Not logged in

register log in

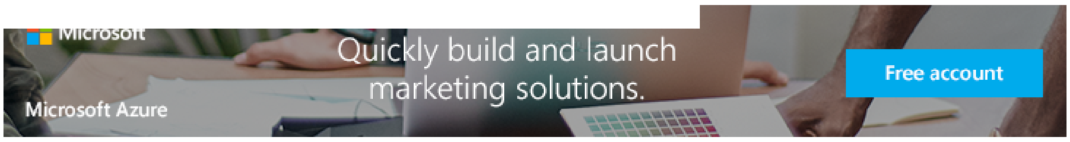
- Information
- Tutorials
- Reference
- Articles
- Forum

Reference

- *C library:*
- *Containers:*
- *Input/Output:*
- *Multi-threading:*
  - <atomic>
  - <condition_variable>
  - <future>
  - <mutex>
  - <thread>
- *Other:*

<atomic>

- *classes:*
  - atomic
  - atomic_flag
- *enum types:*
  - memory_order
- *functions:*
  - atomic_signal_fence
  - atomic_thread_fence
  - kill_dependency
- *initialization macros:*
  - ATOMIC_FLAG_INIT
  - ATOMIC_VAR_INIT
- *functions (C-style atomics):*
  - atomic_compare_exchange_strong
  - atomic_compare_exchange_strong_explicit
  - atomic_compare_exchange_weak
  - atomic_compare_exchange_weak_explicit
  - atomic_exchange
  - atomic_exchange_explicit
  - atomic_fetch_add
  - atomic_fetch_add_explicit
  - atomic_fetch_and
  - atomic_fetch_and_explicit
  - atomic_fetch_or
  - atomic_fetch_or_explicit
  - atomic_fetch_sub
  - atomic_fetch_sub_explicit
  - atomic_fetch_xor
  - atomic_fetch_xor_explicit
  - atomic_flag_clear
  - atomic_flag_clear_explicit
  - atomic_flag_test_and_set
  - atomic_flag_test_and_set_explicit
  - atomic_init
  - atomic_is_lock_free
  - atomic_load
  - atomic_load_explicit
  - atomic_store



header

<atomic>

Atomic

Atomic types are types that encapsulate a value whose access is guaranteed to not cause data races and can be used to synchronize memory accesses among different threads.

This header declares two C++ classes, `atomic` and `atomic_flag`, that implement all the features of atomic types in self-contained classes. The header also declares an entire set of *C-style* types and functions compatible with the atomic support in C.

Classes

atomic	Atomic (class template)
atomic_flag	Atomic flag (class)

Types

memory_order	Memory order (enum)

C-style atomic types

The following *atomic types* are also defined in this header; each with the same behavior as the respective instantiation of `atomic` for the listed *contained type*.

contained type	atomic type	description
`bool`	`atomic_bool`	
`char`	`atomic_char`	atomics for *fundamental integral types*. These are either typedefs of the corresponding full specialization of the `atomic` class template or a base class of such specialization.
`signed char`	`atomic_schar`	
`unsigned char`	`atomic_uchar`	
`short`	`atomic_short`	
`unsigned short`	`atomic_ushort`	
`int`	`atomic_int`	
`unsigned int`	`atomic_uint`	
`long`	`atomic_long`	
`unsigned long`	`atomic_ulong`	
`long long`	`atomic_llong`	
`unsigned long long`	`atomic_ullong`	
`wchar_t`	`atomic_wchar_t`	
`char16_t`	`atomic_char16_t`	
`char32_t`	`atomic_char32_t`	
`intmax_t`	`atomic_intmax_t`	atomics for *width-based integrals* (those defined in `<cinttypes>`). Each of these is either an alias of one of the above *atomics for fundamental integral types* or of a full specialization of the `atomic` class template with an *extended integral type*. Where *N* is one in 8, 16, 32, 64, or any other type width supported by the library.
`uintmax_t`	`atomic_uintmax_t`	
`int_least`*N*`_t`	`atomic_int_least`*N*`_t`	
`uint_least`*N*`_t`	`atomic_uint_least`*N*`_t`	
`int_fast`*N*`_t`	`atomic_int_fast`*N*`_t`	
`uint_fast`*N*`_t`	`atomic_uint_fast`*N*`_t`	
`intptr_t`	`atomic_intptr_t`	
`uintptr_t`	`atomic_uintptr_t`	
`size_t`	`atomic_size_t`	
`ptrdiff_t`	`atomic_ptrdiff_t`	

Functions

kill_dependency	Kill dependency (function)
atomic_thread_fence	Thread fence (function)
atomic_signal_fence	Signal fence (function)

Functions for atomic objects (C-style)

atomic_is_lock_free	Is lock-free (function)
atomic_init	Initialize atomic object (function)
atomic_store	Modify contained value (function)
atomic_store_explicit	Modify contained value (explicit memory order) (function)
atomic_load	Read contained value (function)
atomic_load_explicit	Read contained value (explicit memory order) (function)
atomic_exchange	Read and modify contained value (function)
atomic_exchange_explicit	Read and modify contained value (explicit memory order) (function)
atomic_compare_exchange_weak	Compare and exchange contained value (weak) (function)
atomic_compare_exchange_weak_explicit	Compare and exchange contained value (weak, explicit) (function)
atomic_compare_exchange_strong	Compare and exchange contained value (strong) (function)
atomic_compare_exchange_strong_explicit	Compare and exchange contained value (strong, explicit) (function)
atomic_fetch_add	Add to contained value (function)

atomic_fetch_add_explicit	Add to contained value (explicit memory order) (function)
atomic_fetch_sub	Subtract from contained value (function)
atomic_fetch_sub_explicit	Subtract from contained value (explicit memory order) (function)
atomic_fetch_and	Apply bitwise AND to contained value (function)
atomic_fetch_and_explicit	Apply bitwise AND to contained value (explicit memory order) (function)
atomic_fetch_or	Apply bitwise OR to contained value (function)
atomic_fetch_or_explicit	Apply bitwise OR to contained value (explicit memory order) (function)
atomic_fetch_xor	Apply bitwise XOR to contained value (function)
atomic_fetch_xor_explicit	Apply bitwise XOR to contained value (explicit memory order) (function)

Functions for atomic flags (C-style)

atomic_flag_test_and_set	Test and set atomic flag (function)
atomic_flag_test_and_set_explicit	Test and set atomic flag (explicit memory order) (function)
atomic_flag_clear	Clear atomic flag (function)
atomic_flag_clear_explicit	Clear atomic flag (explicit memory order) (function)

Macro functions

ATOMIC_VAR_INIT	Initialization of atomic variable (macro)
ATOMIC_FLAG_INIT	Initialization of atomic flag (macro)

Macro constants

macro	relative to types	defined as
`ATOMIC_BOOL_LOCK_FREE`	`bool`	`0` if the types are never lock-free. `1` it the types are sometimes lock-free. `2` if the types are always lock-free. Consistent with the value returned by `atomic::is_lock_free`.

`ATOMIC_CHAR_LOCK_FREE`	`char`
	`signed char`
	`unsigned char`	

`ATOMIC_SHORT_LOCK_FREE`	`short`
	`unsigned short`	

`ATOMIC_INT_LOCK_FREE`	`int`
	`unsigned int`	

`ATOMIC_LONG_LOCK_FREE`	`long`
	`unsigned long`	

`ATOMIC_LLONG_LOCK_FREE`	`long long`
	`unsigned long long`	

`ATOMIC_WCHAR_T_LOCK_FREE`	`wchar_t`	
`ATOMIC_CHAR16_T_LOCK_FREE`	`char16_t`	
`ATOMIC_CHAR32_T_LOCK_FREE`	`char32_t`	
`ATOMIC_POINTER_LOCK_FREE`	`U*` *(for any type U)*	