mail

package | standard library

Version: go1.20.1 Latest | Published: Feb 14, 2023 | License: BSD-3-Clause | Imports: 11 | Imported by: 12,537

Documentation

Overview

Package mail implements parsing of mail messages.

For the most part, this package follows the syntax as specified by RFC 5322 and extended by RFC 6532. Notable divergences:

- Obsolete address formats are not parsed, including addresses with embedded route information.
- The full range of spacing (the CFWS syntax element) is not supported, such as breaking addresses across lines.
- No unicode normalization is performed.
- The special characters ()[]:;@\, are allowed to appear unquoted in names.

Index

func (h Header) Date() (time.Time, error)
func (h Header) Get(key string) string
type Message
func ReadMessage(r io.Reader) (msg *Message, err error)

Examples

ParseAddress
ParseAddressList
ReadMessage

Constants

This section is empty.

Variables

View Source

```
var ErrHeaderNotPresent = errors.New("mail: header not in message")
```

Functions

func ParseDate

added in go1.8

```
func ParseDate(date string) (time.Time, error)
```

ParseDate parses an RFC 5322 date string.

Types

type Address

```
type Address struct {
    Name    string // Proper name; may be empty.
    Address string // user@domain
}
```

Address represents a single mail address. An address such as "Barry Gibbs <bg@example.com>" is represented as Address{Name: "Barry Gibbs", Address: "bg@example.com"}.

func ParseAddress

added in go1.1

```
func ParseAddress(address string) (*Address, error)
```

ParseAddress parses a single RFC 5322 address, e.g. "Barry Gibbs <bg@example.com>"

▶ Example

func ParseAddressList

added in go1.1

```
func ParseAddressList(list string) ([]*Address, error)
```

ParseAddressList parses the given string as a list of addresses.

▶ Example

func (*Address) String

```
func (a *Address) String() string
```

String formats the address as a valid RFC 5322 address. If the address's name contains non-ASCII characters the name will be rendered according to RFC 2047.

type AddressParser

added in go1.5

```
type AddressParser struct {
    // WordDecoder optionally specifies a decoder for RFC 2047 encoded-words.
    WordDecoder *mime.WordDecoder
}
```

An AddressParser is an RFC 5322 address parser.

func (*AddressParser) Parse

added in go1.5

```
func (p *AddressParser) Parse(address string) (*Address, error)
```

Parse parses a single RFC 5322 address of the form "Gogh Fir <gf@example.com>" or "foo@example.com".

func (*AddressParser) ParseList

added in go1.5

```
func (p *AddressParser) ParseList(list string) ([]*Address, error)
```

ParseList parses the given string as a list of comma-separated addresses of the form "Gogh Fir <gf@example.com>" or "foo@example.com".

type Header

```
type Header map[string][]string
```

A Header represents the key-value pairs in a mail message header.

func (Header) AddressList

```
func (h Header) AddressList(key string) ([]*Address, error)
```

AddressList parses the named header field as a list of addresses.

func (Header) Date

```
func (h Header) Date() (time.Time, error)
```

Date parses the Date header field.

func (Header) Get

```
func (h Header) Get(key string) string
```

Get gets the first value associated with the given key. It is case insensitive; CanonicalMIMEHeaderKey is used to canonicalize the provided key. If there are no values associated with the key, Get returns "". To access multiple values of a key, or to use non-canonical keys, access the map directly.

type Message

```
type Message struct {
    Header Header
    Body   io.Reader
}
```

A Message represents a parsed mail message.

func ReadMessage

```
func ReadMessage(r io.Reader) (msg *Message, err error)
```

ReadMessage reads a message from r. The headers are parsed, and the body of the message will be available for reading from msg.Body.

▶ Example

Source Files

View all

message.go