDisable ()`  
*Disable thread cancelation altogether.*
- int `setProcessorAffinity (unsigned int cpunum)`  
*Thread's processor affinity method.*
- int `setSchedulePolicy (ThreadPolicy policy)`  
*Set the thread's scheduling policy (if able).*
- int `setSchedulePriority (ThreadPriority priority)`  
*Set the thread's schedule priority.*
- int `setStackSize (size_t size)`  
*Set the thread's desired stack size (in bytes).*
- int `start ()`  
*Start the thread.*
- int `startThread ()`
- int `testCancel ()`  
*Test the cancel state of the thread.*

### Static Public Member Functions

- static `Thread * CurrentThread ()`  
*Return a pointer to the current running thread.*
- static int `GetConcurrency ()`  
*Get the concurrency level for a running application.*
- static `ThreadPriority GetMasterPriority ()`  
*This method will return the ThreadPriority of the master process.*
- static void `Init ()`  
*Initialize Threading in a program.*
- static int `microSleep (unsigned int microsec)`  
*microSleep method, equivalent to the posix `usleep(microsec)`.*
- static int `SetConcurrency (int concurrencyLevel)`  
*Set the concurrency level for a running application.*
- static int `YieldCurrentThread ()`  
*Yield the processor.*

## Friends

- class [ThreadPrivateActions](#)

*The Private Actions class is allowed to operate on private data.*

### 4.28.1 Detailed Description

This class provides an object-oriented thread interface.

### 4.28.2 Member Enumeration Documentation

#### 4.28.2.1 enum ThreadPolicy

Enumerated Type for thread scheduling policy.

##### Enumerator:

**THREAD\_SCHEDULE\_FIFO** First in, First out scheduling.

**THREAD\_SCHEDULE\_ROUND\_ROBIN** Round-robin scheduling (LINUX\_DEFAULT).

**THREAD\_SCHEDULE\_TIME\_SHARE** Time-share scheduling (IRIX\_DEFAULT).

**THREAD\_SCHEDULE\_DEFAULT** Default scheduling.

#### 4.28.2.2 enum ThreadPriority

Enumerated Type for thread priority.

##### Enumerator:

**THREAD\_PRIORITY\_MAX** The maximum possible priority.

**THREAD\_PRIORITY\_HIGH** A high (but not max) setting.

**THREAD\_PRIORITY\_NOMINAL** An average priority.

**THREAD\_PRIORITY\_LOW** A low (but not min) setting.

**THREAD\_PRIORITY\_MIN** The minimum possible priority.

**THREAD\_PRIORITY\_DEFAULT** Priority scheduling default.

### 4.28.3 Constructor & Destructor Documentation

#### 4.28.3.1 Thread ()

Constructor.

#### 4.28.3.2 ~Thread () [virtual]

Destructor.

### 4.28.4 Member Function Documentation

#### 4.28.4.1 int cancel () [virtual]

Cancel the thread. Equivalent to SIGKILL.

Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.28.4.2 virtual void cancelCleanup () [inline, virtual]

Thread's cancel cleanup routine, called upon [cancel\(\)](#), after the cancelation has taken place, but before the thread exits completely. This method should be used to repair parts of the thread's data that may have been damaged by a pre-mature cancel. No-op by default.

#### 4.28.4.3 Thread \* CurrentThread () [static]

Return a pointer to the current running thread.

#### 4.28.4.4 int detach ()

Detach the thread from the calling process. Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.28.4.5 int GetConcurrency () [static]

Get the concurrency level for a running application. In this case, a return code of 0 means that the application is in default mode. A return code of -1 means that the application is incapable of setting an arbitrary concurrency, because it is a one-to-one execution model (sprocs, linuxThreads)

#### 4.28.4.6 void\* getImplementation () [inline]

#### 4.28.4.7 static ThreadPriority GetMasterPriority () [inline, static]

This method will return the ThreadPriority of the master process. (ie, the one calling the thread->start() methods for the first time) The method will almost certainly return Thread::THREAD\_PRIORITY\_DEFAULT if Init() has not been called.

Returns the Thread::ThreadPriority of the master thread.

#### 4.28.4.8 size\_t getProcessId ()

Get the thread's process id. This is the pthread\_t or pid\_t value depending on the threading model being used.

Returns thread process id.

#### 4.28.4.9 int getSchedulePolicy ()

Get the thread's policy (if able). Note setting the environment variable OUTPUT\_THREADLIB\_SCHEDULING\_INFO will output scheduling information for each thread to stdout. Returns policy if normal, -1 if errno set, errno code otherwise.

#### 4.28.4.10 int getSchedulePriority ()

Get the thread's schedule priority (if able). Note setting the environment variable OUTPUT\_THREADLIB\_SCHEDULING\_INFO will output scheduling information for each thread to stdout. Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.28.4.11 size\_t getStackSize ()

Get the thread's desired stack size. Returns the thread's stack size. 0 indicates that the stack size has either not yet been initialized, or not yet been specified by the application.

#### 4.28.4.12 int getThreadId ()

Get a unique thread id. This id is monotonically increasing.

Returns a unique thread identifier

#### 4.28.4.13 void Init () [static]

Initialize Threading in a program. This method must be called before you can do any threading in a program.

#### 4.28.4.14 bool isRunning ()

Query the thread's running status. Returns true if running, false if not.

#### 4.28.4.15 int join ()

Join the calling process with the thread. Returns 0 if normal, -1 if errno set, errno code otherwise.

#### 4.28.4.16 int microSleep (unsigned int *microsec*) [static]

microSleep method, equivalent to the posix usleep(microsec). This is not strictly thread API but is used so often with threads. It's basically UNIX usleep. Parameter is number of microseconds we current thread to sleep. Returns 0 on succes, non-zero on failure (UNIX errno or GetLastError()) will give detailed description.

#### 4.28.4.17 void printSchedulingInfo ()

Print the thread's scheduling information to stdout.

**4.28.4.18 virtual void run () [pure virtual]**

Thread's run method. Must be implemented by derived classes. This is where the action happens.

**4.28.4.19 int setCancelModeAsynchronous ()**

Mark the thread to cancel asynchronously on [Thread::cancel\(\)](#). (May not be available with process-level implementations).

Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.20 int setCancelModeDeferred ()**

Mark the thread to cancel at the earliest convenience on [Thread::cancel\(\)](#) (This is the default). Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.21 int setCancelModeDisable ()**

Disable thread cancelation altogether. [Thread::cancel\(\)](#) has no effect.

Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.22 int SetConcurrency (int *concurrencyLevel*) [static]**

Set the concurrency level for a running application. This method only has effect if the pthreads thread model is being used, and then only when that model is many-to-one (eg. irix). In other cases it is ignored. The concurrency level is only a *\*hint\** as to the number of execution vehicles to use, the actual implementation may do anything it wants. Setting the value to 0 returns things to their default state.

Returns previous concurrency level, -1 indicates no-op.

**4.28.4.23 int setProcessorAffinity (unsigned int *cpunum*)**

Thread's processor affinity method. This binds a thread to a processor whenever possible. This call must be made before [start\(\)](#) or [startThread\(\)](#) and has no effect after the thread has been running. In the pthreads implementation, this is only implemented on sgi, through a pthread extension. On other pthread platforms this is ignored. Returns 0 on success, implementation's error on failure, or -1 if ignored.

**4.28.4.24 int setSchedulePolicy (ThreadPolicy *policy*)**

Set the thread's scheduling policy (if able). Note On some implementations (notably IRIX Sprocs & LinuxThreads) The policy may prohibit the use of SCHEDULE\_ROUND\_ROBIN and SCHEDULE\_FIFO policies - due to their real-time nature, and the danger of deadlocking the machine when used as super-user. In such cases, the command is a no-op.

setting the environment variable OUTPUT\_THREADLIB\_SCHEDULING\_INFO will output scheduling information for each thread to stdout. Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.25 int setSchedulePriority (ThreadPriority *priority*)**

Set the thread's schedule priority. This is a complex method. Beware of thread priorities when using a many-to-many kernel entity implementation (such as IRIX pthreads). If one is not careful to manage the thread priorities, a priority inversion deadlock can easily occur (Although the [OpenThreads::Mutex](#) & [OpenThreads::Barrier](#) constructs have been designed with this scenario in mind). Unless you have explicit need to set the schedule priorities for a given task, it is best to leave them alone.

Note some implementations (notably LinuxThreads and IRIX Sprocs) only allow you to decrease thread priorities dynamically. Thus, a lower priority thread will not allow its priority to be raised on the fly.

setting the environment variable OUTPUT\_THREADLIB\_SCHEDULING\_INFO will output scheduling information for each thread to stdout. Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.26 int setStackSize (size\_t *size*)**

Set the thread's desired stack size (in bytes). This method is an attribute of the thread and must be called *\*before\** the [start\(\)](#) method is invoked.

Note a return code of 13 (EACCESS) means that the thread stack size can no longer be changed. Returns 0 if normal, -1 if errno set, errno code otherwise.

**4.28.4.27 int start ()**

Start the thread. This method will configure the thread, set it's priority, and spawn it.

Note if the stack size specified `setStackSize` is smaller than the smallest allowable stack size, the threads stack size will be set to the minimum allowed, and may be retrieved via the [getStackSize\(\)](#) Returns 0 if normal, -1 if `errno` set, `errno` code otherwise.

**4.28.4.28 int startThread ()****4.28.4.29 int testCancel ()**

Test the cancel state of the thread. If the thread has been canceled this method will cause the thread to exit now. This method operates on the calling thread.

Returns 0 if normal, -1 if called from a thread other than this.

**4.28.4.30 int YieldCurrentThread () [static]**

Yield the processor. Note This method operates on the calling process. And is equivalent to calling `sched_yield()`. Returns 0 if normal, -1 if `errno` set, `errno` code otherwise.

**4.28.5 Friends And Related Function Documentation****4.28.5.1 friend class ThreadPrivateActions [friend]**

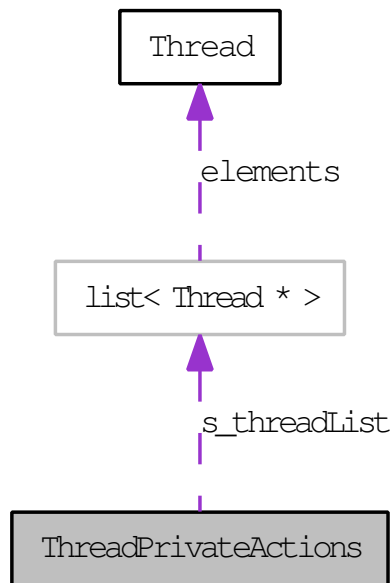
The Private Actions class is allowed to operate on private data.

The documentation for this class was generated from the following files:

- [Thread](#)
- [Win32Thread.cpp](#)

## 4.29 ThreadPrivateActions Class Reference

#include <OpenThreads/sproc/SprocThreadPrivateActions.h> Collaboration diagram for ThreadPrivateActions:



### Static Public Member Functions

- static void [PopCancelFunction](#) ()
- static void [PushCancelFunction](#) (void(\*routine)(void \*), void \*arg)
- static void [ThreadCancelTest](#) ()

### Friends

- class [Thread](#)

### 4.29.1 Member Function Documentation

**4.29.1.1** static void [PopCancelFunction](#) () [static]

**4.29.1.2** static void [PushCancelFunction](#) (void(\*)*(void \*) routine*, void \* *arg*) [static]

**4.29.1.3** static void [ThreadCancelTest](#) () [static]

### 4.29.2 Friends And Related Function Documentation

**4.29.2.1** [Thread](#) [friend]

The documentation for this class was generated from the following files:

- [SprocThreadPrivateActions.h](#)
- [Win32Thread.cpp](#)

## 4.30 TlsHolder Struct Reference

```
#include <OpenThreads/win32/Win32ThreadPrivateData.h>
```

### Public Member Functions

- [TlsHolder \(\)](#)
- [~TlsHolder \(\)](#)
- [DWORD getld \(\)](#)

### 4.30.1 Constructor & Destructor Documentation

**4.30.1.1** [TlsHolder \(\)](#) [[inline](#)]

**4.30.1.2** [~TlsHolder \(\)](#) [[inline](#)]

### 4.30.2 Member Function Documentation

**4.30.2.1** [DWORD getld \(\)](#) [[inline](#)]

The documentation for this struct was generated from the following file:

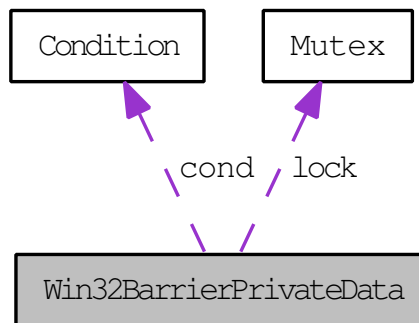
- [Win32ThreadPrivateData.h](#)

## 4.31 Win32BarrierPrivateData Class Reference

```
#include <OpenThreads/win32/Win32BarrierPrivateData.h>
Win32BarrierPrivateData:
```

diagram

for



### Friends

- class [Barrier](#)

### 4.31.1 Friends And Related Function Documentation

#### 4.31.1.1 friend class Barrier [friend]

The documentation for this class was generated from the following files:

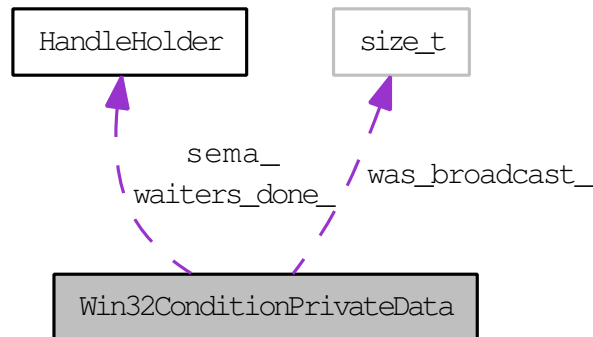
- [Win32BarrierPrivateData.h](#)
- [Win32ThreadBarrier.cpp](#)

## 4.32 Win32ConditionPrivateData Class Reference

#include <OpenThreads/win32/Win32ConditionPrivateData.h> Collaboration  
Win32ConditionPrivateData:

diagram

for



### Public Member Functions

- [Win32ConditionPrivateData \(\)](#)
- [~Win32ConditionPrivateData \(\)](#)
- int [broadcast \(\)](#)
- int [signal \(\)](#)
- int [wait \(Mutex &external\\_mutex, long timeout\\_ms\)](#)

### Public Attributes

- long [waiters\\_](#)  
*number of waiters.*

### Protected Attributes

- [HandleHolder sema\\_](#)  
*Serialize access to the waiters count.*
- [HandleHolder waiters\\_done\\_](#)  
*An auto reset event used by the broadcast/signal thread to wait for the waiting thread(s) to wake up and get a chance at the semaphore.*
- [size\\_t was\\_broadcast\\_](#)  
*Keeps track of whether we were broadcasting or just signaling.*

### Friends

- class [Condition](#)

### 4.32.1 Constructor & Destructor Documentation

4.32.1.1 `Win32ConditionPrivateData ()` [inline]

4.32.1.2 `~Win32ConditionPrivateData ()`

### 4.32.2 Member Function Documentation

4.32.2.1 `int broadcast ()` [inline]

4.32.2.2 `int signal ()` [inline]

4.32.2.3 `int wait (Mutex & external_mutex, long timeout_ms)` [inline]

### 4.32.3 Friends And Related Function Documentation

4.32.3.1 `friend class Condition` [friend]

### 4.32.4 Member Data Documentation

4.32.4.1 `HandleHolder sema_` [protected]

Serialize access to the waiters count. [Mutex](#) `waiters_lock_`; Queue up threads waiting for the condition to become signaled.

4.32.4.2 `long waiters_`

number of waiters.

4.32.4.3 `HandleHolder waiters_done_` [protected]

An auto reset event used by the broadcast/signal thread to wait for the waiting thread(s) to wake up and get a chance at the semaphore.

4.32.4.4 `size_t was_broadcast_` [protected]

Keeps track of whether we were broadcasting or just signaling.

The documentation for this class was generated from the following files:

- [Win32ConditionPrivateData.h](#)
- [WIN32Condition.cpp](#)

## 4.33 Win32MutexPrivateData Class Reference

```
#include <OpenThreads/win32/Win32MutexPrivateData.h>
```

### Friends

- class [Condition](#)
- class [Mutex](#)

### 4.33.1 Friends And Related Function Documentation

**4.33.1.1 friend class Condition [friend]**

**4.33.1.2 friend class Mutex [friend]**

The documentation for this class was generated from the following files:

- [Win32MutexPrivateData.h](#)
- [Win32Mutex.cpp](#)

## 4.34 Win32ThreadCanceled Struct Reference

The documentation for this struct was generated from the following file:

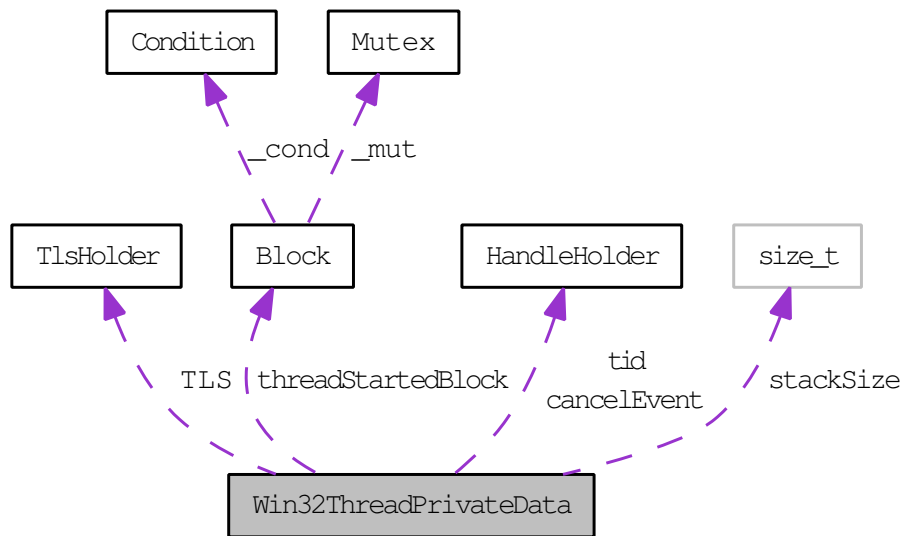
- [Win32Thread.cpp](#)

## 4.35 Win32ThreadPrivateData Class Reference

`#include <OpenThreads/win32/Win32ThreadPrivateData.h>` Collaboration  
 Win32ThreadPrivateData:

diagram

for



### Classes

- struct [TlsHolder](#)

### Public Attributes

- [HandleHolder](#) `cancelEvent`

### Static Public Attributes

- static [TlsHolder](#) `TLS`

### Friends

- class [Thread](#)
- class [ThreadPrivateActions](#)

### 4.35.1 Friends And Related Function Documentation

4.35.1.1 friend class [Thread](#) [friend]

4.35.1.2 friend class [ThreadPrivateActions](#) [friend]

### 4.35.2 Member Data Documentation

4.35.2.1 [HandleHolder](#) `cancelEvent`

4.35.2.2 [Win32ThreadPrivateData::TlsHolder](#) `TLS` [static]

The documentation for this class was generated from the following files:

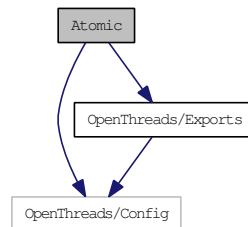
- [Win32ThreadPrivateData.h](#)
- [Win32Thread.cpp](#)

# File Documentation

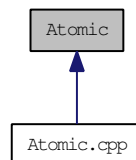
---

## 5.1 Atomic File Reference

```
#include <OpenThreads/Config>
#include <OpenThreads/Exports>
Include dependency graph for Atomic:
```



This graph shows which files directly or indirectly include this file:



### Classes

- class [Atomic](#)  
*This class provides an atomic increment and decrement operation.*
- class [AtomicPtr](#)  
*This class provides an atomic pointer assignment using cas operations.*

### Namespaces

- namespace [OpenThreads](#)

### Defines

- `#define \_OPENTHREADS\_ATOMIC\_INLINE inline`

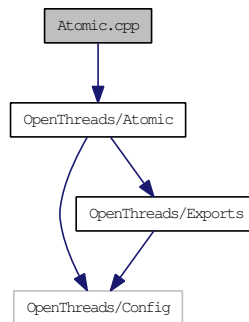
#### 5.1.1 Define Documentation

##### 5.1.1.1 `#define \_OPENTHREADS\_ATOMIC\_INLINE inline`

## 5.2 Atomic.cpp File Reference

```
#include <OpenThreads/Atomic>
```

Include dependency graph for Atomic.cpp:



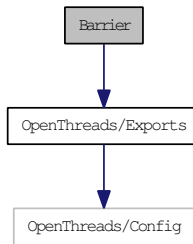
### Namespaces

- namespace [OpenThreads](#)

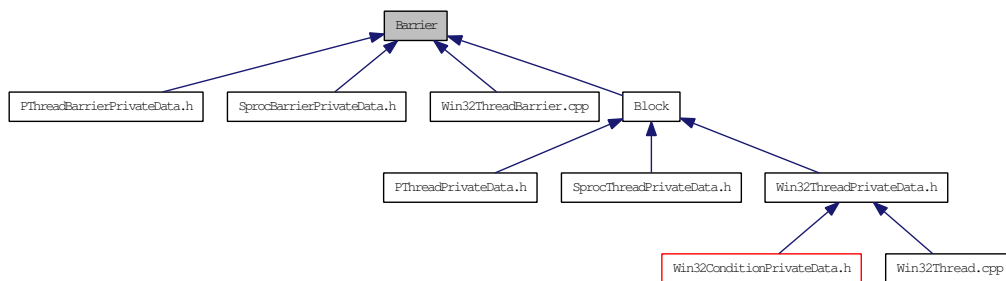
## 5.3 Barrier File Reference

```
#include <OpenThreads/Exports>
```

Include dependency graph for Barrier:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Barrier](#)  
*This class provides an object-oriented thread barrier interface.*

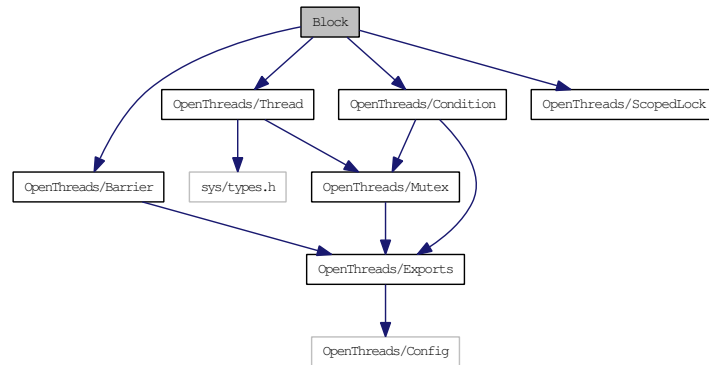
### Namespaces

- namespace [OpenThreads](#)

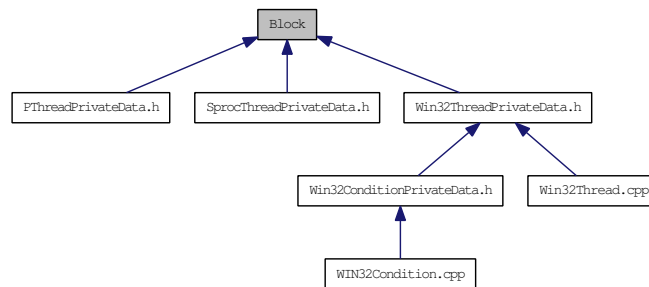
## 5.4 Block File Reference

```
#include <OpenThreads/Thread>
#include <OpenThreads/Barrier>
#include <OpenThreads/Condition>
#include <OpenThreads/ScopedLock>
```

Include dependency graph for Block:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Block](#)  
*Block* is a block that can be used to halt a thread that is waiting another thread to release it.
- class [BlockCount](#)  
*BlockCount* is a block that can be used to halt a thread that is waiting for a specified number of operations to be completed.

### Namespaces

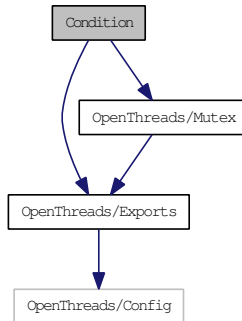
- namespace [OpenThreads](#)

## 5.5 Condition File Reference

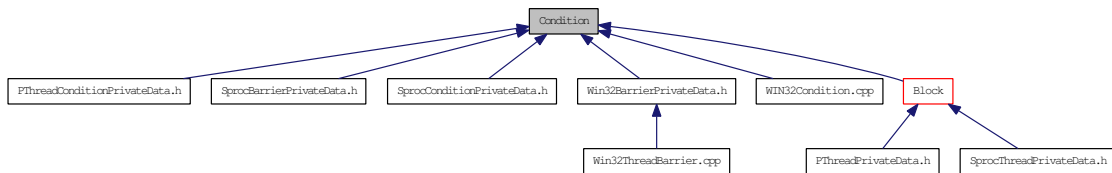
```
#include <OpenThreads/Exports>
```

```
#include <OpenThreads/Mutex>
```

Include dependency graph for Condition:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Condition](#)  
*This class provides an object-oriented thread condition interface.*

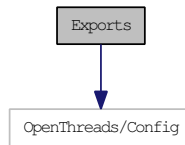
### Namespaces

- namespace [OpenThreads](#)

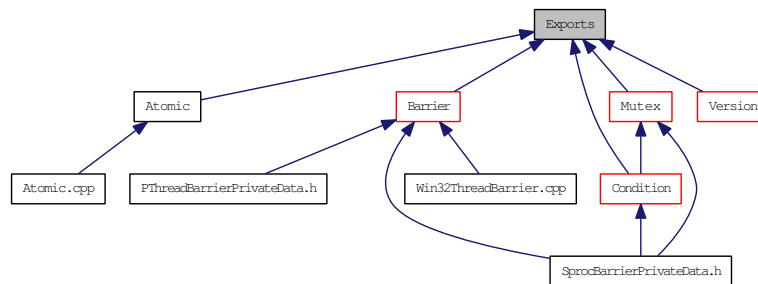
## 5.6 Exports File Reference

```
#include <OpenThreads/Config>
```

Include dependency graph for Exports:



This graph shows which files directly or indirectly include this file:



### Defines

- #define [OPENTHREAD\\_EXPORT\\_DIRECTIVE](#)

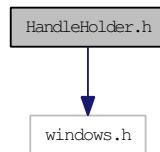
#### 5.6.1 Define Documentation

##### 5.6.1.1 #define OPENTHREAD\_EXPORT\_DIRECTIVE

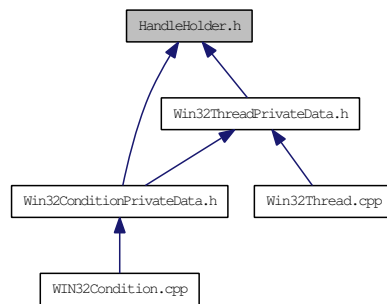
## 5.7 HandleHolder.h File Reference

```
#include <windows.h>
```

Include dependency graph for HandleHolder.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [HandleHolder](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

#### 5.7.1 Define Documentation

##### 5.7.1.1 #define WIN32\_LEAN\_AND\_MEAN

## 5.8 mainpage.h File Reference

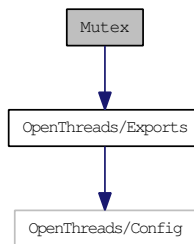
### 5.8.1 Detailed Description

This file contains doxygen special commands and text for the [Main Page](#) and some other minor aspects of this documentation. It is not part of [OpenThreads](#).

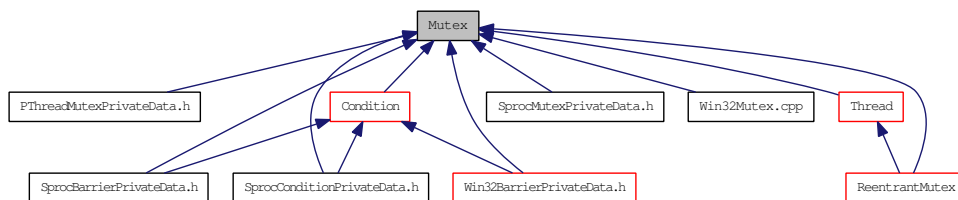
## 5.9 Mutex File Reference

```
#include <OpenThreads/Exports>
```

Include dependency graph for Mutex:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Mutex](#)  
*This class provides an object-oriented thread mutex interface.*

### Namespaces

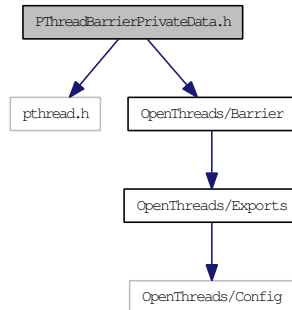
- namespace [OpenThreads](#)

## 5.10 PThreadBarrierPrivateData.h File Reference

```
#include <pthread.h>
```

```
#include <OpenThreads/Barrier>
```

Include dependency graph for PThreadBarrierPrivateData.h:



### Classes

- class [PThreadBarrierPrivateData](#)

### Namespaces

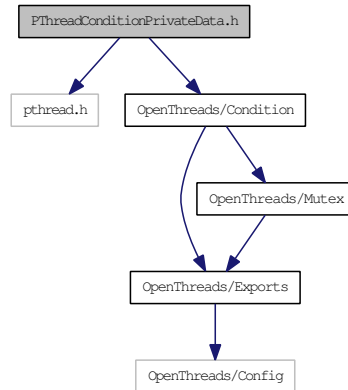
- namespace [OpenThreads](#)

## 5.11 PThreadConditionPrivateData.h File Reference

```
#include <pthread.h>
```

```
#include <OpenThreads/Condition>
```

Include dependency graph for PThreadConditionPrivateData.h:



### Classes

- class [PThreadConditionPrivateData](#)

### Namespaces

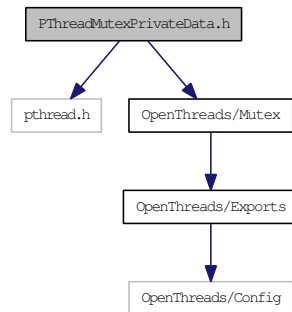
- namespace [OpenThreads](#)

## 5.12 PThreadMutexPrivateData.h File Reference

```
#include <pthread.h>
```

```
#include <OpenThreads/Mutex>
```

Include dependency graph for PThreadMutexPrivateData.h:



### Classes

- class [PThreadMutexPrivateData](#)

### Namespaces

- namespace [OpenThreads](#)

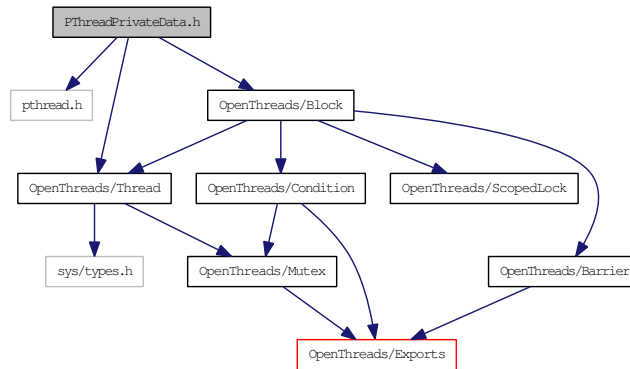
## 5.13 PThreadPrivateData.h File Reference

```
#include <pthread.h>
```

```
#include <OpenThreads/Thread>
```

```
#include <OpenThreads/Block>
```

Include dependency graph for PThreadPrivateData.h:



### Classes

- class [PThreadPrivateData](#)

### Namespaces

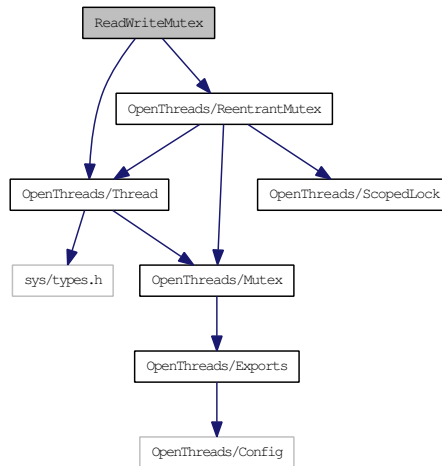
- namespace [OpenThreads](#)

## 5.14 ReadWriteMutex File Reference

```
#include <OpenThreads/Thread>
```

```
#include <OpenThreads/ReentrantMutex>
```

Include dependency graph for ReadWriteMutex:



### Classes

- class [ReadWriteMutex](#)
- class [ScopedReadLock](#)
- class [ScopedWriteLock](#)

### Namespaces

- namespace [OpenThreads](#)

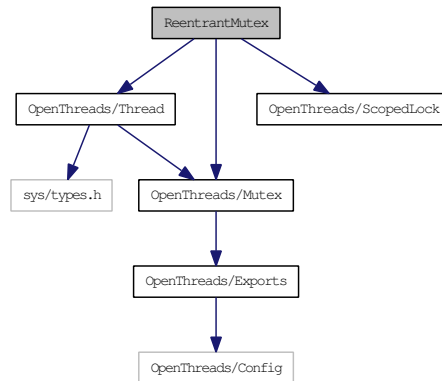
## 5.15 ReentrantMutex File Reference

```
#include <OpenThreads/Thread>
```

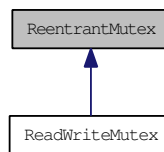
```
#include <OpenThreads/Mutex>
```

```
#include <OpenThreads/ScopedLock>
```

Include dependency graph for ReentrantMutex:



This graph shows which files directly or indirectly include this file:



### Classes

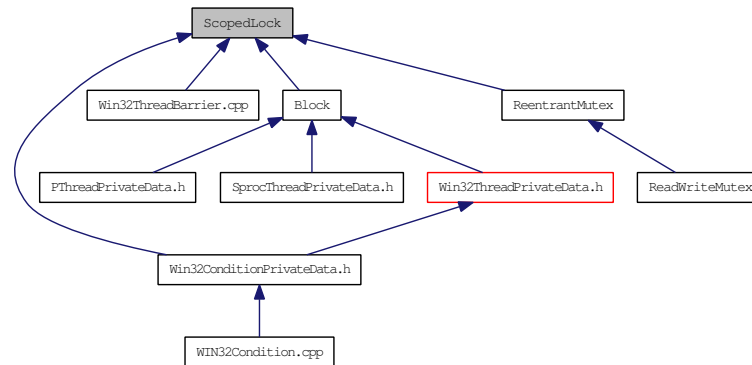
- class [ReentrantMutex](#)

### Namespaces

- namespace [OpenThreads](#)

## 5.16 ScopedLock File Reference

This graph shows which files directly or indirectly include this file:



### Classes

- class [ReverseScopedLock< M >](#)
- class [ReverseScopedPointerLock< M >](#)
- class [ScopedLock< M >](#)
- class [ScopedPointerLock< M >](#)

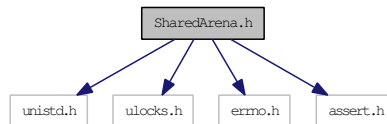
### Namespaces

- namespace [OpenThreads](#)

## 5.17 SharedArena.h File Reference

```
#include <unistd.h>
#include <ulocks.h>
#include <errno.h>
#include <assert.h>
```

Include dependency graph for SharedArena.h:



### Classes

- class [SharedArena](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- #define [OT\\_USESHAREDONLY](#)

#### 5.17.1 Define Documentation

##### 5.17.1.1 #define OT\_USESHAREDONLY

## 5.18 SprocBarrierPrivateData.h File Reference

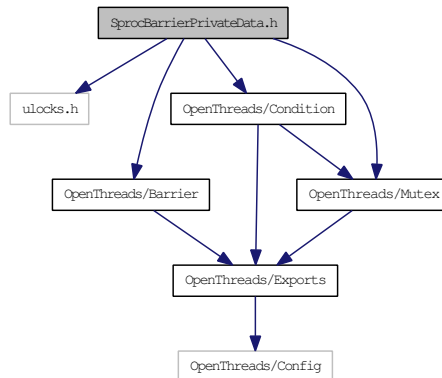
```
#include <ulocks.h>
```

```
#include <OpenThreads/Barrier>
```

```
#include <OpenThreads/Condition>
```

```
#include <OpenThreads/Mutex>
```

Include dependency graph for SprocBarrierPrivateData.h:



### Classes

- class [SprocBarrierPrivateData](#)

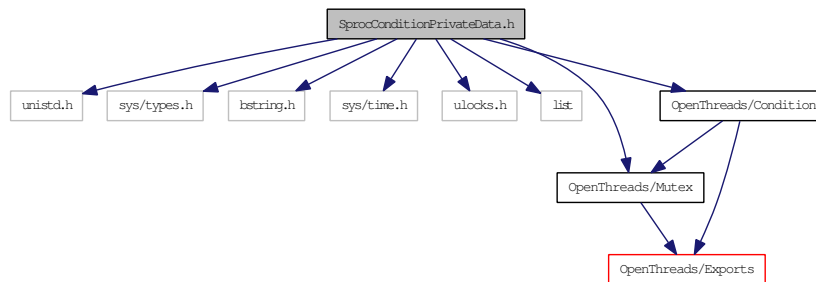
### Namespaces

- namespace [OpenThreads](#)

## 5.19 SprocConditionPrivateData.h File Reference

```
#include <unistd.h>
#include <sys/types.h>
#include <bstring.h>
#include <sys/time.h>
#include <ulocks.h>
#include <list>
#include <OpenThreads/Mutex>
#include <OpenThreads/Condition>
```

Include dependency graph for SprocConditionPrivateData.h:



### Classes

- class [SemaLink](#)
- class [SprocConditionPrivateData](#)

### Namespaces

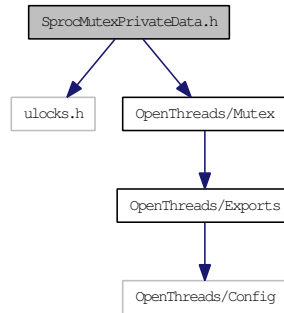
- namespace [OpenThreads](#)

## 5.20 SprocMutexPrivateData.h File Reference

```
#include <ulocks.h>
```

```
#include <OpenThreads/Mutex>
```

Include dependency graph for SprocMutexPrivateData.h:



### Classes

- class [SprocMutexPrivateData](#)

### Namespaces

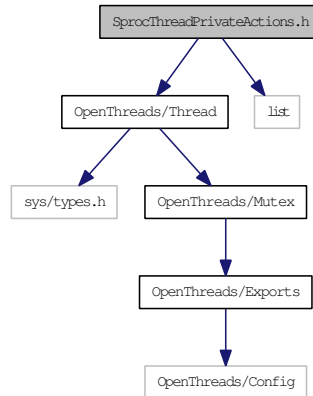
- namespace [OpenThreads](#)

## 5.21 SprocThreadPrivateActions.h File Reference

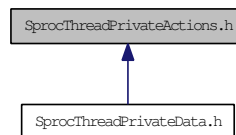
```
#include <OpenThreads/Thread>
```

```
#include <list>
```

Include dependency graph for SprocThreadPrivateActions.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [ThreadPrivateActions](#)

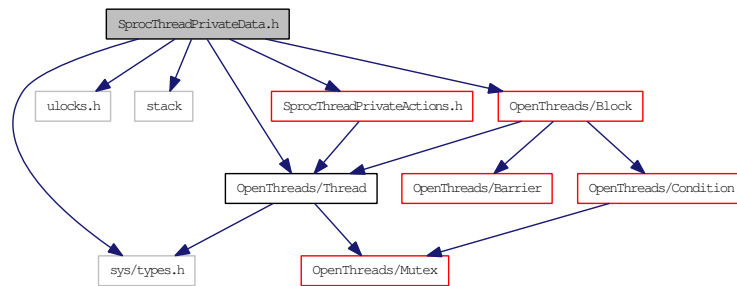
### Namespaces

- namespace [OpenThreads](#)

## 5.22 SprocThreadPrivateData.h File Reference

```
#include <sys/types.h>
#include <ulocks.h>
#include <stack>
#include <OpenThreads/Thread>
#include <OpenThreads/Block>
#include "SprocThreadPrivateActions.h"
```

Include dependency graph for SprocThreadPrivateData.h:



### Classes

- struct **CancelFuncStruct**
- class [SprocThreadPrivateData](#)

### Namespaces

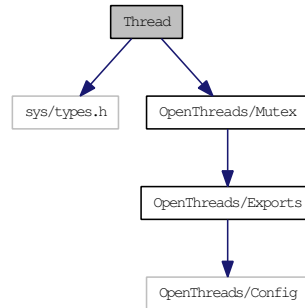
- namespace [OpenThreads](#)

## 5.23 Thread File Reference

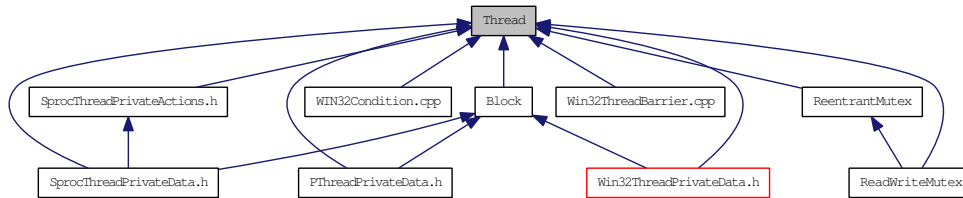
```
#include <sys/types.h>
```

```
#include <OpenThreads/Mutex>
```

Include dependency graph for Thread:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Thread](#)

*This class provides an object-oriented thread interface.*

### Namespaces

- namespace [OpenThreads](#)

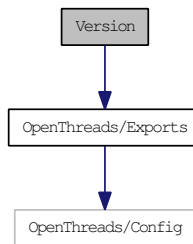
### Functions

- OPENTHREAD\_EXPORT\_DIRECTIVE int [GetNumberOfProcessors](#) ()  
*Get the number of processors.*
- OPENTHREAD\_EXPORT\_DIRECTIVE int [SetProcessorAffinityOfCurrentThread](#) (unsigned int cpunum)  
*Set the processor affinity of current thread.*

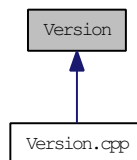
## 5.24 Version File Reference

```
#include <OpenThreads/Exports>
```

Include dependency graph for Version:



This graph shows which files directly or indirectly include this file:



### Defines

- #define [OPENTHREADS\\_MAJOR\\_VERSION](#) 2
- #define [OPENTHREADS\\_MINOR\\_VERSION](#) 4
- #define [OPENTHREADS\\_PATCH\\_VERSION](#) 0
- #define [OPENTHREADS\\_SOVERSION](#) 11
- #define [OPENTHREADS\\_VERSION](#) 1

### Functions

- OPENTHREAD\_EXPORT\_DIRECTIVE const char \* [OpenThreadsGetLibraryName](#) ()  
The [OpenThreadsGetLibraryName\(\)](#) method returns the library name in human-friendly form.
- OPENTHREAD\_EXPORT\_DIRECTIVE const char \* [OpenThreadsGetSOVersion](#) ()  
The [OpenThreadsGetSOVersion\(\)](#) method returns the OpenSceneGraph soversion number.
- OPENTHREAD\_EXPORT\_DIRECTIVE const char \* [OpenThreadsGetVersion](#) ()  
[OpenThreadsGetVersion\(\)](#) returns the library version number.

#### 5.24.1 Define Documentation

5.24.1.1 #define [OPENTHREADS\\_MAJOR\\_VERSION](#) 2

5.24.1.2 #define [OPENTHREADS\\_MINOR\\_VERSION](#) 4

5.24.1.3 #define [OPENTHREADS\\_PATCH\\_VERSION](#) 0

5.24.1.4 #define [OPENTHREADS\\_SOVERSION](#) 11

5.24.1.5 #define [OPENTHREADS\\_VERSION](#) 1

#### 5.24.2 Function Documentation

5.24.2.1 OPENTHREAD\_EXPORT\_DIRECTIVE const char\* [OpenThreadsGetLibraryName](#) ()

The [OpenThreadsGetLibraryName\(\)](#) method returns the library name in human-friendly form.

**5.24.2.2 OPENTHREAD\_EXPORT\_DIRECTIVE const char\* OpenThreadsGetSOVersion ()**

The [OpenThreadsGetSOVersion\(\)](#) method returns the OpenSceneGraph soversion number.

**5.24.2.3 OPENTHREAD\_EXPORT\_DIRECTIVE const char\* OpenThreadsGetVersion ()**

[OpenThreadsGetVersion\(\)](#) returns the library version number. Numbering convention : OpenThreads-1.0 will return 1.0 from OpenThreadsGetVersion.

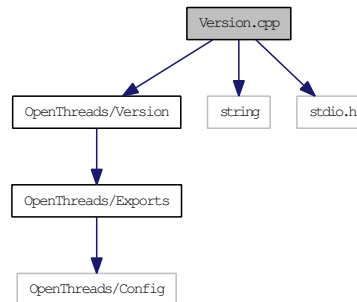
## 5.25 Version.cpp File Reference

```
#include <OpenThreads/Version>
```

```
#include <string>
```

```
#include <stdio.h>
```

Include dependency graph for Version.cpp:



### Functions

- `const char * OpenThreadsGetLibraryName ()`  
*The [OpenThreadsGetLibraryName\(\)](#) method returns the library name in human-friendly form.*
- `const char * OpenThreadsGetSOVersion ()`  
*The [OpenThreadsGetSOVersion\(\)](#) method returns the OpenSceneGraph soversion number.*
- `const char * OpenThreadsGetVersion ()`  
*[OpenThreadsGetVersion\(\)](#) returns the library version number.*

### 5.25.1 Function Documentation

#### 5.25.1.1 `const char* OpenThreadsGetLibraryName ()`

The [OpenThreadsGetLibraryName\(\)](#) method returns the library name in human-friendly form.

#### 5.25.1.2 `const char* OpenThreadsGetSOVersion ()`

The [OpenThreadsGetSOVersion\(\)](#) method returns the OpenSceneGraph soversion number.

#### 5.25.1.3 `const char* OpenThreadsGetVersion ()`

[OpenThreadsGetVersion\(\)](#) returns the library version number. Numbering convention : OpenThreads-1.0 will return 1.0 from [OpenThreadsGetVersion](#).

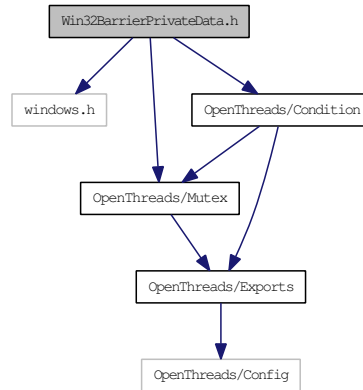
## 5.26 Win32BarrierPrivateData.h File Reference

```
#include <windows.h>
```

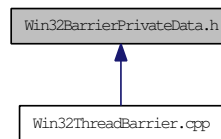
```
#include <OpenThreads/Mutex>
```

```
#include <OpenThreads/Condition>
```

Include dependency graph for Win32BarrierPrivateData.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Win32BarrierPrivateData](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- #define [\\_WIN32\\_WINNT](#) 0x0400
- #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

#### 5.26.1 Define Documentation

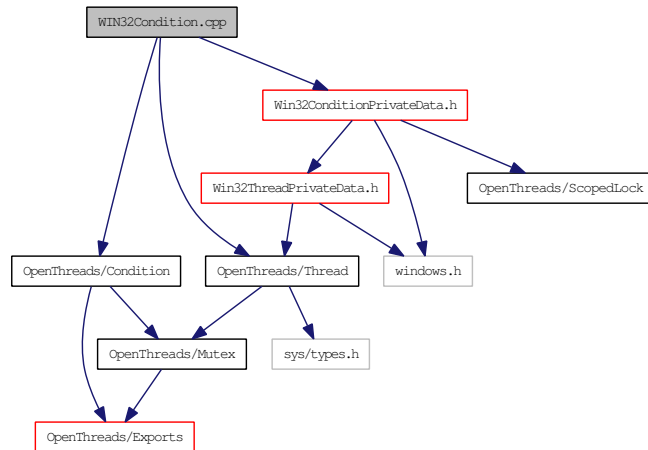
5.26.1.1 #define [\\_WIN32\\_WINNT](#) 0x0400

5.26.1.2 #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

## 5.27 WIN32Condition.cpp File Reference

```
#include <OpenThreads/Condition>
#include <OpenThreads/Thread>
#include "Win32ConditionPrivateData.h"
```

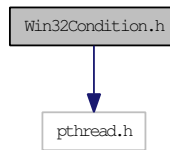
Include dependency graph for WIN32Condition.cpp:



## 5.28 Win32Condition.h File Reference

```
#include <pthread.h>
```

Include dependency graph for Win32Condition.h:



### Classes

- class [Condition](#)

### Namespaces

- namespace [Producer](#)

### Defines

- #define [PRODUCER\\_CONDITION](#)

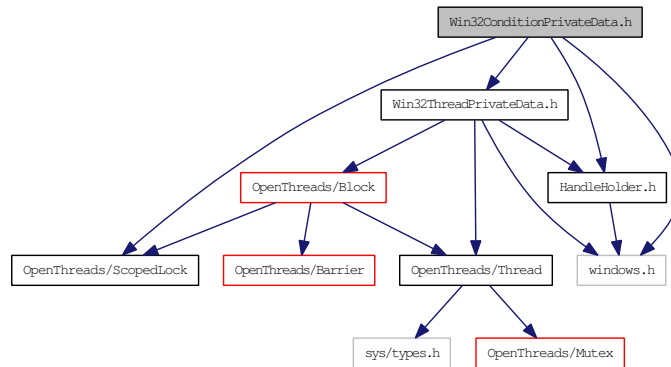
#### 5.28.1 Define Documentation

##### 5.28.1.1 #define PRODUCER\_CONDITION

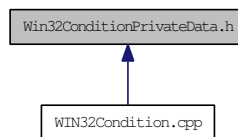
## 5.29 Win32ConditionPrivateData.h File Reference

```
#include <windows.h>
#include <OpenThreads/ScopedLock>
#include "Win32ThreadPrivateData.h"
#include "HandleHolder.h"
```

Include dependency graph for Win32ConditionPrivateData.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Win32ConditionPrivateData](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- #define [\\_WIN32\\_WINNT](#) 0x0400
- #define [InterlockedGet\(x\)](#) InterlockedExchangeAdd(x,0)
- #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

#### 5.29.1 Define Documentation

5.29.1.1 #define [\\_WIN32\\_WINNT](#) 0x0400

5.29.1.2 #define [InterlockedGet\(x\)](#) InterlockedExchangeAdd(x,0)

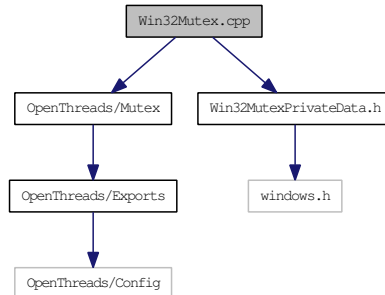
5.29.1.3 #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

## 5.30 Win32Mutex.cpp File Reference

```
#include <OpenThreads/Mutex>
```

```
#include "Win32MutexPrivateData.h"
```

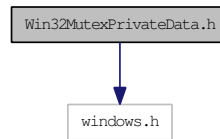
Include dependency graph for Win32Mutex.cpp:



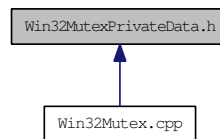
## 5.31 Win32MutexPrivateData.h File Reference

```
#include <windows.h>
```

Include dependency graph for Win32MutexPrivateData.h:



This graph shows which files directly or indirectly include this file:



### Classes

- class [Win32MutexPrivateData](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- #define [\\_WIN32\\_WINNT](#) 0x0400
- #define [USE\\_CRITICAL\\_SECTION](#)
- #define [WIN32\\_LEAN\\_AND\\_MEAN](#)

#### 5.31.1 Define Documentation

**5.31.1.1 #define \_WIN32\_WINNT 0x0400**

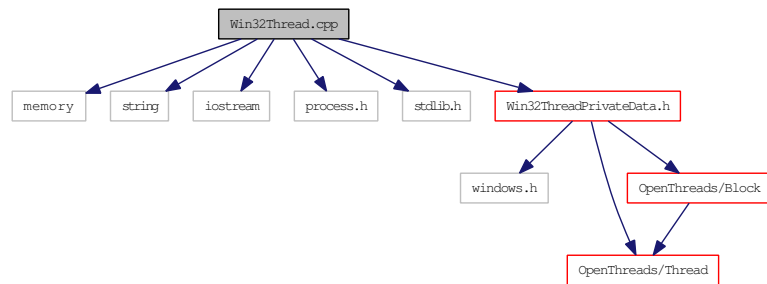
**5.31.1.2 #define USE\_CRITICAL\_SECTION**

**5.31.1.3 #define WIN32\_LEAN\_AND\_MEAN**

## 5.32 Win32Thread.cpp File Reference

```
#include <memory>
#include <string>
#include <iostream>
#include <process.h>
#include <stdlib.h>
#include "Win32ThreadPrivateData.h"
```

Include dependency graph for Win32Thread.cpp:



### Classes

- class [ThreadPrivateActions](#)
- struct [Win32ThreadCanceled](#)

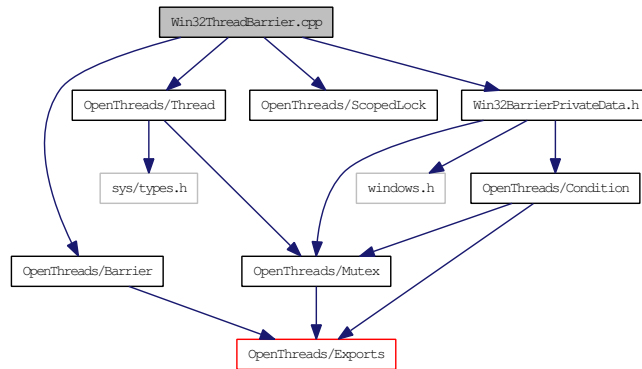
### Namespaces

- namespace [OpenThreads](#)

### 5.33 Win32ThreadBarrier.cpp File Reference

```
#include <OpenThreads/Barrier>
#include <OpenThreads/Thread>
#include <OpenThreads/ScopedLock>
#include "Win32BarrierPrivateData.h"
```

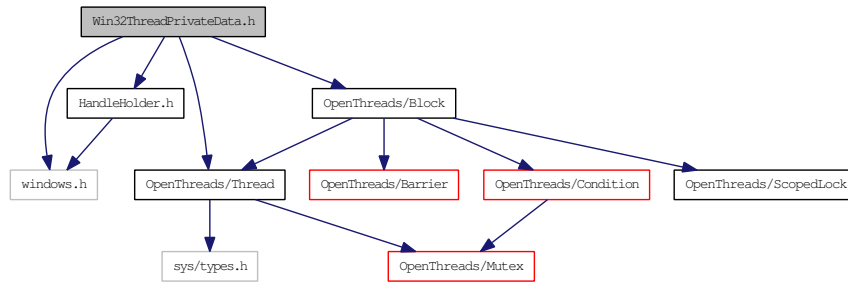
Include dependency graph for Win32ThreadBarrier.cpp:



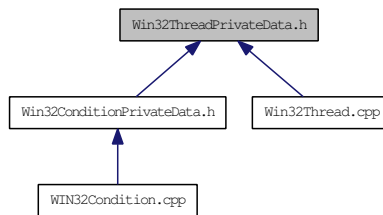
## 5.34 Win32ThreadPrivateData.h File Reference

```
#include <windows.h>
#include <OpenThreads/Thread>
#include <OpenThreads/Block>
#include "HandleHolder.h"
```

Include dependency graph for Win32ThreadPrivateData.h:



This graph shows which files directly or indirectly include this file:



### Classes

- struct [TlsHolder](#)
- class [Win32ThreadPrivateData](#)

### Namespaces

- namespace [OpenThreads](#)

### Defines

- `#define` [\\_WIN32\\_WINNT](#) 0x0400
- `#define` [WIN32\\_LEAN\\_AND\\_MEAN](#)

### Functions

- DWORD [cooperativeWait](#) (HANDLE waitHandle, unsigned long timeout)

#### 5.34.1 Define Documentation

5.34.1.1 `#define` [\\_WIN32\\_WINNT](#) 0x0400

5.34.1.2 `#define` [WIN32\\_LEAN\\_AND\\_MEAN](#)

# Index

---

## - Symbols -

- ~AtomicPtr
  - OpenThreads::AtomicPtr, [16](#)
- ~Barrier
  - OpenThreads::Barrier, [17](#)
- ~Block
  - OpenThreads::Block, [18](#)
- ~BlockCount
  - OpenThreads::BlockCount, [21](#)
- ~Condition
  - OpenThreads::Condition, [23](#)
  - Producer::Condition, [22](#)
- ~HandleHolder
  - OpenThreads::HandleHolder, [24](#)
- ~Mutex
  - OpenThreads::Mutex, [25](#)
- ~ReadWriteMutex
  - OpenThreads::ReadWriteMutex, [31](#)
- ~ReentrantMutex
  - OpenThreads::ReentrantMutex, [33](#)
- ~ReverseScopedLock
  - OpenThreads::ReverseScopedLock, [35](#)
- ~ReverseScopedPointerLock
  - OpenThreads::ReverseScopedPointerLock, [36](#)
- ~ScopedLock
  - OpenThreads::ScopedLock, [37](#)
- ~ScopedPointerLock
  - OpenThreads::ScopedPointerLock, [38](#)
- ~ScopedReadLock
  - OpenThreads::ScopedReadLock, [39](#)
- ~ScopedWriteLock
  - OpenThreads::ScopedWriteLock, [40](#)
- ~Thread
  - OpenThreads::Thread, [49](#)
- ~TlsHolder
  - OpenThreads::Win32ThreadPrivateData::TlsHolder, [54](#)
- ~Win32ConditionPrivateData
  - OpenThreads::Win32ConditionPrivateData, [57](#)
- \_OPENTHREADS\_ATOMIC\_INLINE
  - Atomic, [61](#)
- \_WIN32\_WINNT
  - Win32BarrierPrivateData.h, [87](#)
  - Win32ConditionPrivateData.h, [90](#)
  - Win32MutexPrivateData.h, [92](#)
  - Win32ThreadPrivateData.h, [95](#)
- \_blockCount
  - OpenThreads::BlockCount, [21](#)
- \_cond
  - OpenThreads::Block, [18](#)
  - OpenThreads::BlockCount, [21](#)
  - Producer::Condition, [22](#)
- \_currentCount
  - OpenThreads::BlockCount, [21](#)
- \_mut
  - OpenThreads::Block, [18](#)
  - OpenThreads::BlockCount, [21](#)
- \_mutex
  - OpenThreads::ScopedReadLock, [39](#)
  - OpenThreads::ScopedWriteLock, [40](#)
- \_readCount
  - OpenThreads::ReadWriteMutex, [31](#)
  - \_readCountMutex
    - OpenThreads::ReadWriteMutex, [31](#)
  - \_readWriteMutex
    - OpenThreads::ReadWriteMutex, [31](#)
  - \_released
    - OpenThreads::Block, [18](#)
- A -
  - AND
    - OpenThreads::Atomic, [15](#)
  - assign
    - OpenThreads::AtomicPtr, [16](#)
  - Atomic, [61](#)
    - \_OPENTHREADS\_ATOMIC\_INLINE, [61](#)
    - OpenThreads::Atomic, [15](#)
  - Atomic.cpp, [62](#)
  - AtomicPtr
    - OpenThreads::AtomicPtr, [16](#)
- B -
  - Barrier, [63](#)
    - OpenThreads::Barrier, [17](#)
    - OpenThreads::PThreadBarrierPrivateData, [27](#)
    - OpenThreads::SharedArena, [42](#)
    - OpenThreads::SprocBarrierPrivateData, [43](#)
    - OpenThreads::Win32BarrierPrivateData, [55](#)
  - Block, [64](#)
    - OpenThreads::Block, [18](#)
  - block
    - OpenThreads::Barrier, [17](#)
    - OpenThreads::Block, [18](#)
    - OpenThreads::BlockCount, [21](#)
  - BlockCount
    - OpenThreads::BlockCount, [21](#)
  - broadcast
    - OpenThreads::Condition, [23](#)
    - OpenThreads::Win32ConditionPrivateData, [57](#)
    - Producer::Condition, [22](#)
- C -
  - cancel
    - OpenThreads::Thread, [49](#)
  - cancelCleanup
    - OpenThreads::Thread, [49](#)
  - cancelEvent
    - OpenThreads::Win32ThreadPrivateData, [60](#)
  - completed
    - OpenThreads::BlockCount, [21](#)
  - Condition, [65](#)
    - OpenThreads::Condition, [23](#)
    - OpenThreads::Mutex, [26](#)
    - OpenThreads::PThreadConditionPrivateData, [28](#)
    - OpenThreads::PThreadMutexPrivateData, [29](#)
    - OpenThreads::SemaLink, [41](#)
    - OpenThreads::SharedArena, [42](#)
    - OpenThreads::SprocConditionPrivateData, [44](#)
    - OpenThreads::Win32ConditionPrivateData, [57](#)
    - OpenThreads::Win32MutexPrivateData, [58](#)
    - Producer::Condition, [22](#)

- ConditionDebug
  - OpenThreads::SemaLink, 41
- cooperativeWait
  - OpenThreads, 12
- CurrentThread
  - OpenThreads::Thread, 49
- D -**
- detach
  - OpenThreads::Thread, 49
- E -**
- exchange
  - OpenThreads::Atomic, 15
- Exports, 66
  - OPENTHREAD\_EXPORT\_DIRECTIVE, 66
- G -**
- get
  - OpenThreads::AtomicPtr, 16
  - OpenThreads::HandleHolder, 24
- getBlockCount
  - OpenThreads::BlockCount, 21
- GetConcurrency
  - OpenThreads::Thread, 50
- getCurrentCount
  - OpenThreads::BlockCount, 21
- getId
  - OpenThreads::Win32ThreadPrivateData::TlsHolder, 54
- getImplementation
  - OpenThreads::Thread, 50
- GetMasterPriority
  - OpenThreads::Thread, 50
- GetNumberOfProcessors
  - OpenThreads, 12
- getProcessId
  - OpenThreads::Thread, 50
- getSchedulePolicy
  - OpenThreads::Thread, 50
- getSchedulePriority
  - OpenThreads::Thread, 50
- getStackSize
  - OpenThreads::Thread, 50
- getThreadId
  - OpenThreads::Thread, 50
- H -**
- HandleHolder
  - OpenThreads::HandleHolder, 24
- HandleHolder.h, 67
  - WIN32\_LEAN\_AND\_MEAN, 67
- I -**
- include/ Directory Reference, 4
- include/OpenThreads/ Directory Reference, 6
- Init
  - OpenThreads::Thread, 50
- InterlockedGet
  - Win32ConditionPrivateData.h, 90
- invalidate
  - OpenThreads::Barrier, 17
- isRunning
  - OpenThreads::Thread, 50
- J -**
- join
  - OpenThreads::Thread, 50
- L -**
- lock
  - OpenThreads::Mutex, 25
  - OpenThreads::ReentrantMutex, 33
- M -**
- mainpage.h, 68
- microSleep
  - OpenThreads::Thread, 50
- Mutex, 69
  - OpenThreads::Mutex, 25
  - OpenThreads::PThreadMutexPrivateData, 29
  - OpenThreads::SharedArena, 42
  - OpenThreads::SprocMutexPrivateData, 45
  - OpenThreads::Win32MutexPrivateData, 58
- N -**
- numThreadsCurrentlyBlocked
  - OpenThreads::Barrier, 17
- O -**
- OPENTHREAD\_EXPORT\_DIRECTIVE
  - Exports, 66
- OpenThreads, 11
  - cooperativeWait, 12
  - GetNumberOfProcessors, 12
  - SetProcessorAffinityOfCurrentThread, 12
- OpenThreads::Atomic, 15
  - AND, 15
  - Atomic, 15
  - exchange, 15
  - operator unsigned, 15
  - operator++, 15
  - operator--, 15
  - OR, 15
  - XOR, 15
- OpenThreads::AtomicPtr, 16
  - ~AtomicPtr, 16
  - assign, 16
  - AtomicPtr, 16
  - get, 16
- OpenThreads::Barrier, 17
  - ~Barrier, 17
  - Barrier, 17
  - block, 17
  - invalidate, 17
  - numThreadsCurrentlyBlocked, 17
  - release, 17
  - reset, 17
- OpenThreads::Block, 18
  - ~Block, 18
  - \_cond, 18
  - \_mut, 18
  - \_released, 18
  - Block, 18
  - block, 18
  - release, 18
  - reset, 18
  - set, 18

- OpenThreads::BlockCount, 20
  - ~BlockCount, 21
  - \_blockCount, 21
  - \_cond, 21
  - \_currentCount, 21
  - \_mut, 21
  - block, 21
  - BlockCount, 21
  - completed, 21
  - getBlockCount, 21
  - getCurrentCount, 21
  - release, 21
  - reset, 21
  - setBlockCount, 21
- OpenThreads::Condition, 23
  - ~Condition, 23
  - broadcast, 23
  - Condition, 23
  - signal, 23
  - wait, 23
- OpenThreads::HandleHolder, 24
  - ~HandleHolder, 24
  - get, 24
  - HandleHolder, 24
  - operator bool, 24
  - set, 24
- OpenThreads::Mutex, 25
  - ~Mutex, 25
  - Condition, 26
  - lock, 25
  - Mutex, 25
  - trylock, 25
  - unlock, 26
- OpenThreads::PThreadBarrierPrivateData, 27
  - Barrier, 27
- OpenThreads::PThreadConditionPrivateData, 28
  - Condition, 28
- OpenThreads::PThreadMutexPrivateData, 29
  - Condition, 29
  - Mutex, 29
- OpenThreads::PThreadPrivateData, 30
  - Thread, 30
  - ThreadPrivateActions, 30
- OpenThreads::ReadWriteMutex, 31
  - ~ReadWriteMutex, 31
  - \_readCount, 31
  - \_readCountMutex, 31
  - \_readWriteMutex, 31
  - operator=, 31
  - readLock, 31
  - readUnlock, 31
  - ReadWriteMutex, 31
  - writeLock, 31
  - writeUnlock, 31
- OpenThreads::ReentrantMutex, 33
  - ~ReentrantMutex, 33
  - lock, 33
  - ReentrantMutex, 33
  - trylock, 33
  - unlock, 33
- OpenThreads::ReverseScopedLock, 35
  - ~ReverseScopedLock, 35
  - ReverseScopedLock, 35
- OpenThreads::ReverseScopedPointerLock, 36
  - ~ReverseScopedPointerLock, 36
  - ReverseScopedPointerLock, 36
- OpenThreads::ScopedLock, 37
  - ~ScopedLock, 37
  - ScopedLock, 37
- OpenThreads::ScopedPointerLock, 38
  - ~ScopedPointerLock, 38
  - ScopedPointerLock, 38
- OpenThreads::ScopedReadLock, 39
  - ~ScopedReadLock, 39
  - \_mutex, 39
  - operator=, 39
  - ScopedReadLock, 39
- OpenThreads::ScopedWriteLock, 40
  - ~ScopedWriteLock, 40
  - \_mutex, 40
  - operator=, 40
  - ScopedWriteLock, 40
- OpenThreads::SemaLink, 41
  - Condition, 41
  - ConditionDebug, 41
  - SprocConditionPrivatedata, 41
- OpenThreads::SharedArena, 42
  - Barrier, 42
  - Condition, 42
  - Mutex, 42
- OpenThreads::SprocBarrierPrivateData, 43
  - Barrier, 43
- OpenThreads::SprocConditionPrivateData, 44
  - Condition, 44
- OpenThreads::SprocMutexPrivateData, 45
  - Mutex, 45
  - SprocThreadPrivateActions, 45
- OpenThreads::SprocThreadPrivateData, 46
  - Thread, 46
  - ThreadPrivateActions, 46
- OpenThreads::Thread, 47
  - ~Thread, 49
  - cancel, 49
  - cancelCleanup, 49
  - CurrentThread, 49
  - detach, 49
  - GetConcurrency, 50
  - getImplementation, 50
  - GetMasterPriority, 50
  - getProcessId, 50
  - getSchedulePolicy, 50
  - getSchedulePriority, 50
  - getStackSize, 50
  - getThreadId, 50
  - Init, 50
  - isRunning, 50
  - join, 50
  - microSleep, 50
  - printSchedulingInfo, 50
  - run, 50
  - setCancelModeAsynchronous, 51
  - setCancelModeDeferred, 51
  - setCancelModeDisable, 51
  - SetConcurrency, 51
  - setProcessorAffinity, 51
  - setSchedulePolicy, 51
  - setSchedulePriority, 51
  - setStackSize, 51
  - start, 51
  - startThread, 52
  - testCancel, 52
  - Thread, 49

- THREAD\_PRIORITY\_DEFAULT, 49
- THREAD\_PRIORITY\_HIGH, 49
- THREAD\_PRIORITY\_LOW, 49
- THREAD\_PRIORITY\_MAX, 49
- THREAD\_PRIORITY\_MIN, 49
- THREAD\_PRIORITY\_NOMINAL, 49
- THREAD\_SCHEDULE\_DEFAULT, 49
- THREAD\_SCHEDULE\_FIFO, 49
- THREAD\_SCHEDULE\_ROUND\_ROBIN, 49
- THREAD\_SCHEDULE\_TIME\_SHARE, 49
- ThreadPolicy, 49
- ThreadPriority, 49
- ThreadPrivateActions, 52
- YieldCurrentThread, 52
- OpenThreads::ThreadPrivateActions, 53
  - PopCancelFunction, 53
  - PushCancelFunction, 53
  - Thread, 53
  - ThreadCancelTest, 53
- OpenThreads::Win32BarrierPrivateData, 55
  - Barrier, 55
- OpenThreads::Win32ConditionPrivateData, 56
  - ~Win32ConditionPrivateData, 57
  - broadcast, 57
  - Condition, 57
  - sema\_, 57
  - signal, 57
  - wait, 57
  - waiters\_, 57
  - waiters\_done\_, 57
  - was\_broadcast\_, 57
  - Win32ConditionPrivateData, 57
- OpenThreads::Win32MutexPrivateData, 58
  - Condition, 58
  - Mutex, 58
- OpenThreads::Win32ThreadPrivateData, 60
  - cancelEvent, 60
  - Thread, 60
  - ThreadPrivateActions, 60
  - TLS, 60
- OpenThreads::Win32ThreadPrivateData::TlsHolder, 54
  - ~TlsHolder, 54
  - getId, 54
  - TlsHolder, 54
- OPENTHREADS\_MAJOR\_VERSION
  - Version, 84
- OPENTHREADS\_MINOR\_VERSION
  - Version, 84
- OPENTHREADS\_PATCH\_VERSION
  - Version, 84
- OPENTHREADS\_SOVERSION
  - Version, 84
- OPENTHREADS\_VERSION
  - Version, 84
- OpenThreadsGetLibraryName
  - Version, 84
  - Version.cpp, 86
- OpenThreadsGetSOVersion
  - Version, 84
  - Version.cpp, 86
- OpenThreadsGetVersion
  - Version, 85
  - Version.cpp, 86
- operator bool
  - OpenThreads::HandleHolder, 24
- operator unsigned
  - OpenThreads::Atomic, 15
- operator++
  - OpenThreads::Atomic, 15
- operator--
  - OpenThreads::Atomic, 15
- operator=
  - OpenThreads::ReadWriteMutex, 31
  - OpenThreads::ScopedReadLock, 39
  - OpenThreads::ScopedWriteLock, 40
- OR
  - OpenThreads::Atomic, 15
- OT\_USESHAREDONLY
  - SharedArena.h, 77
- P -**
- PopCancelFunction
  - OpenThreads::ThreadPrivateActions, 53
- printSchedulingInfo
  - OpenThreads::Thread, 50
- Producer, 13
- Producer::Condition, 22
  - ~Condition, 22
  - \_cond, 22
  - broadcast, 22
  - Condition, 22
  - wait, 22
- PRODUCER\_CONDITION
  - Win32Condition.h, 89
- PThreadBarrierPrivateData.h, 70
- PThreadConditionPrivateData.h, 71
- PThreadMutexPrivateData.h, 72
- PThreadPrivateData.h, 73
- PushCancelFunction
  - OpenThreads::ThreadPrivateActions, 53
- R -**
- readLock
  - OpenThreads::ReadWriteMutex, 31
- readUnlock
  - OpenThreads::ReadWriteMutex, 31
- ReadWriteMutex, 74
  - OpenThreads::ReadWriteMutex, 31
- ReentrantMutex, 75
  - OpenThreads::ReentrantMutex, 33
- release
  - OpenThreads::Barrier, 17
  - OpenThreads::Block, 18
  - OpenThreads::BlockCount, 21
- reset
  - OpenThreads::Barrier, 17
  - OpenThreads::Block, 18
  - OpenThreads::BlockCount, 21
- ReverseScopedLock
  - OpenThreads::ReverseScopedLock, 35
- ReverseScopedPointerLock
  - OpenThreads::ReverseScopedPointerLock, 36
- run
  - OpenThreads::Thread, 50
- S -**
- ScopedLock, 76
  - OpenThreads::ScopedLock, 37
- ScopedPointerLock
  - OpenThreads::ScopedPointerLock, 38
- ScopedReadLock

- OpenThreads::ScopedReadLock, 39
- ScopedWriteLock
  - OpenThreads::ScopedWriteLock, 40
- sema\_
  - OpenThreads::Win32ConditionPrivateData, 57
- set
  - OpenThreads::Block, 18
  - OpenThreads::HandleHolder, 24
- setBlockCount
  - OpenThreads::BlockCount, 21
- setCancelModeAsynchronous
  - OpenThreads::Thread, 51
- setCancelModeDeferred
  - OpenThreads::Thread, 51
- setCancelModeDisable
  - OpenThreads::Thread, 51
- SetConcurrency
  - OpenThreads::Thread, 51
- setProcessorAffinity
  - OpenThreads::Thread, 51
- SetProcessorAffinityOfCurrentThread
  - OpenThreads, 12
- setSchedulePolicy
  - OpenThreads::Thread, 51
- setSchedulePriority
  - OpenThreads::Thread, 51
- setStackSize
  - OpenThreads::Thread, 51
- SharedArena.h, 77
  - OT\_USESHAREDONLY, 77
- signal
  - OpenThreads::Condition, 23
  - OpenThreads::Win32ConditionPrivateData, 57
- SprocBarrierPrivateData.h, 78
- SprocConditionPrivatedata
  - OpenThreads::SemaLink, 41
- SprocConditionPrivateData.h, 79
- SprocMutexPrivateData.h, 80
- SprocThreadPrivateActions
  - OpenThreads::SprocMutexPrivateData, 45
- SprocThreadPrivateActions.h, 81
- SprocThreadPrivateData.h, 82
- src/ Directory Reference, 9
- src/OpenThreads/ Directory Reference, 5
- src/OpenThreads/common/ Directory Reference, 3
- src/OpenThreads/pthreads/ Directory Reference, 7
- src/OpenThreads/sproc/ Directory Reference, 8
- src/OpenThreads/win32/ Directory Reference, 10
- start
  - OpenThreads::Thread, 51
- startThread
  - OpenThreads::Thread, 52
- T -**
- testCancel
  - OpenThreads::Thread, 52
- Thread, 83
  - OpenThreads::PThreadPrivateData, 30
  - OpenThreads::SprocThreadPrivateData, 46
  - OpenThreads::Thread, 49
  - OpenThreads::ThreadPrivateActions, 53
  - OpenThreads::Win32ThreadPrivateData, 60
- THREAD\_PRIORITY\_DEFAULT
  - OpenThreads::Thread, 49
- THREAD\_PRIORITY\_HIGH
  - OpenThreads::Thread, 49
- THREAD\_PRIORITY\_LOW
  - OpenThreads::Thread, 49
- THREAD\_PRIORITY\_MAX
  - OpenThreads::Thread, 49
- THREAD\_PRIORITY\_MIN
  - OpenThreads::Thread, 49
- THREAD\_PRIORITY\_NOMINAL
  - OpenThreads::Thread, 49
- THREAD\_SCHEDULE\_DEFAULT
  - OpenThreads::Thread, 49
- THREAD\_SCHEDULE\_FIFO
  - OpenThreads::Thread, 49
- THREAD\_SCHEDULE\_ROUND\_ROBIN
  - OpenThreads::Thread, 49
- THREAD\_SCHEDULE\_TIME\_SHARE
  - OpenThreads::Thread, 49
- ThreadCancelTest
  - OpenThreads::ThreadPrivateActions, 53
- ThreadPolicy
  - OpenThreads::Thread, 49
- ThreadPriority
  - OpenThreads::Thread, 49
- ThreadPrivateActions
  - OpenThreads::PThreadPrivateData, 30
  - OpenThreads::SprocThreadPrivateData, 46
  - OpenThreads::Thread, 52
  - OpenThreads::Win32ThreadPrivateData, 60
- TLS
  - OpenThreads::Win32ThreadPrivateData, 60
- TlsHolder
  - OpenThreads::Win32ThreadPrivateData::TlsHolder, 54
- trylock
  - OpenThreads::Mutex, 25
  - OpenThreads::ReentrantMutex, 33
- U -**
- unlock
  - OpenThreads::Mutex, 26
  - OpenThreads::ReentrantMutex, 33
- USE\_CRITICAL\_SECTION
  - Win32MutexPrivateData.h, 92
- V -**
- Version, 84
  - OPENTHREADS\_MAJOR\_VERSION, 84
  - OPENTHREADS\_MINOR\_VERSION, 84
  - OPENTHREADS\_PATCH\_VERSION, 84
  - OPENTHREADS\_SOVERSION, 84
  - OPENTHREADS\_VERSION, 84
  - OpenThreadsGetLibraryName, 84
  - OpenThreadsGetSOVersion, 84
  - OpenThreadsGetVersion, 85
- Version.cpp, 86
  - OpenThreadsGetLibraryName, 86
  - OpenThreadsGetSOVersion, 86
  - OpenThreadsGetVersion, 86
- W -**
- wait
  - OpenThreads::Condition, 23
  - OpenThreads::Win32ConditionPrivateData, 57
  - Producer::Condition, 22
- waiters\_
  - OpenThreads::Win32ConditionPrivateData, 57
- waiters\_done\_
  - OpenThreads::Win32ConditionPrivateData, 57

- OpenThreads::Win32ConditionPrivateData, [57](#)
- was\_broadcast\_
  - OpenThreads::Win32ConditionPrivateData, [57](#)
- WIN32\_LEAN\_AND\_MEAN
  - HandleHolder.h, [67](#)
  - Win32BarrierPrivateData.h, [87](#)
  - Win32ConditionPrivateData.h, [90](#)
  - Win32MutexPrivateData.h, [92](#)
  - Win32ThreadPrivateData.h, [95](#)
- Win32BarrierPrivateData.h, [87](#)
  - \_WIN32\_WINNT, [87](#)
  - WIN32\_LEAN\_AND\_MEAN, [87](#)
- Win32Condition.cpp, [88](#)
- Win32Condition.h, [89](#)
  - PRODUCER\_CONDITION, [89](#)
- Win32ConditionPrivateData
  - OpenThreads::Win32ConditionPrivateData, [57](#)
- Win32ConditionPrivateData.h, [90](#)
  - \_WIN32\_WINNT, [90](#)
  - InterlockedGet, [90](#)
  - WIN32\_LEAN\_AND\_MEAN, [90](#)
- Win32Mutex.cpp, [91](#)
- Win32MutexPrivateData.h, [92](#)
  - \_WIN32\_WINNT, [92](#)
  - USE\_CRITICAL\_SECTION, [92](#)
  - WIN32\_LEAN\_AND\_MEAN, [92](#)
- Win32Thread.cpp, [93](#)
- Win32ThreadBarrier.cpp, [94](#)
- Win32ThreadCanceled, [59](#)
- Win32ThreadPrivateData.h, [95](#)
  - \_WIN32\_WINNT, [95](#)
  - WIN32\_LEAN\_AND\_MEAN, [95](#)
- writeLock
  - OpenThreads::ReadWriteMutex, [31](#)
- writeUnlock
  - OpenThreads::ReadWriteMutex, [31](#)
- X -**
- XOR
  - OpenThreads::Atomic, [15](#)
- Y -**
- YieldCurrentThread
  - OpenThreads::Thread, [52](#)