- Contents
  - Buffered channels
    - Writing
    - Reading
    - Seeking
    - Gotchas
    - Example

File manipulation

This is a guide to basic file manipulation in OCaml using only what the standard library provides.

Official documentation for the modules of interest: the core library including the initially opened module Pervasives, Printf.

The standard library doesn't provide functions that directly read a file into a string or save a string into a file. Such functions can be found in third-party libraries such as Extlib. See Std.input_file and Std.output_file.

Buffered channels

The normal way of opening a file in OCaml returns a **channel**. There are two kinds of channels:

- channels that write to a file: type `out_channel`

- channels that read from a file: type `in_channel`

Writing

For writing into a file, you would do this:

1. Open the file to obtain an `out_channel`
2. Write stuff to the channel
3. If you want to force writing to the physical device, you must flush the channel, otherwise writing will not take place immediately.
4. When you are done, you can close the channel. This flushes the channel automatically.

Commonly used functions: `open_out`, `open_out_bin`, `flush`, `close_out`, `close_out_noerr`

Standard `out_channels`: `stdout`, `stderr`

Reading

For reading data from a file you would do this:

1. Open the file to obtain an `in_channel`
2. Read characters from the channel. Reading consumes the channel, so if you read a character, the channel will point to the next character in the file.
3. When there are no more characters to read, the `End_of_file` exception is raised. Often, this is where you want to close the channel.

Commonly used functions: `open_in`, `open_in_bin`, `close_in`, `close_in_noerr`

Standard `in_channel`: `stdin`

Seeking

Whenever you write or read something to or from a channel, the current position changes to the next character after what you just wrote or read. Occasionally, you may want to skip to a particular position in the file, or restart reading from the beginning. This is possible for channels that point to regular files, use `seek_in` or `seek_out`.

Gotchas

- Don't forget to flush your `out_channels` if you want to actually write something. This is particularly important if you are writing to non-files such as the standard output (`stdout`) or a socket.
- Don't forget to close any unused channel, because operating systems have a limit on the number of files that can be opened simultaneously. You must catch any exception that would occur during the file manipulation, close the corresponding channel, and re-raise the exception.
- The `Unix` module provides access to non-buffered file descriptors among other things. It provides standard file descriptors that have the same name as the corresponding standard channels: `stdin`, `stdout` and `stderr`. Therefore if you do `open Unix`, you may get type errors. If you want to be sure that you are using the `stdout` channel and not the `stdout` file descriptor, you can prepend it with the module name where it comes from: `Pervasives.stdout`. *Note that most things that don't seem to belong to any module actually belong to the `Pervasives` module, which is automatically opened.*
- `open_out` and `open_out_bin` truncate the given file if it already exists! Use `open_out_gen` if you

want an alternate behavior.

Example

```
open Printf

let file = "example.dat"
let message = "Hello!"

let () =
  (* Write message to file *)
  let oc = open_out file in    (* create or truncate file, return channel *)
  fprintf oc "%s\n" message;   (* write something *)
  close_out oc;                (* flush and close the channel *)

  (* Read file and display the first line *)
  let ic = open_in file in
  try
    let line = input_line ic in  (* read line from in_channel and discard \n *)
    print_endline line;          (* write the result to stdout *)
    flush stdout;                (* write on the underlying device now *)
    close_in ic                  (* close the input channel *)

  with e ->                      (* some unexpected exception occurs *)
    close_in_noerr ic;           (* emergency closing *)
    raise e                      (* exit with error: files are closed but
                                    channels are not flushed *)

  (* normal exit: all channels are flushed and closed *)
```

Contact

- Website Issues
- About This Site
- Find Us on GitHub
- Credits