

[Discover Packages](#) > [Standard library](#) > [sync](#) > [atomic](#) 


atomic

package

standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [1](#) |Imported by: [115,927](#)

Details

 Valid [go.mod](#) file  Redistributable license  Tagged version  Stable version [Learn more](#)

Repository

cs.opensource.google/go/go

Links

 [Report a Vulnerability](#) Documentation 

<> Documentation

Overview

Package atomic provides low-level atomic memory primitives useful for implementing synchronization algorithms.

These functions require great care to be used correctly. Except for special, low-level applications, synchronization is better done with channels or the facilities of the sync package. Share memory by communicating; don't communicate by sharing memory.

The swap operation, implemented by the SwapT functions, is the atomic equivalent of:

```
old = *addr
*addr = new
return old
```

The compare-and-swap operation, implemented by the CompareAndSwapT functions, is the atomic equivalent of:

```
if *addr == old {
    *addr = new
    return true
}
return false
```

The add operation, implemented by the AddT functions, is the atomic equivalent of:

```
*addr += delta
return *addr
```

The load and store operations, implemented by the LoadT and StoreT functions, are the atomic equivalents of "return *addr" and "*addr = val".

In the terminology of the Go memory model, if the effect of an atomic operation A is observed by atomic operation B, then A “synchronizes before” B. Additionally, all the atomic operations executed in a program behave as though executed in some sequentially consistent order. This definition provides the same semantics as C++'s sequentially consistent atomics and Java's volatile variables.

Index

```
func AddInt32(addr *int32, delta int32) (new int32)
func AddInt64(addr *int64, delta int64) (new int64)
func AddUint32(addr *uint32, delta uint32) (new uint32)
func AddUint64(addr *uint64, delta uint64) (new uint64)
func AddUintptr(addr *uintptr, delta uintptr) (new uintptr)
func CompareAndSwapInt32(addr *int32, old, new int32) (swapped bool)
func CompareAndSwapInt64(addr *int64, old, new int64) (swapped bool)
func CompareAndSwapPointer(addr *unsafe.Pointer, old, new unsafe.Pointer) (swapped bool)
func CompareAndSwapUint32(addr *uint32, old, new uint32) (swapped bool)
func CompareAndSwapUint64(addr *uint64, old, new uint64) (swapped bool)
func CompareAndSwapUintptr(addr *uintptr, old, new uintptr) (swapped bool)
func LoadInt32(addr *int32) (val int32)
func LoadInt64(addr *int64) (val int64)
func LoadPointer(addr *unsafe.Pointer) (val unsafe.Pointer)
func LoadUint32(addr *uint32) (val uint32)
func LoadUint64(addr *uint64) (val uint64)
func LoadUintptr(addr *uintptr) (val uintptr)
func StoreInt32(addr *int32, val int32)
func StoreInt64(addr *int64, val int64)
func StorePointer(addr *unsafe.Pointer, val unsafe.Pointer)
func StoreUint32(addr *uint32, val uint32)
func StoreUint64(addr *uint64, val uint64)
func StoreUintptr(addr *uintptr, val uintptr)
func SwapInt32(addr *int32, new int32) (old int32)
func SwapInt64(addr *int64, new int64) (old int64)
func SwapPointer(addr *unsafe.Pointer, new unsafe.Pointer) (old unsafe.Pointer)
func SwapUint32(addr *uint32, new uint32) (old uint32)
func SwapUint64(addr *uint64, new uint64) (old uint64)
func SwapUintptr(addr *uintptr, new uintptr) (old uintptr)
type Bool
    func (x *Bool) CompareAndSwap(old, new bool) (swapped bool)
    func (x *Bool) Load() bool
    func (x *Bool) Store(val bool)
    func (x *Bool) Swap(new bool) (old bool)
```

type Int32

```
func (x *Int32) Add(delta int32) (new int32)
func (x *Int32) CompareAndSwap(old, new int32) (swapped bool)
func (x *Int32) Load() int32
func (x *Int32) Store(val int32)
func (x *Int32) Swap(new int32) (old int32)
```

type Int64

```
func (x *Int64) Add(delta int64) (new int64)
func (x *Int64) CompareAndSwap(old, new int64) (swapped bool)
func (x *Int64) Load() int64
func (x *Int64) Store(val int64)
func (x *Int64) Swap(new int64) (old int64)
```

type Pointer

```
func (x *Pointer[T]) CompareAndSwap(old, new *T) (swapped bool)
func (x *Pointer[T]) Load() *T
func (x *Pointer[T]) Store(val *T)
func (x *Pointer[T]) Swap(new *T) (old *T)
```

type Uint32

```
func (x *Uint32) Add(delta uint32) (new uint32)
func (x *Uint32) CompareAndSwap(old, new uint32) (swapped bool)
func (x *Uint32) Load() uint32
func (x *Uint32) Store(val uint32)
func (x *Uint32) Swap(new uint32) (old uint32)
```

type Uint64

```
func (x *Uint64) Add(delta uint64) (new uint64)
func (x *Uint64) CompareAndSwap(old, new uint64) (swapped bool)
func (x *Uint64) Load() uint64
func (x *Uint64) Store(val uint64)
func (x *Uint64) Swap(new uint64) (old uint64)
```

type Uintptr

```
func (x *Uintptr) Add(delta uintptr) (new uintptr)
func (x *Uintptr) CompareAndSwap(old, new uintptr) (swapped bool)
func (x *Uintptr) Load() uintptr
func (x *Uintptr) Store(val uintptr)
func (x *Uintptr) Swap(new uintptr) (old uintptr)
```

type Value

```
func (v *Value) CompareAndSwap(old, new any) (swapped bool)
func (v *Value) Load() (val any)
func (v *Value) Store(val any)
func (v *Value) Swap(new any) (old any)
```

Bugs

Examples

Value (Config)

Value (ReadMostly)

Constants

This section is empty.

Variables

This section is empty.

Functions

func [AddInt32](#)

```
func AddInt32(addr *int32, delta int32) (new int32)
```

AddInt32 atomically adds delta to *addr and returns the new value. Consider using the more ergonomic and less error-prone [Int32.Add](#) instead.

func [AddInt64](#)

```
func AddInt64(addr *int64, delta int64) (new int64)
```

AddInt64 atomically adds delta to *addr and returns the new value. Consider using the more ergonomic and less error-prone [Int64.Add](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func [AddUint32](#)

```
func AddUint32(addr *uint32, delta uint32) (new uint32)
```

AddUint32 atomically adds delta to *addr and returns the new value. To subtract a signed positive constant value c from x, do AddUint32(&x, ^uint32(c-1)). In particular, to decrement x, do AddUint32(&x, ^uint32(0)). Consider using the more ergonomic and less error-prone [Uint32.Add](#) instead.

func [AddUint64](#)

```
func AddUint64(addr *uint64, delta uint64) (new uint64)
```

AddUint64 atomically adds delta to *addr and returns the new value. To subtract a signed positive constant value c from x, do AddUint64(&x, ^uint64(c-1)). In particular, to decrement x, do AddUint64(&x, ^uint64(0)). Consider using the more ergonomic and less error-prone [Uint64.Add](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func [AddUintptr](#)

```
func AddUintptr(addr *uintptr, delta uintptr) (new uintptr)
```

AddUintptr atomically adds delta to *addr and returns the new value. Consider using the more ergonomic and less error-prone [Uintptr.Add](#) instead.

func CompareAndSwapInt32

```
func CompareAndSwapInt32(addr *int32, old, new int32) (swapped bool)
```

CompareAndSwapInt32 executes the compare-and-swap operation for an int32 value. Consider using the more ergonomic and less error-prone [Int32.CompareAndSwap](#) instead.

func CompareAndSwapInt64

```
func CompareAndSwapInt64(addr *int64, old, new int64) (swapped bool)
```

CompareAndSwapInt64 executes the compare-and-swap operation for an int64 value. Consider using the more ergonomic and less error-prone [Int64.CompareAndSwap](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func CompareAndSwapPointer

```
func CompareAndSwapPointer(addr *unsafe.Pointer, old, new unsafe.Pointer) (swapped bool)
```

CompareAndSwapPointer executes the compare-and-swap operation for a unsafe.Pointer value. Consider using the more ergonomic and less error-prone `[Pointer.CompareAndSwap]` instead.

func CompareAndSwapUint32

```
func CompareAndSwapUint32(addr *uint32, old, new uint32) (swapped bool)
```

CompareAndSwapUint32 executes the compare-and-swap operation for a uint32 value. Consider using the more ergonomic and less error-prone [Uint32.CompareAndSwap](#) instead.

func CompareAndSwapUint64

```
func CompareAndSwapUint64(addr *uint64, old, new uint64) (swapped bool)
```

CompareAndSwapUint64 executes the compare-and-swap operation for a uint64 value. Consider using the more ergonomic and less error-prone [Uint64.CompareAndSwap](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func CompareAndSwapUintptr

```
func CompareAndSwapUintptr(addr *uintptr, old, new uintptr) (swapped bool)
```

CompareAndSwapUintptr executes the compare-and-swap operation for a uintptr value. Consider using the more ergonomic and less error-prone [Uintptr.CompareAndSwap](#) instead.

func LoadInt32

```
func LoadInt32(addr *int32) (val int32)
```

LoadInt32 atomically loads *addr. Consider using the more ergonomic and less error-prone [Int32.Load](#) instead.

func LoadInt64

```
func LoadInt64(addr *int64) (val int64)
```

LoadInt64 atomically loads *addr. Consider using the more ergonomic and less error-prone [Int64.Load](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func LoadPointer

```
func LoadPointer(addr *unsafe.Pointer) (val unsafe.Pointer)
```

LoadPointer atomically loads *addr. Consider using the more ergonomic and less error-prone [Pointer.Load](#) instead.

func LoadUint32

```
func LoadUint32(addr *uint32) (val uint32)
```

LoadUint32 atomically loads *addr. Consider using the more ergonomic and less error-prone [Uint32.Load](#) instead.

func LoadUint64

```
func LoadUint64(addr *uint64) (val uint64)
```

LoadUint64 atomically loads *addr. Consider using the more ergonomic and less error-prone [Uint64.Load](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func LoadUintptr

```
func LoadUintptr(addr *uintptr) (val uintptr)
```

LoadUintptr atomically loads *addr. Consider using the more ergonomic and less error-prone [Uintptr.Load](#) instead.

func StoreInt32

```
func StoreInt32(addr *int32, val int32)
```

StoreInt32 atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Int32.Store](#) instead.

func StoreInt64

```
func StoreInt64(addr *int64, val int64)
```

StoreInt64 atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Int64.Store](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func StorePointer

```
func StorePointer(addr *unsafe.Pointer, val unsafe.Pointer)
```

StorePointer atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Pointer.Store](#) instead.

func StoreUint32

```
func StoreUint32(addr *uint32, val uint32)
```

StoreUint32 atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Uint32.Store](#) instead.

func StoreUint64

```
func StoreUint64(addr *uint64, val uint64)
```

StoreUint64 atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Uint64.Store](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func StoreUintptr

```
func StoreUintptr(addr *uintptr, val uintptr)
```

StoreUintptr atomically stores val into *addr. Consider using the more ergonomic and less error-prone [Uintptr.Store](#) instead.

func SwapInt32

added in go1.2

```
func SwapInt32(addr *int32, new int32) (old int32)
```

SwapInt32 atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [Int32.Swap](#) instead.

func SwapInt64

added in go1.2

```
func SwapInt64(addr *int64, new int64) (old int64)
```

SwapInt64 atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [Int64.Swap](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func [SwapPointer](#)

added in go1.2

```
func SwapPointer(addr *unsafe.Pointer, new unsafe.Pointer) (old unsafe.Pointer)
```

SwapPointer atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [\[Pointer.Swap\]](#) instead.

func [SwapUint32](#)

added in go1.2

```
func SwapUint32(addr *uint32, new uint32) (old uint32)
```

SwapUint32 atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [Uint32.Swap](#) instead.

func [SwapUint64](#)

added in go1.2

```
func SwapUint64(addr *uint64, new uint64) (old uint64)
```

SwapUint64 atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [Uint64.Swap](#) instead (particularly if you target 32-bit platforms; see the bugs section).

func [SwapUintptr](#)

added in go1.2

```
func SwapUintptr(addr *uintptr, new uintptr) (old uintptr)
```

SwapUintptr atomically stores new into *addr and returns the previous *addr value. Consider using the more ergonomic and less error-prone [Uintptr.Swap](#) instead.

Types

type [Bool](#)

added in go1.19

```
type Bool struct {  
    // contains filtered or unexported fields  
}
```

A Bool is an atomic boolean value. The zero value is false.

func (*Bool) [CompareAndSwap](#)

added in go1.19

```
func (x *Bool) CompareAndSwap(old, new bool) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for the boolean value x.

func (*Bool) Load

added in go1.19

```
func (x *Bool) Load() bool
```

Load atomically loads and returns the value stored in x.

func (*Bool) Store

added in go1.19

```
func (x *Bool) Store(val bool)
```

Store atomically stores val into x.

func (*Bool) Swap

added in go1.19

```
func (x *Bool) Swap(new bool) (old bool)
```

Swap atomically stores new into x and returns the previous value.

type Int32

added in go1.19

```
type Int32 struct {  
    // contains filtered or unexported fields  
}
```

An Int32 is an atomic int32. The zero value is zero.

func (*Int32) Add

added in go1.19

```
func (x *Int32) Add(delta int32) (new int32)
```

Add atomically adds delta to x and returns the new value.

func (*Int32) CompareAndSwap

added in go1.19

```
func (x *Int32) CompareAndSwap(old, new int32) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Int32) Load

added in go1.19

```
func (x *Int32) Load() int32
```

Load atomically loads and returns the value stored in x.

func (*Int32) Store

added in go1.19

```
func (x *Int32) Store(val int32)
```

Store atomically stores val into x.

func (*Int32) Swap

added in go1.19

```
func (x *Int32) Swap(new int32) (old int32)
```

Swap atomically stores new into x and returns the previous value.

type Int64

added in go1.19

```
type Int64 struct {  
    // contains filtered or unexported fields  
}
```

An Int64 is an atomic int64. The zero value is zero.

func (*Int64) Add

added in go1.19

```
func (x *Int64) Add(delta int64) (new int64)
```

Add atomically adds delta to x and returns the new value.

func (*Int64) CompareAndSwap

added in go1.19

```
func (x *Int64) CompareAndSwap(old, new int64) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Int64) Load

added in go1.19

```
func (x *Int64) Load() int64
```

Load atomically loads and returns the value stored in x.

func (*Int64) Store

added in go1.19

```
func (x *Int64) Store(val int64)
```

Store atomically stores val into x.

func (*Int64) Swap

added in go1.19

```
func (x *Int64) Swap(new int64) (old int64)
```

Swap atomically stores new into x and returns the previous value.

type Pointer

added in go1.19

```
type Pointer[T any] struct {  
    // contains filtered or unexported fields  
}
```

A Pointer is an atomic pointer of type *T. The zero value is a nil *T.

func (*Pointer[T]) CompareAndSwap

added in go1.19

```
func (x *Pointer[T]) CompareAndSwap(old, new *T) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Pointer[T]) Load

added in go1.19

```
func (x *Pointer[T]) Load() *T
```

Load atomically loads and returns the value stored in x.

func (*Pointer[T]) Store

added in go1.19

```
func (x *Pointer[T]) Store(val *T)
```

Store atomically stores val into x.

func (*Pointer[T]) Swap

added in go1.19

```
func (x *Pointer[T]) Swap(new *T) (old *T)
```

Swap atomically stores new into x and returns the previous value.

type Uint32

added in go1.19

```
type Uint32 struct {  
    // contains filtered or unexported fields  
}
```

An Uint32 is an atomic uint32. The zero value is zero.

func (*Uint32) Add

added in go1.19

```
func (x *Uint32) Add(delta uint32) (new uint32)
```

Add atomically adds delta to x and returns the new value.

func (*Uint32) CompareAndSwap

added in go1.19

```
func (x *Uint32) CompareAndSwap(old, new uint32) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Uint32) Load

added in go1.19

```
func (x *Uint32) Load() uint32
```

Load atomically loads and returns the value stored in x.

func (*Uint32) Store

added in go1.19

```
func (x *Uint32) Store(val uint32)
```

Store atomically stores val into x.

func (*Uint32) Swap

added in go1.19

```
func (x *Uint32) Swap(new uint32) (old uint32)
```

Swap atomically stores new into x and returns the previous value.

type Uint64

added in go1.19

```
type Uint64 struct {  
    // contains filtered or unexported fields  
}
```

An Uint64 is an atomic uint64. The zero value is zero.

func (*Uint64) Add

added in go1.19

```
func (x *Uint64) Add(delta uint64) (new uint64)
```

Add atomically adds delta to x and returns the new value.

func (*Uint64) CompareAndSwap

added in go1.19

```
func (x *Uint64) CompareAndSwap(old, new uint64) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Uint64) Load

added in go1.19

```
func (x *Uint64) Load() uint64
```

Load atomically loads and returns the value stored in x.

func (*Uint64) Store

added in go1.19

```
func (x *Uint64) Store(val uint64)
```

Store atomically stores val into x.

func (*Uint64) Swap

added in go1.19

```
func (x *Uint64) Swap(new uint64) (old uint64)
```

Swap atomically stores new into x and returns the previous value.

type Uintptr

added in go1.19

```
type Uintptr struct {  
    // contains filtered or unexported fields  
}
```

An Uintptr is an atomic uintptr. The zero value is zero.

func (*Uintptr) Add

added in go1.19

```
func (x *Uintptr) Add(delta uintptr) (new uintptr)
```

Add atomically adds delta to x and returns the new value.

func (*Uintptr) CompareAndSwap

added in go1.19

```
func (x *Uintptr) CompareAndSwap(old, new uintptr) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for x.

func (*Uintptr) Load

added in go1.19

```
func (x *Uintptr) Load() uintptr
```

Load atomically loads and returns the value stored in x.

func (*Uintptr) Store

added in go1.19

```
func (x *Uintptr) Store(val uintptr)
```

Store atomically stores val into x.

func (*Uintptr) Swap

added in go1.19

```
func (x *Uintptr) Swap(new uintptr) (old uintptr)
```

Swap atomically stores new into x and returns the previous value.

type **Value**

added in go1.4

```
type Value struct {  
    // contains filtered or unexported fields  
}
```

A Value provides an atomic load and store of a consistently typed value. The zero value for a Value returns nil from Load. Once Store has been called, a Value must not be copied.

A Value must not be copied after first use.

► [Example \(Config\)](#)

► [Example \(ReadMostly\)](#)

func (*Value) **CompareAndSwap**

added in go1.17

```
func (v *Value) CompareAndSwap(old, new any) (swapped bool)
```

CompareAndSwap executes the compare-and-swap operation for the Value.

All calls to CompareAndSwap for a given Value must use values of the same concrete type. CompareAndSwap of an inconsistent type panics, as does CompareAndSwap(old, nil).

func (*Value) **Load**

added in go1.4

```
func (v *Value) Load() (val any)
```

Load returns the value set by the most recent Store. It returns nil if there has been no call to Store for this Value.

func (*Value) **Store**

added in go1.4

```
func (v *Value) Store(val any)
```

Store sets the value of the Value v to val. All calls to Store for a given Value must use values of the same concrete type. Store of an inconsistent type panics, as does Store(nil).

func (*Value) **Swap**

added in go1.17

```
func (v *Value) Swap(new any) (old any)
```

Swap stores new into Value and returns the previous value. It returns nil if the Value is empty.

All calls to Swap for a given Value must use values of the same concrete type. Swap of an inconsistent type panics, as does Swap(nil).

Notes

Bugs

- On 386, the 64-bit functions use instructions unavailable before the Pentium MMX.

On non-Linux ARM, the 64-bit functions use instructions unavailable before the ARMv6k core.

On ARM, 386, and 32-bit MIPS, it is the caller's responsibility to arrange for 64-bit alignment of 64-bit words accessed atomically via the primitive atomic functions (types [Int64](#) and [Uint64](#) are automatically aligned). The first word in an allocated struct, array, or slice; in a global variable; or in a local variable (because the subject of all atomic operations will escape to the heap) can be relied upon to be 64-bit aligned.

Source Files

[View all](#) 

[doc.go](#)

[type.go](#)

[value.go](#)

Why Go

[Use Cases](#)

[Case Studies](#)

Get Started

[Playground](#)

[Tour](#)

[Stack Overflow](#)

[Help](#)

Packages

[Standard Library](#)

[About Go Packages](#)

About

[Download](#)

[Blog](#)

[Issue Tracker](#)

[Release Notes](#)

[Brand Guidelines](#)

[Code of Conduct](#)

Connect

[Twitter](#)

[GitHub](#)

[Slack](#)

[r/golang](#)

[Meetup](#)

[Golang Weekly](#)

[Copyright](#)

[Terms of Service](#)

[Privacy Policy](#)

[Report an Issue](#)



Google