Thema: Dateien

Der folgende stark gekürzte Text enthält Teile der Spezifikation des Pakets „os“.

```
Copyright 2009 The Go Authors. All rights reserved. Use of this source code is governed by a BSD-
style license that can be found in the LICENSE file. Package os provides a platform-independent
interface to operating system functionality. The design is Unix-like, although the error handling
is Go-like; failing calls return values of type error rather than error numbers. Often, more
information is available within the error. For example, if a call that takes a file name fails,
such as Open or Stat, the error will include the failing file name when printed and will be of
type *PathError, which may be unpacked for more information. The os interface is intended to be
uniform across all operating systems. Features not generally available appear in the system-
specific package syscall.

Here is a simple example, opening a file and reading some of it.
```

```
file, err := os.Open("file.go") // For read access.
if err != nil {
     log.Fatal(err)
}
// If the open fails, the error string will be self-explanatory, like
// open file.go: no such file or directory
// The file's data can then be read into a slice of bytes. Read and Write take their byte counts
// from the length of the argument slice.

data := make([]byte, 100)
count, err := file.Read(data)
if err != nil {
     log.Fatal(err)
}
fmt.Printf("read %d bytes: %q\n", count, data[:count])
```

```
// Flags to OpenFile wrapping those of the underlying system. Not all
// flags may be implemented on a given system.
Const (
     // Exactly one of O_RDONLY, O_WRONLY, or O_RDWR must be specified.
     O_RDONLY int = syscall.O_RDONLY // open the file read-only.
     O_WRONLY int = syscall.O_WRONLY // open the file write-only.
     O_RDWR   int = syscall.O_RDWR   // open the file read-write.
     // The remaining values may be or'ed in to control behavior.
     O_APPEND int = syscall.O_APPEND // append data to the file when writing.
     O_CREATE int = syscall.O_CREAT  // create a new file if none exists.
     O_EXCL   int = syscall.O_EXCL   // used with O_CREATE, file must not exist
     O_SYNC   int = syscall.O_SYNC   // open for synchronous I/O.
     O_TRUNC  int = syscall.O_TRUNC  // if possible, truncate file when opened.
)
// Seek whence values. Deprecated: Use io.SeekStart, io.SeekCurrent, and io.SeekEnd.
const (
     SEEK_SET int = 0 // seek relative to the origin of the file
     SEEK_CUR int = 1 // seek relative to the current offset
     SEEK_END int = 2 // seek relative to the end
)
```

FUNCTIONS

```
// Create creates the named file with mode 0666 (before umask), truncating it if it already
// exists. If successful, methods on the returned File can be used for I/O; the associated file
// descriptor has mode O_RDWR. If there is an error, it will be of type *PathError.
func Create(name string) (*File, error)

// Open opens the named file for reading.  If successful, methods on the returned file can be
// used for reading; the associated file descriptor has mode O_RDONLY. If there is an error, it
// will be of type *PathError.
func Open(name string) (*File, error)

// OpenFile is the generalized open call; most users will use Open or Create instead.  It opens
// the named file with specified flag (O_RDONLY etc.) and perm, (0666 etc.) if applicable.  If
// successful, methods on the returned File can be used for I/O. If there is an error, it will be
// of type *PathError.
func OpenFile(name string, flag int, perm FileMode) (*File, error)

// Close closes the File, rendering it unusable for I/O. It returns an error, if any.
func (f *File) Close() error
```

```
// Read reads up to len(b) bytes from the File. It returns the number of bytes read and an error,
// if any. EOF is signaled by a zero count with err set to io.EOF.
func (f *File) Read(b []byte) (n int, err error)

// ReadAt reads len(b) bytes from the File starting at byte offset off. It returns the number of
// bytes read and the error, if any. ReadAt always returns a non-nil error when n < len(b). At
// end of file, that error is io.EOF.
func (f *File) ReadAt(b []byte, off int64) (n int, err error)

// Seek sets the offset for the next Read or Write on file to offset, interpreted according to
// whence: 0 means relative to the origin of the file, 1 means relative to the current offset,
// and 2 means relative to the end. It returns the new offset and an error, if any. The behavior
// of Seek on a file opened with O_APPEND is not specified.
func (f *File) Seek(offset int64, whence int) (ret int64, err error)

// Write writes len(b) bytes to the File.
// It returns the number of bytes written and an error, if any.
// Write returns a non-nil error when n != len(b).
func (f *File) Write(b []byte) (n int, err error)

// WriteAt writes len(b) bytes to the File starting at byte offset off. It returns the number of
// bytes written and an error, if any. WriteAt returns a non-nil error when n != len(b).
func (f *File) WriteAt(b []byte, off int64) (n int, err error)

// WriteString is like Write, but writes the contents of string s rather than a slice of bytes.
func (f *File) WriteString(s string) (n int, err error)

// Sync commits the current contents of the file to stable storage. Typically, this means
// flushing the file system's in-memory copy of recently written data to disk.
func (f *File) Sync() error

// Truncate changes the size of the named file. If the file is a symbolic link, it changes the
// size of the link's target. If there is an error, it will be of type *PathError.
func Truncate(name string, size int64) error
```

```
// Mkdir creates a new directory with the specified name and permission bits. If there is an
// error, it will be of type *PathError.
func Mkdir(name string, perm FileMode) error

// Chdir changes the current working directory to the named directory. If there is an error, it
// will be of type *PathError.
func Chdir(dir string) error

// Rename renames (moves) oldpath to newpath. If newpath already exists, Rename replaces it. OS-
// specific restrictions may apply when oldpath and newpath are in different directories. If
// there is an error, it will be of type *LinkError.
func Rename(oldpath, newpath string) error

// Remove removes the named file or directory. If there is an error, it will be of type
// *PathError.
func Remove(name string) error

// Chmod changes the mode of the named file to mode. If the file is a symbolic link, it changes
// the mode of the link's target. If there is an error, it will be of type *PathError.
func Chmod(name string, mode FileMode) error

// Chown changes the numeric uid and gid of the named file. If the file is a symbolic link, it
// changes the uid and gid of the link's target. If there is an error, it will be of type
// *PathError.
func Chown(name string, uid, gid int) error
```

Bonus: Die exportierte Variable os.Args

Die Variable `os.Args` ist ein Slice von Strings und enthält bei einem *Aufruf eines kompilierten Go-Programms in der Konsole* die Aufrufparameter als Strings. Testen Sie das folgende Beispiel!

```
package main
import ( "os" ; "fmt" )

func main () {
  fmt.Println ("Aufruf mit ", len (os.Args), "Parametern!")
  for i:=0; i <len(os.Args);i++ {
    fmt.Println (i, os.Args[i])
  }
}
```