

Discover Packages > Standard library > net > url 

url







package


standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: 6 |

Imported by: 291,496

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 url parses URLs and implements query escaping.

Index

[func JoinPath\(base string, elem ...string\) \(result string, err error\)](#)

[func PathEscape\(s string\) string](#)

[func PathUnescape\(s string\) \(string, error\)](#)

[func QueryEscape\(s string\) string](#)

[func QueryUnescape\(s string\) \(string, error\)](#)

[type Error](#)

[func \(e *Error\) Error\(\) string](#)

[func \(e *Error\) Temporary\(\) bool](#)

[func \(e *Error\) Timeout\(\) bool](#)

[func \(e *Error\) Unwrap\(\) error](#)

[type EscapeError](#)

[func \(e EscapeError\) Error\(\) string](#)

[type InvalidHostError](#)

[func \(e InvalidHostError\) Error\(\) string](#)

[type URL](#)

[func Parse\(rawURL string\) \(*URL, error\)](#)

[func ParseRequestURI\(rawURL string\) \(*URL, error\)](#)

[func \(u *URL\) EscapedFragment\(\) string](#)

[func \(u *URL\) EscapedPath\(\) string](#)

```
func (u *URL) Hostname() string
func (u *URL) IsAbs() bool
func (u *URL) JoinPath(elem ...string) *URL
func (u *URL) MarshalBinary() (text []byte, err error)
func (u *URL) Parse(ref string) (*URL, error)
func (u *URL) Port() string
func (u *URL) Query() Values
func (u *URL) Redacted() string
func (u *URL) RequestURI() string
func (u *URL) ResolveReference(ref *URL) *URL
func (u *URL) String() string
func (u *URL) UnmarshalBinary(text []byte) error
```

type Userinfo

```
func User(username string) *Userinfo
func UserPassword(username, password string) *Userinfo
func (u *Userinfo) Password() (string, bool)
func (u *Userinfo) String() string
func (u *Userinfo) Username() string
```

type Values

```
func ParseQuery(query string) (Values, error)
func (v Values) Add(key, value string)
func (v Values) Del(key string)
func (v Values) Encode() string
func (v Values) Get(key string) string
func (v Values) Has(key string) bool
func (v Values) Set(key, value string)
```

Examples

[ParseQuery](#)

[PathEscape](#)

[PathUnescape](#)

[QueryEscape](#)

[QueryUnescape](#)

[URL](#)

[URL \(Roundtrip\)](#)

[URL.EscapedFragment](#)

[URL.EscapedPath](#)

[URL.Hostname](#)

[URL.IsAbs](#)

[URL.MarshalBinary](#)

[URL.Parse](#)

[URL.Port](#)

[URL.Query](#)

[URL.Redacted](#)

[URL.RequestURI](#)

[URL.ResolveReference](#)

[URL.String](#)
[URL.UnmarshalBinary](#)
[Values](#)
[Values.Add](#)
[Values.Del](#)
[Values.Encode](#)
[Values.Get](#)
[Values.Has](#)
[Values.Set](#)

Constants

This section is empty.

Variables

This section is empty.

Functions

func [JoinPath](#)

added in go1.19

```
func JoinPath(base string, elem ...string) (result string, err error)
```

`JoinPath` returns a URL string with the provided path elements joined to the existing path of `base` and the resulting path cleaned of any `./` or `../` elements.

func [PathEscape](#)

added in go1.8

```
func PathEscape(s string) string
```

`PathEscape` escapes the string so it can be safely placed inside a URL path segment, replacing special characters (including `/`) with `%XX` sequences as needed.

► [Example](#)

func [PathUnescape](#)

added in go1.8

```
func PathUnescape(s string) (string, error)
```

`PathUnescape` does the inverse transformation of `PathEscape`, converting each 3-byte encoded substring of the form `"%AB"` into the hex-decoded byte `0xAB`. It returns an error if any `%` is not followed by two hexadecimal digits.

`PathUnescape` is identical to `QueryUnescape` except that it does not unescape `'+'` to `' '` (space).

► [Example](#)

func QueryEscape

```
func QueryEscape(s string) string
```

QueryEscape escapes the string so it can be safely placed inside a URL query.

► [Example](#)

func QueryUnescape

```
func QueryUnescape(s string) (string, error)
```

QueryUnescape does the inverse transformation of QueryEscape, converting each 3-byte encoded substring of the form "%AB" into the hex-decoded byte 0xAB. It returns an error if any % is not followed by two hexadecimal digits.

► [Example](#)

Types

type Error

```
type Error struct {  
    Op  string  
    URL string  
    Err error  
}
```

Error reports an error and the operation and URL that caused it.

func (*Error) Error

```
func (e *Error) Error() string
```

func (*Error) Temporary

added in go1.6

```
func (e *Error) Temporary() bool
```

func (*Error) Timeout

added in go1.6

```
func (e *Error) Timeout() bool
```

func (*Error) Unwrap

added in go1.13

```
func (e *Error) Unwrap() error
```

type `EscapeError`

```
type EscapeError string
```

func (EscapeError) `Error`

```
func (e EscapeError) Error() string
```

type `InvalidHostError`

added in go1.6

```
type InvalidHostError string
```

func (InvalidHostError) `Error`

added in go1.6

```
func (e InvalidHostError) Error() string
```

type `URL`

```
type URL struct {
    Scheme      string
    Opaque      string    // encoded opaque data
    User        *UserInfo // username and password information
    Host        string    // host or host:port
    Path        string    // path (relative paths may omit leading slash)
    RawPath     string    // encoded path hint (see EscapedPath method)
    OmitHost    bool      // do not emit empty host (authority)
    ForceQuery  bool      // append a query ('?') even if RawQuery is empty
    RawQuery    string    // encoded query values, without '?'
    Fragment    string    // fragment for references, without '#'
    RawFragment string    // encoded fragment hint (see EscapedFragment method)
}
```

A `URL` represents a parsed URL (technically, a URI reference).

The general form represented is:

```
[scheme:][//[userinfo@]host][/]path[?query][#fragment]
```

URLs that do not start with a slash after the scheme are interpreted as:

```
scheme:opaque[?query][#fragment]
```

Note that the `Path` field is stored in decoded form: `/%47%6f%2f` becomes `/Go/`. A consequence is that it is impossible to tell which slashes in the `Path` were slashes in the raw URL and which were `%2f`. This distinction is rarely important, but when it is, the code should use the `EscapedPath` method, which preserves the original encoding of `Path`.

The `RawPath` field is an optional field which is only set when the default encoding of `Path` is different from the escaped path. See the `EscapedPath` method for more details.

URL's `String` method uses the `EscapedPath` method to obtain the path.

► [Example](#)

► [Example \(Roundtrip\)](#)

func `Parse`

```
func Parse(rawURL string) (*URL, error)
```

`Parse` parses a raw url into a `URL` structure.

The url may be relative (a path, without a host) or absolute (starting with a scheme). Trying to parse a hostname and path without a scheme is invalid but may not necessarily return an error, due to parsing ambiguities.

func `ParseRequestURI`

```
func ParseRequestURI(rawURL string) (*URL, error)
```

`ParseRequestURI` parses a raw url into a `URL` structure. It assumes that url was received in an HTTP request, so the url is interpreted only as an absolute URI or an absolute path. The string url is assumed not to have a `#fragment` suffix. (Web browsers strip `#fragment` before sending the URL to a web server.)

func (*URL) `EscapedFragment`

added in go1.15

```
func (u *URL) EscapedFragment() string
```

`EscapedFragment` returns the escaped form of `u.Fragment`. In general there are multiple possible escaped forms of any fragment. `EscapedFragment` returns `u.RawFragment` when it is a valid escaping of `u.Fragment`. Otherwise `EscapedFragment` ignores `u.RawFragment` and computes an escaped form on its own. The `String` method uses `EscapedFragment` to construct its result. In general, code should call `EscapedFragment` instead of reading `u.RawFragment` directly.

► [Example](#)

func (*URL) `EscapedPath`

added in go1.5

```
func (u *URL) EscapedPath() string
```

`EscapedPath` returns the escaped form of `u.Path`. In general there are multiple possible escaped forms of any path. `EscapedPath` returns `u.RawPath` when it is a valid escaping of `u.Path`. Otherwise `EscapedPath` ignores `u.RawPath` and computes an escaped form on its own. The `String` and `RequestURI` methods use

EscapedPath to construct their results. In general, code should call EscapedPath instead of reading u.RawPath directly.

► [Example](#)

func (*URL) Hostname

added in go1.8

```
func (u *URL) Hostname() string
```

Hostname returns u.Host, stripping any valid port number if present.

If the result is enclosed in square brackets, as literal IPv6 addresses are, the square brackets are removed from the result.

► [Example](#)

func (*URL) IsAbs

```
func (u *URL) IsAbs() bool
```

IsAbs reports whether the URL is absolute. Absolute means that it has a non-empty scheme.

► [Example](#)

func (*URL) JoinPath

added in go1.19

```
func (u *URL) JoinPath(elem ...string) *URL
```

JoinPath returns a new URL with the provided path elements joined to any existing path and the resulting path cleaned of any ./ or ../ elements. Any sequences of multiple / characters will be reduced to a single /.

func (*URL) MarshalBinary

added in go1.8

```
func (u *URL) MarshalBinary() (text []byte, err error)
```

► [Example](#)

func (*URL) Parse

```
func (u *URL) Parse(ref string) (*URL, error)
```

Parse parses a URL in the context of the receiver. The provided URL may be relative or absolute. Parse returns nil, err on parse failure, otherwise its return value is the same as ResolveReference.

► [Example](#)

func (*URL) Port

added in go1.8

```
func (u *URL) Port() string
```

Port returns the port part of u.Host, without the leading colon.

If u.Host doesn't contain a valid numeric port, Port returns an empty string.

► [Example](#)

func (*URL) Query

```
func (u *URL) Query() Values
```

Query parses RawQuery and returns the corresponding values. It silently discards malformed value pairs. To check errors use ParseQuery.

► [Example](#)

func (*URL) Redacted

added in go1.15

```
func (u *URL) Redacted() string
```

Redacted is like String but replaces any password with "xxxxx". Only the password in u.URL is redacted.

► [Example](#)

func (*URL) RequestURI

```
func (u *URL) RequestURI() string
```

RequestURI returns the encoded path?query or opaque?query string that would be used in an HTTP request for u.

► [Example](#)

func (*URL) ResolveReference

```
func (u *URL) ResolveReference(ref *URL) *URL
```

ResolveReference resolves a URI reference to an absolute URI from an absolute base URI u, per [RFC 3986 Section 5.2](#). The URI reference may be relative or absolute. ResolveReference always returns a new URL instance, even if the returned URL is identical to either the base or reference. If ref is an absolute URL, then ResolveReference ignores base and returns a copy of ref.

► Example

func (*URL) String

```
func (u *URL) String() string
```

String reassembles the URL into a valid URL string. The general form of the result is one of:

```
scheme:opaque?query#fragment  
scheme://userinfo@host/path?query#fragment
```

If u.Opaque is non-empty, String uses the first form; otherwise it uses the second form. Any non-ASCII characters in host are escaped. To obtain the path, String uses u.EscapedPath().

In the second form, the following rules apply:

- if u.Scheme is empty, scheme: is omitted.
- if u.User is nil, userinfo@ is omitted.
- if u.Host is empty, host/ is omitted.
- if u.Scheme and u.Host are empty and u.User is nil, the entire scheme://userinfo@host/ is omitted.
- if u.Host is non-empty and u.Path begins with a /, the form host/path does not add its own /.
- if u.RawQuery is empty, ?query is omitted.
- if u.Fragment is empty, #fragment is omitted.

► Example

func (*URL) UnmarshalBinary

added in go1.8

```
func (u *URL) UnmarshalBinary(text []byte) error
```

► Example

type Userinfo

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

The Userinfo type is an immutable encapsulation of username and password details for a URL. An existing Userinfo value is guaranteed to have a username set (potentially empty, as allowed by [RFC 2396](#)), and optionally a password.

func User

```
func User(username string) *Userinfo
```

User returns a `Userinfo` containing the provided username and no password set.

func `UserPassword`

```
func UserPassword(username, password string) *Userinfo
```

`UserPassword` returns a `Userinfo` containing the provided username and password.

This functionality should only be used with legacy web sites. [RFC 2396](#) warns that interpreting `Userinfo` this way “is NOT RECOMMENDED, because the passing of authentication information in clear text (such as URI) has proven to be a security risk in almost every case where it has been used.”

func (*`Userinfo`) `Password`

```
func (u *Userinfo) Password() (string, bool)
```

`Password` returns the password in case it is set, and whether it is set.

func (*`Userinfo`) `String`

```
func (u *Userinfo) String() string
```

`String` returns the encoded userinfo information in the standard form of "username[:password]".

func (*`Userinfo`) `Username`

```
func (u *Userinfo) Username() string
```

`Username` returns the username.

type `Values`

```
type Values map[string][]string
```

`Values` maps a string key to a list of values. It is typically used for query parameters and form values. Unlike in the `http.Header` map, the keys in a `Values` map are case-sensitive.

► [Example](#)

func `ParseQuery`

```
func ParseQuery(query string) (Values, error)
```

`ParseQuery` parses the URL-encoded query string and returns a map listing the values specified for each key. `ParseQuery` always returns a non-nil map containing all the valid query parameters found; `err` describes the first decoding error encountered, if any.

Query is expected to be a list of key=value settings separated by ampersands. A setting without an equals sign is interpreted as a key set to an empty value. Settings containing a non-URL-encoded semicolon are considered invalid.

► [Example](#)

func (Values) **Add**

```
func (v Values) Add(key, value string)
```

Add adds the value to key. It appends to any existing values associated with key.

► [Example](#)

func (Values) **Del**

```
func (v Values) Del(key string)
```

Del deletes the values associated with key.

► [Example](#)

func (Values) **Encode**

```
func (v Values) Encode() string
```

Encode encodes the values into “URL encoded” form ("bar=baz&foo=quux") sorted by key.

► [Example](#)

func (Values) **Get**

```
func (v Values) Get(key string) string
```

Get gets the first value associated with the given key. If there are no values associated with the key, Get returns the empty string. To access multiple values, use the map directly.

► [Example](#)

func (Values) **Has**

added in go1.17

```
func (v Values) Has(key string) bool
```

Has checks whether a given key is set.

► [Example](#)

func (Values) [Set](#)

```
func (v Values) Set(key, value string)
```

Set sets the key to value. It replaces any existing values.

► [Example](#)



Source Files

[View all](#) [↗](#)

[url.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](#)

