
errors

```
1 import "github.com/juju/errors"
```

 [reference](#)

The juju/errors provides an easy way to annotate errors without losing the original error context.

The exported `New` and `Errorf` functions are designed to replace the `errors.New` and `fmt.Errorf` functions respectively. The same underlying error is there, but the package also records the location at which the error was created.

A primary use case for this library is to add extra context any time an error is returned from a function.

```
1 if err := SomeFunc(); err != nil {  
2     return err  
3 }
```

This instead becomes:

```
1 if err := SomeFunc(); err != nil {  
2     return errors.Trace(err)  
3 }
```

which just records the file and line number of the Trace call, or

```
1 if err := SomeFunc(); err != nil {  
2     return errors.Annotate(err, "more context")  
3 }
```

which also adds an annotation to the error.

When you want to check to see if an error is of a particular type, a helper function is normally exported by the package that returned the error, like the `os` package does. The underlying cause of the error is available using the `Cause` function.

```
1 os.IsNotExist(errors.Cause(err))
```

The result of the `Error()` call on an annotated error is the annotations joined with colons, then the result of the `Error()` method for the underlying error that was the cause.

```
1 err := errors.Errorf("original")  
2 err = errors.Annotatef(err, "context")  
3 err = errors.Annotatef(err, "more context")  
4 err.Error() -> "more context: context: original"
```

Obviously recording the file, line and functions is not very useful if you cannot get them back out again.

```
1 errors.ErrorStack(err)
```

will return something like:

```
1 first error
2 github.com/juju/errors/annotation_test.go:193:
3 github.com/juju/errors/annotation_test.go:194: annotation
4 github.com/juju/errors/annotation_test.go:195:
5 github.com/juju/errors/annotation_test.go:196: more context
6 github.com/juju/errors/annotation_test.go:197:
```

The first error was generated by an external system, so there was no location associated. The second, fourth, and last lines were generated with Trace calls, and the other two through Annotate.

Sometimes when responding to an error you want to return a more specific error for the situation.

```
1 if err := FindField(field); err != nil {
2     return errors.Wrap(err, errors.NotFoundf(field))
3 }
```

This returns an error where the complete error stack is still available, and `errors.Cause()` will return the `NotFound` error.

func AlreadyExistsf

```
1 func AlreadyExistsf(format string, args ...interface{}) error
```

`AlreadyExistsf` returns an error which satisfies `IsAlreadyExists()`.

func Annotate

```
1 func Annotate(other error, message string) error
```

`Annotate` is used to add extra context to an existing error. The location of the `Annotate` call is recorded with the annotations. The file, line and function are also recorded.

For example:

```
1 if err := SomeFunc(); err != nil {
2     return errors.Annotate(err, "failed to frombulate")
3 }
```

func Annotatef

```
1 func Annotatef(other error, format string, args ...interface{}) error
```

Annotatef is used to add extra context to an existing error. The location of the Annotate call is recorded with the annotations. The file, line and function are also recorded.

For example:

```
1 if err := SomeFunc(); err != nil {  
2     return errors.Annotatef(err, "failed to frombulate the %s", arg)  
3 }
```

func BadRequestf

```
1 func BadRequestf(format string, args ...interface{}) error
```

BadRequestf returns an error which satisfies `IsBadRequest()`.

func Cause

```
1 func Cause(err error) error
```

Cause returns the cause of the given error. This will be either the original error, or the result of a `Wrap` or `Mask` call.

Cause is the usual way to diagnose errors that may have been wrapped by the other errors functions.

func DeferredAnnotatef

```
1 func DeferredAnnotatef(err *error, format string, args ...interface{})
```

DeferredAnnotatef annotates the given error (when it is not nil) with the given format string and arguments (like `fmt.Sprintf`). If `*err` is nil, `DeferredAnnotatef` does nothing. This method is used in a `defer` statement in order to annotate any resulting error with the same message.

For example:

```
1 defer DeferredAnnotatef(&err, "failed to frombulate the %s", arg)
```

func Details

```
1 func Details(err error) string
```

Details returns information about the stack of errors wrapped by err, in the format:

```
1 [{filename:99: error one} {otherfile:55: cause of error one}]
```

This is a terse alternative to ErrorStack as it returns a single line.

func ErrorStack

```
1 func ErrorStack(err error) string
```

ErrorStack returns a string representation of the annotated error. If the error passed as the parameter is not an annotated error, the result is simply the result of the Error() method on that error.

If the error is an annotated error, a multi-line string is returned where each line represents one entry in the annotation stack. The full filename from the call stack is used in the output.

```
1 first error
2 github.com/juju/errors/annotation_test.go:193:
3 github.com/juju/errors/annotation_test.go:194: annotation
4 github.com/juju/errors/annotation_test.go:195:
5 github.com/juju/errors/annotation_test.go:196: more context
6 github.com/juju/errors/annotation_test.go:197:
```

func Errorf

```
1 func Errorf(format string, args ...interface{}) error
```

Errorf creates a new annotated error and records the location that the error is created. This should be a drop in replacement for fmt.Errorf.

For example:

```
1 return errors.Errorf("validation failed: %s", message)
```

func Forbiddenf

```
1 func Forbiddenf(format string, args ...interface{}) error
```

Forbiddenf returns an error which satisfies IsForbidden()

func IsAlreadyExists

```
1 func IsAlreadyExists(err error) bool
```

IsAlreadyExists reports whether the error was created with AlreadyExistsf() or NewAlreadyExists().

func IsBadRequest

```
1 func IsBadRequest(err error) bool
```

IsBadRequest reports whether err was created with BadRequestf() or NewBadRequest().

func IsForbidden

```
1 func IsForbidden(err error) bool
```

IsForbidden reports whether err was created with Forbiddenf() or NewForbidden().

func IsMethodNotAllowed

```
1 func IsMethodNotAllowed(err error) bool
```

IsMethodNotAllowed reports whether err was created with MethodNotAllowedf() or NewMethodNotAllowed().

func IsNotAssigned

```
1 func IsNotAssigned(err error) bool
```

IsNotAssigned reports whether err was created with NotAssignedf() or NewNotAssigned().

func IsNotFound

```
1 func IsNotFound(err error) bool
```

IsNotFound reports whether err was created with NotFoundf() or NewNotFound().

func IsNotImplemented

```
1 func IsNotImplemented(err error) bool
```

IsNotImplemented reports whether err was created with NotImplementedf() or NewNotImplemented().

func IsNotProvisioned

```
1 func IsNotProvisioned(err error) bool
```

IsNotProvisioned reports whether err was created with NotProvisionedf() or NewNotProvisioned().

func IsNotSupported

```
1 func IsNotSupported(err error) bool
```

IsNotSupported reports whether the error was created with NotSupportedf() or NewNotSupported().

func IsNotValid

```
1 func IsNotValid(err error) bool
```

IsNotValid reports whether the error was created with NotValidf() or NewNotValid().

func IsUnauthorized

```
1 func IsUnauthorized(err error) bool
```

IsUnauthorized reports whether err was created with Unauthorizedf() or NewUnauthorized().

func IsUserNotFound

```
1 func IsUserNotFound(err error) bool
```

IsUserNotFound reports whether err was created with UserNotFoundf() or NewUserNotFound().

func Mask

```
1 func Mask(other error) error
```

Mask hides the underlying error type, and records the location of the masking.

func Maskf

```
1 func Maskf(other error, format string, args ...interface{}) error
```

Mask masks the given error with the given format string and arguments (like `fmt.Sprintf`), returning a new error that maintains the error stack, but hides the underlying error type. The error string still contains the full annotations. If you want to hide the annotations, call `Wrap`.

func MethodNotAllowedf

```
1 func MethodNotAllowedf(format string, args ...interface{}) error
```

`MethodNotAllowedf` returns an error which satisfies `IsMethodNotAllowed()`.

func New

```
1 func New(message string) error
```

`New` is a drop in replacement for the standard library `errors` module that records the location that the error is created.

For example:

```
1 return errors.New("validation failed")
```

func NewAlreadyExists

```
1 func NewAlreadyExists(err error, msg string) error
```

`NewAlreadyExists` returns an error which wraps `err` and satisfies `IsAlreadyExists()`.

func NewBadRequest

```
1 func NewBadRequest(err error, msg string) error
```

NewBadRequest returns an error which wraps err that satisfies IsBadRequest().

func NewForbidden

```
1 func NewForbidden(err error, msg string) error
```

NewForbidden returns an error which wraps err that satisfies IsForbidden().

func NewMethodNotAllowed

```
1 func NewMethodNotAllowed(err error, msg string) error
```

NewMethodNotAllowed returns an error which wraps err that satisfies IsMethodNotAllowed().

func NewNotAssigned

```
1 func NewNotAssigned(err error, msg string) error
```

NewNotAssigned returns an error which wraps err that satisfies IsNotAssigned().

func NewNotFound

```
1 func NewNotFound(err error, msg string) error
```

NewNotFound returns an error which wraps err that satisfies IsNotFound().

func NewNotImplemented

```
1 func NewNotImplemented(err error, msg string) error
```

NewNotImplemented returns an error which wraps err and satisfies IsNotImplemented().

func NewNotProvisioned

```
1 func NewNotProvisioned(err error, msg string) error
```

NewNotProvisioned returns an error which wraps err that satisfies IsNotProvisioned().

func NewNotSupported

```
1 func NewNotSupported(err error, msg string) error
```

NewNotSupported returns an error which wraps err and satisfies IsNotSupported().

func NewNotValid

```
1 func NewNotValid(err error, msg string) error
```

NewNotValid returns an error which wraps err and satisfies IsNotValid().

func NewUnauthorized

```
1 func NewUnauthorized(err error, msg string) error
```

NewUnauthorized returns an error which wraps err and satisfies IsUnauthorized().

func NewUserNotFound

```
1 func NewUserNotFound(err error, msg string) error
```

NewUserNotFound returns an error which wraps err and satisfies IsUserNotFound().

func NotAssignedf

```
1 func NotAssignedf(format string, args ...interface{}) error
```

NotAssignedf returns an error which satisfies IsNotAssigned().

func NotFoundf

```
1 func NotFoundf(format string, args ...interface{}) error
```

NotFoundf returns an error which satisfies IsNotFound().

func NotImplementedf

```
1 func NotImplementedf(format string, args ...interface{}) error
```

NotImplementedf returns an error which satisfies IsNotImplemented().

func NotProvisionedf

```
1 func NotProvisionedf(format string, args ...interface{}) error
```

NotProvisionedf returns an error which satisfies IsNotProvisioned().

func NotSupportedf

```
1 func NotSupportedf(format string, args ...interface{}) error
```

NotSupportedf returns an error which satisfies IsNotSupported().

func NotValidf

```
1 func NotValidf(format string, args ...interface{}) error
```

NotValidf returns an error which satisfies IsNotValid().

func Trace

```
1 func Trace(other error) error
```

Trace adds the location of the Trace call to the stack. The Cause of the resulting error is the same as the error parameter. If the other error is nil, the result will be nil.

For example:

```
1 if err := SomeFunc(); err != nil {
2     return errors.Trace(err)
3 }
```

func Unauthorizedf

```
1 func Unauthorizedf(format string, args ...interface{}) error
```

Unauthorizedf returns an error which satisfies IsUnauthorized().

func UserNotFoundf

```
1 func UserNotFoundf(format string, args ...interface{}) error
```

UserNotFoundf returns an error which satisfies IsUserNotFound().

func Wrap

```
1 func Wrap(other, newDescriptive error) error
```

Wrap changes the Cause of the error. The location of the Wrap call is also stored in the error stack.

For example:

```
1 if err := SomeFunc(); err != nil {
2     newErr := &packageError{"more context", private_value}
3     return errors.Wrap(err, newErr)
4 }
```

func Wrapf

```
1 func Wrapf(other, newDescriptive error, format string, args ...
    interface{}) error
```

Wrapf changes the Cause of the error, and adds an annotation. The location of the Wrap call is also stored in the error stack.

For example:

```
1 if err := SomeFunc(); err != nil {
2     return errors.Wrapf(err, simpleErrorType, "invalid value %q", value
    )
}
```

```
3 }
```

type Err

```
1 type Err struct {  
2     // contains filtered or unexported fields  
3 }
```

Err holds a description of an error along with information about where the error was created.

It may be embedded in custom error types to add extra information that this errors package can understand.

func NewErr

```
1 func NewErr(format string, args ...interface{}) Err
```

NewErr is used to return an Err for the purpose of embedding in other structures. The location is not specified, and needs to be set with a call to SetLocation.

For example:

```
1 type FooError struct {  
2     errors.Err  
3     code int  
4 }  
5  
6 func NewFooError(code int) error {  
7     err := &FooError{errors.NewErr("foo"), code}  
8     err.SetLocation(1)  
9     return err  
10 }
```

func NewErrWithCause

```
1 func NewErrWithCause(other error, format string, args ...interface{})  
   Err
```

NewErrWithCause is used to return an Err with cause by other error for the purpose of embedding in other structures. The location is not specified, and needs to be set with a call to SetLocation.

For example:

```
1 type FooError struct {
2     errors.Err
3     code int
4 }
5
6 func (e *FooError) Annotate(format string, args ...interface{}) error {
7     err := &FooError{errors.NewErrWithCause(e.Err, format, args...), e.
8         code}
9     err.SetLocation(1)
10    return err
11 }
```

func (*Err) Cause

```
1 func (e *Err) Cause() error
```

The Cause of an error is the most recent error in the error stack that meets one of these criteria: the original error that was raised; the new error that was passed into the Wrap function; the most recently masked error; or nil if the error itself is considered the Cause. Normally this method is not invoked directly, but instead through the Cause stand alone function.

func (*Err) Error

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

Error implements error.Error.

func (*Err) Format

```
1 func (e *Err) Format(s fmt.State, verb rune)
```

Format implements fmt.Formatter When printing errors with %+v it also prints the stack trace. %#v unsurprisingly will print the real underlying type.

func (*Err) Location

```
1 func (e *Err) Location() (filename string, line int)
```

Location is the file and line of where the error was most recently created or annotated.

func (*Err) Message

```
1 func (e *Err) Message() string
```

Message returns the message stored with the most recent location. This is the empty string if the most recent call was Trace, or the message stored with Annotate or Mask.

func (*Err) SetLocation

```
1 func (e *Err) SetLocation(callDepth int)
```

SetLocation records the source location of the error at callDepth stack frames above the call.

func (*Err) StackTrace

```
1 func (e *Err) StackTrace() []string
```

StackTrace returns one string for each location recorded in the stack of errors. The first value is the originating error, with a line for each other annotation or tracing of the error.

func (*Err) Underlying

```
1 func (e *Err) Underlying() error
```

Underlying returns the previous error in the error stack, if any. A client should not ever really call this method. It is used to build the error stack and should not be introspected by client calls. Or more specifically, clients should not depend on anything but the [Cause](#) of an error.

Generated by godoc2md