# netip

package · standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 7 | Imported by: 1,180

**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 netip defines an IP address type that's a small value type. Building on that Addr type, the package also defines AddrPort (an IP address and a port), and Prefix (an IP address and a bit length prefix).

Compared to the net.IP type, this package's Addr type takes less memory, is immutable, and is comparable (supports == and being a map key).

### Index

- func (ip Addr) AsSlice() []byte
- func (ip Addr) BitLen() int
- func (ip Addr) Compare(ip2 Addr) int
- func (ip Addr) Is4() bool
- func (ip Addr) Is4In6() bool
- func (ip Addr) Is6() bool
- func (ip Addr) IsGlobalUnicast() bool
- func (ip Addr) IsInterfaceLocalMulticast() bool
- func (ip Addr) IsLinkLocalMulticast() bool
- func (ip Addr) IsLinkLocalUnicast() bool
- func (ip Addr) IsLoopback() bool
- func (ip Addr) IsMulticast() bool
- func (ip Addr) IsPrivate() bool
- func (ip Addr) IsUnspecified() bool
- func (ip Addr) IsValid() bool
- func (ip Addr) Less(ip2 Addr) bool
- func (ip Addr) MarshalBinary() ([]byte, error)
- func (ip Addr) MarshalText() ([]byte, error)
- func (ip Addr) Next() Addr
- func (ip Addr) Prefix(b int) (Prefix, error)
- func (ip Addr) Prev() Addr
- func (ip Addr) String() string
- func (ip Addr) StringExpanded() string
- func (ip Addr) Unmap() Addr
- func (ip *Addr) UnmarshalBinary(b []byte) error
- func (ip *Addr) UnmarshalText(text []byte) error
- func (ip Addr) WithZone(zone string) Addr
- func (ip Addr) Zone() string

type AddrPort

- func AddrPortFrom(ip Addr, port uint16) AddrPort
- func MustParseAddrPort(s string) AddrPort
- func ParseAddrPort(s string) (AddrPort, error)
- func (p AddrPort) Addr() Addr
- func (p AddrPort) AppendTo(b []byte) []byte
- func (p AddrPort) IsValid() bool
- func (p AddrPort) MarshalBinary() ([]byte, error)
- func (p AddrPort) MarshalText() ([]byte, error)
- func (p AddrPort) Port() uint16
- func (p AddrPort) String() string
- func (p *AddrPort) UnmarshalBinary(b []byte) error
- func (p *AddrPort) UnmarshalText(text []byte) error

type Prefix

- func MustParsePrefix(s string) Prefix
- func ParsePrefix(s string) (Prefix, error)
- func PrefixFrom(ip Addr, bits int) Prefix
- func (p Prefix) Addr() Addr

## Constants

This section is empty.

## Variables

This section is empty.

## Functions

This section is empty.

## Types

### type Addr

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

Addr represents an IPv4 or IPv6 address (with or without a scoped addressing zone), similar to net.IP or net.IPAddr.

Unlike net.IP or net.IPAddr, Addr is a comparable value type (it supports == and can be a map key) and is immutable.

The zero Addr is not a valid IP address. Addr{} is distinct from both 0.0.0.0 and ::.

### func AddrFrom16

```
func AddrFrom16(addr [16]byte) Addr
```

AddrFrom16 returns the IPv6 address given by the bytes in addr. An IPv4-mapped IPv6 address is left as an IPv6 address. (Use Unmap to convert them if needed.)

### func AddrFrom4

```
func AddrFrom4(addr [4]byte) Addr
```

AddrFrom4 returns the address of the IPv4 address given by the bytes in addr.

### func AddrFromSlice

```
func AddrFromSlice(slice []byte) (ip Addr, ok bool)
```

AddrFromSlice parses the 4- or 16-byte byte slice as an IPv4 or IPv6 address. Note that a net.IP can be passed directly as the []byte argument. If slice's length is not 4 or 16, AddrFromSlice returns Addr{}, false.

### func IPv4Unspecified

```
func IPv4Unspecified() Addr
```

IPv4Unspecified returns the IPv4 unspecified address "0.0.0.0".

### func IPv6LinkLocalAllNodes

```
func IPv6LinkLocalAllNodes() Addr
```

IPv6LinkLocalAllNodes returns the IPv6 link-local all nodes multicast address ff02::1.

### func IPv6LinkLocalAllRouters

added in go1.20

```
func IPv6LinkLocalAllRouters() Addr
```

IPv6LinkLocalAllRouters returns the IPv6 link-local all routers multicast address ff02::2.

### func IPv6Loopback

added in go1.20

```
func IPv6Loopback() Addr
```

IPv6Loopback returns the IPv6 loopback address ::1.

### func IPv6Unspecified

```
func IPv6Unspecified() Addr
```

IPv6Unspecified returns the IPv6 unspecified address "::".

### func MustParseAddr

```
func MustParseAddr(s string) Addr
```

MustParseAddr calls ParseAddr(s) and panics on error. It is intended for use in tests with hard-coded strings.

### func ParseAddr

```
func ParseAddr(s string) (Addr, error)
```

ParseAddr parses s as an IP address, returning the result. The string s can be in dotted decimal ("192.0.2.1"), IPv6 ("2001:db8::68"), or IPv6 with a scoped addressing zone ("fe80::1cc0:3e8c:119f:c2e1%ens18").

### func (Addr) AppendTo

```
func (ip Addr) AppendTo(b []byte) []byte
```

AppendTo appends a text encoding of ip, as generated by MarshalText, to b and returns the extended buffer.

### func (Addr) As16

```
func (ip Addr) As16() (a16 [16]byte)
```

As16 returns the IP address in its 16-byte representation. IPv4 addresses are returned as IPv4-mapped IPv6 addresses. IPv6 addresses with zones are returned without their zone (use the Zone method to get it). The ip zero value returns all zeroes.

### func (Addr) As4

```
func (ip Addr) As4() (a4 [4]byte)
```

As4 returns an IPv4 or IPv4-in-IPv6 address in its 4-byte representation. If ip is the zero Addr or an IPv6 address, As4 panics. Note that 0.0.0.0 is not the zero Addr.

### func (Addr) AsSlice

```
func (ip Addr) AsSlice() []byte
```

AsSlice returns an IPv4 or IPv6 address in its respective 4-byte or 16-byte representation.

### func (Addr) BitLen

```
func (ip Addr) BitLen() int
```

BitLen returns the number of bits in the IP address: 128 for IPv6, 32 for IPv4, and 0 for the zero Addr.

Note that IPv4-mapped IPv6 addresses are considered IPv6 addresses and therefore have bit length 128.

### func (Addr) Compare

```
func (ip Addr) Compare(ip2 Addr) int
```

Compare returns an integer comparing two IPs. The result will be 0 if ip == ip2, -1 if ip < ip2, and +1 if ip > ip2. The definition of "less than" is the same as the Less method.

### func (Addr) Is4

```
func (ip Addr) Is4() bool
```

Is4 reports whether ip is an IPv4 address.

It returns false for IPv4-mapped IPv6 addresses. See Addr.Unmap.

### func (Addr) Is4In6

```
func (ip Addr) Is4In6() bool
```

Is4In6 reports whether ip is an IPv4-mapped IPv6 address.

### func (Addr) Is6

```
func (ip Addr) Is6() bool
```

Is6 reports whether ip is an IPv6 address, including IPv4-mapped IPv6 addresses.

### func (Addr) IsGlobalUnicast

```
func (ip Addr) IsGlobalUnicast() bool
```

IsGlobalUnicast reports whether ip is a global unicast address.

It returns true for IPv6 addresses which fall outside of the current IANA-allocated 2000::/3 global unicast space, with the exception of the link-local address space. It also returns true even if ip is in the IPv4 private address space or IPv6 unique local address space. It returns false for the zero Addr.

For reference, see RFC 1122, RFC 4291, and RFC 4632.

### func (Addr) IsInterfaceLocalMulticast

```
func (ip Addr) IsInterfaceLocalMulticast() bool
```

IsInterfaceLocalMulticast reports whether ip is an IPv6 interface-local multicast address.

### func (Addr) IsLinkLocalMulticast

```
func (ip Addr) IsLinkLocalMulticast() bool
```

IsLinkLocalMulticast reports whether ip is a link-local multicast address.

### func (Addr) IsLinkLocalUnicast

```
func (ip Addr) IsLinkLocalUnicast() bool
```

IsLinkLocalUnicast reports whether ip is a link-local unicast address.

### func (Addr) IsLoopback

```
func (ip Addr) IsLoopback() bool
```

IsLoopback reports whether ip is a loopback address.

### func (Addr) IsMulticast

```
func (ip Addr) IsMulticast() bool
```

IsMulticast reports whether ip is a multicast address.

### func (Addr) IsPrivate

```
func (ip Addr) IsPrivate() bool
```

IsPrivate reports whether ip is a private address, according to RFC 1918 (IPv4 addresses) and RFC 4193 (IPv6 addresses). That is, it reports whether ip is in 10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16, or fc00::/7. This is the same as net.IP.IsPrivate.

### func (Addr) IsUnspecified

```
func (ip Addr) IsUnspecified() bool
```

IsUnspecified reports whether ip is an unspecified address, either the IPv4 address "0.0.0.0" or the IPv6 address "::".

Note that the zero Addr is not an unspecified address.

### func (Addr) IsValid

```
func (ip Addr) IsValid() bool
```

IsValid reports whether the Addr is an initialized address (not the zero Addr).

Note that "0.0.0.0" and "::" are both valid values.

### func (Addr) Less

```
func (ip Addr) Less(ip2 Addr) bool
```

Less reports whether ip sorts before ip2. IP addresses sort first by length, then their address. IPv6 addresses with zones sort just after the same address without a zone.

### func (Addr) MarshalBinary

```
func (ip Addr) MarshalBinary() ([]byte, error)
```

MarshalBinary implements the encoding.BinaryMarshaler interface. It returns a zero-length slice for the zero Addr, the 4-byte form for an IPv4 address, and the 16-byte form with zone appended for an IPv6 address.

### func (Addr) MarshalText

```
func (ip Addr) MarshalText() ([]byte, error)
```

MarshalText implements the encoding.TextMarshaler interface, The encoding is the same as returned by String, with one exception: If ip is the zero Addr, the encoding is the empty string.

### func (Addr) Next

```
func (ip Addr) Next() Addr
```

Next returns the address following ip. If there is none, it returns the zero Addr.

### func (Addr) Prefix

```
func (ip Addr) Prefix(b int) (Prefix, error)
```

Prefix keeps only the top b bits of IP, producing a Prefix of the specified length. If ip is a zero Addr, Prefix always returns a zero Prefix and a nil error. Otherwise, if bits is less than zero or greater than ip.BitLen(), Prefix returns an error.

### func (Addr) Prev

```
func (ip Addr) Prev() Addr
```

Prev returns the IP before ip. If there is none, it returns the IP zero value.

### func (Addr) String

```
func (ip Addr) String() string
```

String returns the string form of the IP address ip. It returns one of 5 forms:

- "invalid IP", if ip is the zero Addr
- IPv4 dotted decimal ("192.0.2.1")
- IPv6 ("2001:db8::1")
- "::ffff:1.2.3.4" (if Is4In6)

- IPv6 with zone ("fe80:db8::1%eth0")

Note that unlike package net's IP.String method, IPv4-mapped IPv6 addresses format with a "::ffff:" prefix before the dotted quad.

### func (Addr) StringExpanded

```
func (ip Addr) StringExpanded() string
```

StringExpanded is like String but IPv6 addresses are expanded with leading zeroes and no "::" compression. For example, "2001:db8::1" becomes "2001:0db8:0000:0000:0000:0000:0000:0001".

### func (Addr) Unmap

```
func (ip Addr) Unmap() Addr
```

Unmap returns ip with any IPv4-mapped IPv6 address prefix removed.

That is, if ip is an IPv6 address wrapping an IPv4 address, it returns the wrapped IPv4 address. Otherwise it returns ip unmodified.

### func (*Addr) UnmarshalBinary

```
func (ip *Addr) UnmarshalBinary(b []byte) error
```

UnmarshalBinary implements the encoding.BinaryUnmarshaler interface. It expects data in the form generated by MarshalBinary.

### func (*Addr) UnmarshalText

```
func (ip *Addr) UnmarshalText(text []byte) error
```

UnmarshalText implements the encoding.TextUnmarshaler interface. The IP address is expected in a form accepted by ParseAddr.

If text is empty, UnmarshalText sets *ip to the zero Addr and returns no error.

### func (Addr) WithZone

```
func (ip Addr) WithZone(zone string) Addr
```

WithZone returns an IP that's the same as ip but with the provided zone. If zone is empty, the zone is removed. If ip is an IPv4 address, WithZone is a no-op and returns ip unchanged.

### func (Addr) Zone

```
func (ip Addr) Zone() string
```

Zone returns ip's IPv6 scoped addressing zone, if any.

### type AddrPort

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

AddrPort is an IP and a port number.

### func AddrPortFrom

```
func AddrPortFrom(ip Addr, port uint16) AddrPort
```

AddrPortFrom returns an AddrPort with the provided IP and port. It does not allocate.

### func MustParseAddrPort

```
func MustParseAddrPort(s string) AddrPort
```

MustParseAddrPort calls ParseAddrPort(s) and panics on error. It is intended for use in tests with hard-coded strings.

### func ParseAddrPort

```
func ParseAddrPort(s string) (AddrPort, error)
```

ParseAddrPort parses s as an AddrPort.

It doesn't do any name resolution: both the address and the port must be numeric.

### func (AddrPort) Addr

```
func (p AddrPort) Addr() Addr
```

Addr returns p's IP address.

### func (AddrPort) AppendTo

```
func (p AddrPort) AppendTo(b []byte) []byte
```

AppendTo appends a text encoding of p, as generated by MarshalText, to b and returns the extended buffer.

### func (AddrPort) IsValid

```
func (p AddrPort) IsValid() bool
```

IsValid reports whether p.Addr() is valid. All ports are valid, including zero.

### func (AddrPort) MarshalBinary

```
func (p AddrPort) MarshalBinary() ([]byte, error)
```

MarshalBinary implements the encoding.BinaryMarshaler interface. It returns Addr.MarshalBinary with an additional two bytes appended containing the port in little-endian.

### func (AddrPort) MarshalText

```
func (p AddrPort) MarshalText() ([]byte, error)
```

MarshalText implements the encoding.TextMarshaler interface. The encoding is the same as returned by String, with one exception: if p.Addr() is the zero Addr, the encoding is the empty string.

### func (AddrPort) Port

```
func (p AddrPort) Port() uint16
```

Port returns p's port.

### func (AddrPort) String

```
func (p AddrPort) String() string
```

### func (*AddrPort) UnmarshalBinary

```
func (p *AddrPort) UnmarshalBinary(b []byte) error
```

UnmarshalBinary implements the encoding.BinaryUnmarshaler interface. It expects data in the form generated by MarshalBinary.

### func (*AddrPort) UnmarshalText

```
func (p *AddrPort) UnmarshalText(text []byte) error
```

UnmarshalText implements the encoding.TextUnmarshaler interface. The AddrPort is expected in a form generated by MarshalText or accepted by ParseAddrPort.

### type Prefix

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

Prefix is an IP address prefix (CIDR) representing an IP network.

The first Bits() of Addr() are specified. The remaining bits match any address. The range of Bits() is [0,32] for IPv4 or [0,128] for IPv6.

### func MustParsePrefix

```
func MustParsePrefix(s string) Prefix
```

MustParsePrefix calls ParsePrefix(s) and panics on error. It is intended for use in tests with hard-coded strings.

### func ParsePrefix

```
func ParsePrefix(s string) (Prefix, error)
```

ParsePrefix parses s as an IP address prefix. The string can be in the form "192.168.1.0/24" or "2001:db8::/32", the CIDR notation defined in RFC 4632 and RFC 4291. IPv6 zones are not permitted in prefixes, and an error will be returned if a zone is present.

Note that masked address bits are not zeroed. Use Masked for that.

### func PrefixFrom

```
func PrefixFrom(ip Addr, bits int) Prefix
```

PrefixFrom returns a Prefix with the provided IP address and bit prefix length.

It does not allocate. Unlike Addr.Prefix, PrefixFrom does not mask off the host bits of ip.

If bits is less than zero or greater than ip.BitLen, Prefix.Bits will return an invalid value -1.

### func (Prefix) Addr

```
func (p Prefix) Addr() Addr
```

Addr returns p's IP address.

### func (Prefix) AppendTo

```
func (p Prefix) AppendTo(b []byte) []byte
```

AppendTo appends a text encoding of p, as generated by MarshalText, to b and returns the extended buffer.

### func (Prefix) Bits

```
func (p Prefix) Bits() int
```

Bits returns p's prefix length.

It reports -1 if invalid.

### func (Prefix) Contains

```
func (p Prefix) Contains(ip Addr) bool
```

Contains reports whether the network p includes ip.

An IPv4 address will not match an IPv6 prefix. An IPv4-mapped IPv6 address will not match an IPv4 prefix. A zero-value IP will not match any prefix. If ip has an IPv6 zone, Contains returns false, because Prefixes strip zones.

### func (Prefix) IsSingleIP

```
func (p Prefix) IsSingleIP() bool
```

IsSingleIP reports whether p contains exactly one IP.

### func (Prefix) IsValid

```
func (p Prefix) IsValid() bool
```

IsValid reports whether p.Bits() has a valid range for p.Addr(). If p.Addr() is the zero Addr, IsValid returns false. Note that if p is the zero Prefix, then p.IsValid() == false.

### func (Prefix) MarshalBinary

```
func (p Prefix) MarshalBinary() ([]byte, error)
```

MarshalBinary implements the encoding.BinaryMarshaler interface. It returns Addr.MarshalBinary with an additional byte appended containing the prefix bits.

### func (Prefix) MarshalText

```
func (p Prefix) MarshalText() ([]byte, error)
```

MarshalText implements the encoding.TextMarshaler interface, The encoding is the same as returned by String, with one exception: If p is the zero value, the encoding is the empty string.

### func (Prefix) Masked

```
func (p Prefix) Masked() Prefix
```

Masked returns p in its canonical form, with all but the high p.Bits() bits of p.Addr() masked off.

If p is zero or otherwise invalid, Masked returns the zero Prefix.

### func (Prefix) Overlaps

```
func (p Prefix) Overlaps(o Prefix) bool
```

Overlaps reports whether p and o contain any IP addresses in common.

If p and o are of different address families or either have a zero IP, it reports false. Like the Contains method, a prefix with an IPv4-mapped IPv6 address is still treated as an IPv6 mask.

### func (Prefix) String

```
func (p Prefix) String() string
```

String returns the CIDR notation of p: "<ip>/<bits>".

### func (*Prefix) UnmarshalBinary

```
func (p *Prefix) UnmarshalBinary(b []byte) error
```

UnmarshalBinary implements the encoding.BinaryUnmarshaler interface. It expects data in the form generated by MarshalBinary.

### func (*Prefix) UnmarshalText

```
func (p *Prefix) UnmarshalText(text []byte) error
```

UnmarshalText implements the encoding.TextUnmarshaler interface. The IP address is expected in a form accepted by ParsePrefix or generated by MarshalText.

## Source Files

View all

leaf_alts.go netip.go uint128.go