

# Writing to a File

This lesson provides code examples and their explanations for writing to a file.

## WE'LL COVER THE FOLLOWING ^

- Writing data to a file

## Writing data to a file #

Writing data to a file is demonstrated in the following program:

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

func main () {
    outputFile, outputError := os.OpenFile("output/output.dat", os.O_WRONLY|os.O_CREATE, 0666)
    if outputError != nil {
        fmt.Printf("An error occurred with file creation\n")
        return
    }
    defer outputFile.Close()
    outputWriter:= bufio.NewWriter(outputFile)
    outputString := "hello world!\n"
    for i:=0; i<10; i++ {
        outputWriter.WriteString(outputString)
    }
    outputWriter.Flush()
}
```



Writing using bufio Package

Apart from a filehandle, we now need a writer from `bufio`. We open a file **output.dat** for write-only at **line 9**. The file is created if it does not exist.

The `os.OpenFile` is a function that allows more control over the opening mode

The `os.OpenFile` is a function that allows more control over the opening mode of a file. We see that the `OpenFile` function takes a filename, one or more flags (logically OR-d together using the `|` bitwise OR operator if more than one) and the file permissions to use. The following flags are commonly used:

- `os.O_RDONLY`: the read flag for read-only access
- `os.O_WRONLY`: the write flag for write-only access
- `os.O_RDWR`: the flag that provides both read and write access
- `os.O_CREATE`: the create flag to create the file if it doesn't exist
- `os.O_TRUNC`: the truncate flag to truncate to size 0 if the file already exists

When reading, the file permissions are ignored so we can use a value of 0. When writing, we use the standard Unix file permissions of 0666 (even on Windows).

The error-handling is done from **line 10** to **line 13**, and at **line 14**, we assure the file will be closed with the `defer` keyword.

Then at **line 15**, we make the writer-object (the buffer) called `outputWriter`. We make a simple `hello world!\n` string, which will be written *ten* times to the buffer in the for-loop (see **line 18**). The buffer is then written completely to the file at **line 20**. In simple write tasks, this can be done more efficiently with:

```
fmt.Fprintf(outputFile, "Some test data.\n")
```

Using the F version of the `fmt` print functions that can write to any `io.Writer`, including a file.

The program below illustrates an alternative to `fmt.Fprintf`:

```
package main
import "os"

func main() {
    os.Stdout.WriteString("hello, world\n")
    f, _ := os.OpenFile("output/test.txt", os.O_CREATE|os.O_WRONLY, 0)
    defer f.Close()

    f.WriteString("hello, world in a file\n")
}
```



In the code above, at **line 5**, instead of printing with `fmt`, we use the `Stdout.WriteString` method from `os` to display a string. At **line 6**, we open a file `test.txt` for writing. At **line 9**, we are writing a string to it. Because of the `defer` at **line 7**, the file will be closed at **line 10**.

With `os.Stdout.WriteString("hello, world\n")`, we can also write to the screen. In `f, _ := os.OpenFile("test", os.O_CREATE|os.O_WRONLY, 0)` we create or open for write-only a file `test.txt`. A possible error is disregarded with `_`. We don't make use of a buffer, we write immediately to the file with: `f.WriteString( )`.

---

Now that you are familiar with the write operations, the next lesson brings you a challenge to solve.