

Discover Packages > Standard library > encoding > asn1 

asn1







package



standard library

Version: [go1.20.1](#) **Latest** | Published: Feb 14, 2023 | License: [BSD-3-Clause](#) | Imports: [12](#) |

Imported by: [9,724](#)

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 `asn1` implements parsing of DER-encoded ASN.1 data structures, as defined in ITU-T Rec X.690.

See also “A Layman's Guide to a Subset of ASN.1, BER, and DER,”

<http://luca.ntop.org/Teaching/Appunti/asn1.html>.

Index

Constants

Variables

`func Marshal(val any) ([]byte, error)`

`func MarshalWithParams(val any, params string) ([]byte, error)`

`func Unmarshal(b []byte, val any) (rest []byte, err error)`

`func UnmarshalWithParams(b []byte, val any, params string) (rest []byte, err error)`

`type BitString`

`func (b BitString) At(i int) int`

`func (b BitString) RightAlign() []byte`

`type Enumerated`

`type Flag`

`type ObjectIdentifier`

`func (oi ObjectIdentifier) Equal(other ObjectIdentifier) bool`

`func (oi ObjectIdentifier) String() string`

`type RawContent`

```
type RawValue
type StructuralError
    func (e StructuralError) Error() string
type SyntaxError
    func (e SyntaxError) Error() string
```

Constants

[View Source](#)

```
const (
    TagBoolean          = 1
    TagInteger          = 2
    TagBitString        = 3
    TagOctetString      = 4
    TagNull             = 5
    TagOID              = 6
    TagEnum             = 10
    TagUTF8String       = 12
    TagSequence         = 16
    TagSet              = 17
    TagNumericString    = 18
    TagPrintableString  = 19
    TagT61String        = 20
    TagIA5String        = 22
    TagUTCTime          = 23
    TagGeneralizedTime  = 24
    TagGeneralString    = 27
    TagBMPString        = 30
)
```

ASN.1 tags represent the type of the following object.

[View Source](#)

```
const (
    ClassUniversal      = 0
    ClassApplication    = 1
    ClassContextSpecific = 2
    ClassPrivate        = 3
)
```

ASN.1 class types represent the namespace of the tag.

Variables

[View Source](#)

```
var NullBytes = []byte{TagNull, 0}
```

NullBytes contains bytes representing the DER-encoded ASN.1 NULL type.

[View Source](#)

```
var NullRawValue = RawValue{Tag: TagNull}
```

NullRawValue is a RawValue with its Tag set to the ASN.1 NULL type tag (5).

Functions

func Marshal

```
func Marshal(val any) ([]byte, error)
```

Marshal returns the ASN.1 encoding of val.

In addition to the struct tags recognised by Unmarshal, the following can be used:

ia5:	causes strings to be marshaled as ASN.1, IA5String values
omitempty:	causes empty slices to be skipped
printable:	causes strings to be marshaled as ASN.1, PrintableString values
utf8:	causes strings to be marshaled as ASN.1, UTF8String values
utc:	causes time.Time to be marshaled as ASN.1, UTCTime values
generalized:	causes time.Time to be marshaled as ASN.1, GeneralizedTime values

func MarshalWithParams

added in go1.10

```
func MarshalWithParams(val any, params string) ([]byte, error)
```

MarshalWithParams allows field parameters to be specified for the top-level element. The form of the params is the same as the field tags.

func Unmarshal

```
func Unmarshal(b []byte, val any) (rest []byte, err error)
```

Unmarshal parses the DER-encoded ASN.1 data structure b and uses the reflect package to fill in an arbitrary value pointed at by val. Because Unmarshal uses the reflect package, the structs being written to must use upper case field names. If val is nil or not a pointer, Unmarshal returns an error.

After parsing b, any bytes that were leftover and not used to fill val will be returned in rest. When parsing a SEQUENCE into a struct, any trailing elements of the SEQUENCE that do not have matching fields in val will not be included in rest, as these are considered valid elements of the SEQUENCE and not trailing data.

An ASN.1 INTEGER can be written to an int, int32, int64, or *big.Int (from the math/big package). If the encoded value does not fit in the Go type, Unmarshal returns a parse error.

An ASN.1 BIT STRING can be written to a BitString.

An ASN.1 OCTET STRING can be written to a []byte.

An ASN.1 OBJECT IDENTIFIER can be written to an ObjectIdentifier.

An ASN.1 ENUMERATED can be written to an Enumerated.

An ASN.1 UTCTIME or GENERALIZEDTIME can be written to a time.Time.

An ASN.1 PrintableString, IA5String, or NumericString can be written to a string.

Any of the above ASN.1 values can be written to an interface{}. The value stored in the interface has the corresponding Go type. For integers, that type is int64.

An ASN.1 SEQUENCE OF x or SET OF x can be written to a slice if an x can be written to the slice's element type.

An ASN.1 SEQUENCE or SET can be written to a struct if each of the elements in the sequence can be written to the corresponding element in the struct.

The following tags on struct fields have special meaning to Unmarshal:

```
application specifies that an APPLICATION tag is used
private       specifies that a PRIVATE tag is used
default:x     sets the default value for optional integer fields (only used if optional
is also present)
explicit      specifies that an additional, explicit tag wraps the implicit one
optional      marks the field as ASN.1 OPTIONAL
set           causes a SET, rather than a SEQUENCE type to be expected
tag:x         specifies the ASN.1 tag number; implies ASN.1 CONTEXT SPECIFIC
```

When decoding an ASN.1 value with an IMPLICIT tag into a string field, Unmarshal will default to a PrintableString, which doesn't support characters such as '@' and '&'. To force other encodings, use the following tags:

```
ia5           causes strings to be unmarshaled as ASN.1 IA5String values
numeric       causes strings to be unmarshaled as ASN.1 NumericString values
utf8          causes strings to be unmarshaled as ASN.1 UTF8String values
```

If the type of the first field of a structure is RawContent then the raw ASN1 contents of the struct will be stored in it.

If the name of a slice type ends with "SET" then it's treated as if the "set" tag was set on it. This results in interpreting the type as a SET OF x rather than a SEQUENCE OF x. This can be used with nested slices where a struct tag cannot be given.

Other ASN.1 types are not supported; if it encounters them, Unmarshal returns a parse error.

func UnmarshalWithParams

```
func UnmarshalWithParams(b []byte, val any, params string) (rest []byte, err error)
```

UnmarshalWithParams allows field parameters to be specified for the top-level element. The form of the params is the same as the field tags.

Types

type BitString

```
type BitString struct {  
    Bytes    []byte // bits packed into bytes.  
    BitLength int    // length in bits.  
}
```

BitString is the structure to use when you want an ASN.1 BIT STRING type. A bit string is padded up to the nearest byte in memory and the number of valid bits is recorded. Padding bits will be zero.

func (BitString) At

```
func (b BitString) At(i int) int
```

At returns the bit at the given index. If the index is out of range it returns 0.

func (BitString) RightAlign

```
func (b BitString) RightAlign() []byte
```

RightAlign returns a slice where the padding bits are at the beginning. The slice may share memory with the BitString.

type Enumerated

```
type Enumerated int
```

An Enumerated is represented as a plain int.

type Flag

```
type Flag bool
```

A Flag accepts any data and is set to true if present.

type ObjectIdentifier

```
type ObjectIdentifier []int
```

An ObjectIdentifier represents an ASN.1 OBJECT IDENTIFIER.

func (ObjectIdentifier) Equal

```
func (oi ObjectIdentifier) Equal(other ObjectIdentifier) bool
```

Equal reports whether oi and other represent the same identifier.

func (ObjectIdentifier) String

added in go1.3

```
func (oi ObjectIdentifier) String() string
```

type RawContent

```
type RawContent []byte
```

RawContent is used to signal that the undecoded, DER data needs to be preserved for a struct. To use it, the first field of the struct must have this type. It's an error for any of the other fields to have this type.

type RawValue

```
type RawValue struct {
    Class, Tag int
    IsCompound bool
    Bytes      []byte
    FullBytes  []byte // includes the tag and length
}
```

A RawValue represents an undecoded ASN.1 object.

type StructuralError

```
type StructuralError struct {
    Msg string
}
```

A StructuralError suggests that the ASN.1 data is valid, but the Go type which is receiving it doesn't match.

func (StructuralError) Error

```
func (e StructuralError) Error() string
```

type SyntaxError

```
type SyntaxError struct {
    Msg string
}
```

A SyntaxError suggests that the ASN.1 data is invalid.

func (SyntaxError) Error

```
func (e SyntaxError) Error() string
```



[asn1.go](#)

[common.go](#)

[marshal.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](#)

