govalidator

A package of validators and sanitizers for strings, structs and collections. Based on validator.js.

Installation

Make sure that Go is installed on your computer. Type the following command in your terminal:

```
go get github.com/asaskevich/govalidator
```

or you can get specified release of the package with `gopkg.in`:

```
go get gopkg.in/asaskevich/govalidator.v4
```

After it the package is ready to use.

Import package in your project

Add following line in your `*.go` file:

```
import "github.com/asaskevich/govalidator"
```

If you are unhappy to use long `govalidator`, you can do something like this:

```
import (
  valid "github.com/asaskevich/govalidator"
)
```

Activate behavior to require all fields have a validation tag by default

`SetFieldsRequiredByDefault` causes validation to fail when struct fields do not include validations or are not explicitly marked as exempt (using `valid:"-"` or `valid:"email,optional"`). A good place to activate this is a package init function or the main() function.

`SetNilPtrAllowedByRequired` causes validation to pass when struct fields marked by `required` are set to nil. This is disabled by default for consistency, but some packages that need to be able to determine between `nil` and `zero value` state can use this. If disabled, both `nil` and `zero` values cause validation errors.

```
import "github.com/asaskevich/govalidator"

func init() {
  govalidator.SetFieldsRequiredByDefault(true)
}
```

Here's some code to explain it:

```
// this struct definition will fail govalidator.ValidateStruct() (and the field
values do not matter):
type exampleStruct struct {
  Name  string ``
  Email string `valid:"email"`
}

// this, however, will only fail when Email is empty or an invalid email address:
type exampleStruct2 struct {
  Name  string `valid:"-"`
  Email string `valid:"email"`
}

// lastly, this will only fail when Email is an invalid email address but not when
it's empty:
type exampleStruct2 struct {
  Name  string `valid:"-"`
  Email string `valid:"email,optional"`
}
```

Recent breaking changes (see #123)

Custom validator function signature

A context was added as the second parameter, for structs this is the object being validated – this makes dependent validation possible.

```
import "github.com/asaskevich/govalidator"

// old signature
func(i interface{}) bool

// new signature
func(i interface{}, o interface{}) bool
```

Adding a custom validator

This was changed to prevent data races when accessing custom validators.

```
import "github.com/asaskevich/govalidator"

// before
govalidator.CustomTypeTagMap["customByteArrayValidator"] =
CustomTypeValidator(func(i interface{}, o interface{}) bool {
  // ...
})

// after
govalidator.CustomTypeTagMap.Set("customByteArrayValidator",
CustomTypeValidator(func(i interface{}, o interface{}) bool {
```

```
  // ...
}))
```

List of functions:

```
func Abs(value float64) float64
func BlackList(str, chars string) string
func ByteLength(str string, params ...string) bool
func CamelCaseToUnderscore(str string) string
func Contains(str, substring string) bool
func Count(array []interface{}, iterator ConditionIterator) int
func Each(array []interface{}, iterator Iterator)
func ErrorByField(e error, field string) string
func ErrorsByField(e error) map[string]string
func Filter(array []interface{}, iterator ConditionIterator) []interface{}
func Find(array []interface{}, iterator ConditionIterator) interface{}
func GetLine(s string, index int) (string, error)
func GetLines(s string) []string
func InRange(value, left, right float64) bool
func IsASCII(str string) bool
func IsAlpha(str string) bool
func IsAlphanumeric(str string) bool
func IsBase64(str string) bool
func IsByteLength(str string, min, max int) bool
func IsCIDR(str string) bool
func IsCreditCard(str string) bool
func IsDNSName(str string) bool
func IsDataURI(str string) bool
func IsDialString(str string) bool
func IsDivisibleBy(str, num string) bool
func IsEmail(str string) bool
func IsFilePath(str string) (bool, int)
func IsFloat(str string) bool
func IsFullWidth(str string) bool
func IsHalfWidth(str string) bool
func IsHexadecimal(str string) bool
func IsHexcolor(str string) bool
func IsHost(str string) bool
func IsIP(str string) bool
func IsIPv4(str string) bool
func IsIPv6(str string) bool
func IsISBN(str string, version int) bool
func IsISBN10(str string) bool
func IsISBN13(str string) bool
func IsISO3166Alpha2(str string) bool
func IsISO3166Alpha3(str string) bool
func IsISO693Alpha2(str string) bool
func IsISO693Alpha3b(str string) bool
func IsISO4217(str string) bool
func IsIn(str string, params ...string) bool
func IsInt(str string) bool
```

```
func IsJSON(str string) bool
func IsLatitude(str string) bool
func IsLongitude(str string) bool
func IsLowerCase(str string) bool
func IsMAC(str string) bool
func IsMongoID(str string) bool
func IsMultibyte(str string) bool
func IsNatural(value float64) bool
func IsNegative(value float64) bool
func IsNonNegative(value float64) bool
func IsNonPositive(value float64) bool
func IsNull(str string) bool
func IsNumeric(str string) bool
func IsPort(str string) bool
func IsPositive(value float64) bool
func IsPrintableASCII(str string) bool
func IsRFC3339(str string) bool
func IsRFC3339WithoutZone(str string) bool
func IsRGBcolor(str string) bool
func IsRequestURI(rawurl string) bool
func IsRequestURL(rawurl string) bool
func IsSSN(str string) bool
func IsSemver(str string) bool
func IsTime(str string, format string) bool
func IsURL(str string) bool
func IsUTFDigit(str string) bool
func IsUTFLetter(str string) bool
func IsUTFLetterNumeric(str string) bool
func IsUTFNumeric(str string) bool
func IsUUID(str string) bool
func IsUUIDv3(str string) bool
func IsUUIDv4(str string) bool
func IsUUIDv5(str string) bool
func IsUpperCase(str string) bool
func IsVariableWidth(str string) bool
func IsWhole(value float64) bool
func LeftTrim(str, chars string) string
func Map(array []interface{}, iterator ResultIterator) []interface{}
func Matches(str, pattern string) bool
func NormalizeEmail(str string) (string, error)
func PadBoth(str string, padStr string, padLen int) string
func PadLeft(str string, padStr string, padLen int) string
func PadRight(str string, padStr string, padLen int) string
func Range(str string, params ...string) bool
func RemoveTags(s string) string
func ReplacePattern(str, pattern, replace string) string
func Reverse(s string) string
func RightTrim(str, chars string) string
func RuneLength(str string, params ...string) bool
func SafeFileName(str string) string
func SetFieldsRequiredByDefault(value bool)
func Sign(value float64) float64
```

```
func StringLength(str string, params ...string) bool
func StringMatches(s string, params ...string) bool
func StripLow(str string, keepNewLines bool) string
func ToBoolean(str string) (bool, error)
func ToFloat(str string) (float64, error)
func ToInt(str string) (int64, error)
func ToJSON(obj interface{}) (string, error)
func ToString(obj interface{}) string
func Trim(str, chars string) string
func Truncate(str string, length int, ending string) string
func UnderscoreToCamelCase(s string) string
func ValidateStruct(s interface{}) (bool, error)
func WhiteList(str, chars string) string
type ConditionIterator
type CustomTypeValidator
type Error
func (e Error) Error() string
type Errors
func (es Errors) Error() string
func (es Errors) Errors() []error
type ISO3166Entry
type Iterator
type ParamValidator
type ResultIterator
type UnsupportedTypeError
func (e *UnsupportedTypeError) Error() string
type Validator
```

Examples

ISURL

```
println(govalidator.IsURL(`http://user@pass:domain.com/path/page`))
```

TOSTRING

```
type User struct {
    FirstName string
    LastName string
}

str := govalidator.ToString(&User{"John", "Juan"})
println(str)
```

EACH, MAP, FILTER, COUNT FOR SLICES

Each iterates over the slice/array and calls Iterator for every item

```
data := []interface{}{1, 2, 3, 4, 5}
var fn govalidator.Iterator = func(value interface{}, index int) {
    println(value.(int))
```

```
}
govalidator.Each(data, fn)
```

```
data := []interface{}{1, 2, 3, 4, 5}
var fn govalidator.ResultIterator = func(value interface{}, index int) interface{} {
    return value.(int) * 3
}
_ = govalidator.Map(data, fn) // result = []interface{}{1, 6, 9, 12, 15}
```

```
data := []interface{}{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}
var fn govalidator.ConditionIterator = func(value interface{}, index int) bool {
    return value.(int)%2 == 0
}
_ = govalidator.Filter(data, fn) // result = []interface{}{2, 4, 6, 8, 10}
_ = govalidator.Count(data, fn) // result = 5
```

VALIDATESTRUCT #2

If you want to validate structs, you can use tag `valid` for any field in your structure. All validators used with this field in one tag are separated by comma. If you want to skip validation, place `-` in your tag. If you need a validator that is not on the list below, you can add it like this:

```
govalidator.TagMap["duck"] = govalidator.Validator(func(str string) bool {
    return str == "duck"
})
```

For completely custom validators (interface-based), see below.

Here is a list of available validators for struct fields (validator - used function):

```
"email":             IsEmail,
"url":               IsURL,
"dialstring":        IsDialString,
"requrl":            IsRequestURL,
"requri":            IsRequestURI,
"alpha":             IsAlpha,
"utfletter":         IsUTFLetter,
"alphanum":          IsAlphanumeric,
"utfletternum":      IsUTFLetterNumeric,
"numeric":           IsNumeric,
"utfnumeric":        IsUTFNumeric,
"utfdigit":          IsUTFDigit,
"hexadecimal":       IsHexadecimal,
"hexcolor":          IsHexcolor,
"rgbcolor":          IsRGBcolor,
"lowercase":         IsLowerCase,
"uppercase":         IsUpperCase,
"int":               IsInt,
"float":             IsFloat,
"null":              IsNull,
```

```
"uuid":                IsUUID,
"uuidv3":              IsUUIDv3,
"uuidv4":              IsUUIDv4,
"uuidv5":              IsUUIDv5,
"creditcard":          IsCreditCard,
"isbn10":              IsISBN10,
"isbn13":              IsISBN13,
"json":                IsJSON,
"multibyte":           IsMultibyte,
"ascii":               IsASCII,
"printableascii":      IsPrintableASCII,
"fullwidth":           IsFullWidth,
"halfwidth":           IsHalfWidth,
"variablewidth":       IsVariableWidth,
"base64":              IsBase64,
"datauri":             IsDataURI,
"ip":                  IsIP,
"port":                IsPort,
"ipv4":                IsIPv4,
"ipv6":                IsIPv6,
"dns":                 IsDNSName,
"host":                IsHost,
"mac":                 IsMAC,
"latitude":            IsLatitude,
"longitude":           IsLongitude,
"ssn":                 IsSSN,
"semver":              IsSemver,
"rfc3339":             IsRFC3339,
"rfc3339WithoutZone": IsRFC3339WithoutZone,
"ISO3166Alpha2":       IsISO3166Alpha2,
"ISO3166Alpha3":       IsISO3166Alpha3,
```

Validators with parameters

```
"range(min|max)": Range,
"length(min|max)": ByteLength,
"runelength(min|max)": RuneLength,
"stringlength(min|max)": StringLength,
"matches(pattern)": StringMatches,
"in(string1|string2|...|stringN)": IsIn,
"rsapub(keylength)" : IsRsaPub,
```

And here is small example of usage:

```
type Post struct {
    Title    string `valid:"alphanum,required"`
    Message  string `valid:"duck,ascii"`
    Message2 string `valid:"animal(dog)"`
    AuthorIP string `valid:"ipv4"`
    Date     string `valid:"-"`
}
```

```
post := &Post{
    Title:   "My Example Post",
    Message: "duck",
    Message2: "dog",
    AuthorIP: "123.234.54.3",
}

// Add your own struct validation tags
govalidator.TagMap["duck"] = govalidator.Validator(func(str string) bool {
    return str == "duck"
})

// Add your own struct validation tags with parameter
govalidator.ParamTagMap["animal"] = govalidator.ParamValidator(func(str string,
params ...string) bool {
    species := params[0]
    return str == species
})
govalidator.ParamTagRegexMap["animal"] = regexp.MustCompile("^animal\\((\\w+)\\)$")

result, err := govalidator.ValidateStruct(post)
if err != nil {
    println("error: " + err.Error())
}
println(result)
```

WHITELIST

```
// Remove all characters from string ignoring characters between "a" and "z"
println(govalidator.WhiteList("a3a43a5a4a3a2a23a4a5a4a3a4", "a-z") ==
"aaaaaaaaaaaa")
```

CUSTOM VALIDATION FUNCTIONS

Custom validation using your own domain specific validators is also available - here's an example of how to use it:

```
import "github.com/asaskevich/govalidator"

type CustomByteArray [6]byte // custom types are supported and can be validated

type StructWithCustomByteArray struct {
  ID              CustomByteArray
`valid:"customByteArrayValidator,customMinLengthValidator"` // multiple custom
validators are possible as well and will be evaluated in sequence
  Email           string          `valid:"email"`
  CustomMinLength int             `valid:"-"`
}

govalidator.CustomTypeTagMap.Set("customByteArrayValidator",
CustomTypeValidator(func(i interface{}, context interface{}) bool {
  switch v := context.(type) { // you can type switch on the context interface being
validated
```

```
  case StructWithCustomByteArray:
    // you can check and validate against some other field in the context,
    // return early or not validate against the context at all – your choice
  case SomeOtherType:
    // ...
  default:
    // expecting some other type? Throw/panic here or continue
  }

  switch v := i.(type) { // type switch on the struct field being validated
  case CustomByteArray:
    for _, e := range v { // this validator checks that the byte array is not empty,
i.e. not all zeroes
      if e != 0 {
        return true
      }
    }
  }
  return false
}))
govalidator.CustomTypeTagMap.Set("customMinLengthValidator",
CustomTypeValidator(func(i interface{}, context interface{}) bool {
  switch v := context.(type) { // this validates a field against the value in
another field, i.e. dependent validation
  case StructWithCustomByteArray:
    return len(v.ID) >= v.CustomMinLength
  }
  return false
}))
```

CUSTOM ERROR MESSAGES

Custom error messages are supported via annotations by adding the `~` separator - here's an example of how to use it:

```
type Ticket struct {
  Id        int64     `json:"id"`
  FirstName string    `json:"firstname" valid:"required~First name is blank"`
}
```

Notes

Documentation is available here: godoc.org. Full information about code coverage is also available here: govalidator on gocover.io.

Support

If you do have a contribution to the package, feel free to create a Pull Request or an Issue.

What to contribute

If you don't know what to do, there are some features and functions that need to be done

- [ ] Refactor code

- [ ] Edit docs and README: spellcheck, grammar and typo check
- [ ] Create actual list of contributors and projects that currently using this package
- [ ] Resolve issues and bugs
- [ ] Update actual list of functions
- [ ] Update list of validators that available for `ValidateStruct` and add new
- [ ] Implement new validators: `IsFQDN`, `IsIMEI`, `IsPostalCode`, `IsISIN`, `IsISRC` etc
- [ ] Implement validation by maps
- [ ] Implement fuzzing testing
- [ ] Implement some struct/map/array utilities
- [ ] Implement map/array validation
- [ ] Implement benchmarking
- [ ] Implement batch of examples
- [ ] Look at forks for new features and fixes

Advice

Feel free to create what you want, but keep in mind when you implement new features:

- Code must be clear and readable, names of variables/constants clearly describes what they are doing
- Public functions must be documented and described in source file and added to README.md to the list of available functions
- There are must be unit-tests for any new functions and improvements

Credits

Contributors

This project exists thanks to all the people who contribute. [Contribute].

Special thanks to contributors

- Daniel Lohse
- Attila Oláh
- Daniel Korner
- Steven Wilkin
- Deiwin Sarjas
- Noah Shibley
- Nathan Davies
- Matt Sanford
- Simon ccl1115

Contribute on Open Collective

Backers

Thank you to all our backers! 🙏 [Become a backer]

Become a Backer

Sponsors

Support this project by becoming a sponsor. Your logo will show up here with a link to your website. [Become a sponsor]

Become a Sponsor

License

govalidator f21760c49a FOSSA

42 Issues Found

LICENSE SCAN

MIT - 100%

DEEP IMPACT STATS

+ 1741 Deep Dependencies

+ 11 Obligations from 101 Licenses

View More Details on FOSSA