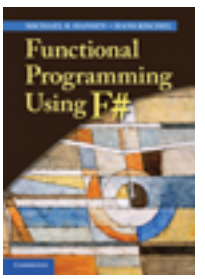

Functional Programming Using F#

Michael R. Hansen, Hans Rischel

Book DOI: http://dx.doi.org/10.1017/CBO9781139093996

Online ISBN: 9781139093996

Hardback ISBN: 9781107019027

Paperback ISBN: 9781107684065

Chapter

Appendix B - The TextProcessing library pp. 346-349

Chapter DOI: http://dx.doi.org/10.1017/CBO9781139093996.016

Cambridge University Press

Appendix B

The `TextProcessing` library

This appendix contains the source code of the `TextProcessing` library that was introduced in Chapter 10. It consists of a signature file `TextProcessing.fsi` and an implementation file `TextProcessing.fs`. This library is organized into four groups:

- A group on regular expressions. This group is documented on Page 224. See Table 10.4.
- A group on file functions. This group is documented on Page 230. See Table 10.6.
- A group on file handling. This group is documented on Page 230. See Table 10.8.
- A group on culture-dependent string ordering. This group is documented in Section 10.6. See Table 10.9.

The interface file `TextProcessing.fsi` is given in Table B.1. The listing of the implementation file `TextProcessing.fs` is split into four tables: Table B.2 – B.5, one for each of the above-mentioned groups. The source can also be found on the homepage of the book.

```
module TextProcessing

// Regular expressions

open System.Text.RegularExpressions

val captureSingle : Match -> int -> string
val captureList : Match -> int -> string list
val captureCount : Match -> int -> int
val captureCountList : Match -> int list

// File functions

open System.IO

val fileXfold : ('a -> StreamReader -> 'a) -> 'a -> string -> 'a
val fileXiter : (StreamReader -> unit) -> string -> unit
val fileFold : ('a -> string -> 'a) -> 'a -> string -> 'a
val fileIter : (string -> unit) -> string -> unit

// File handling

open System.IO

val saveValue:     'a -> string -> unit
val restoreValue:  string -> 'a
```

```
// Culture-dependent string ordering

open System

exception StringOrderingMismatch

[<Sealed>]
type orderString =
  interface IComparable

val orderString : string -> (string -> orderString)
val orderCulture : orderString -> string
```

Table B.1 *The file* `TextProcessing.fsi`

```
module TextProcessing

// Regular expressions

open System.Text.RegularExpressions

let captureSingle (ma:Match) (n:int) =
   ma.Groups.[n].Captures.[0].Value

let captureList (ma:Match) (n:int) =
  let capt = ma.Groups.[n].Captures
  let m = capt.Count - 1
  [for i in 0..m -> capt.[i].Value]

let captureCount (ma:Match) (n:int) =
  ma.Groups.[n].Captures.Count

let captureCountList (ma:Match) =
  let m = ma.Groups.Count - 1
  [for n in 0..m -> ma.Groups.[n].Captures.Count]
```

Table B.2 *The file* `TextProcessing.fs` – *Regular expression*

```
// File functions

open System
open System.IO

let fileXfold f e0 path =
  use s = File.OpenText path
  let rec fld e =
    if s.EndOfStream then e
    else fld (f e s)
  let res = fld e0
  s.Close()
  res

let fileXiter g path =
  use s = File.OpenText path
  while not(s.EndOfStream)
    do g s
  s.Close()

let fileFold f e s =
  fileXfold (fun e s -> f e (s.ReadLine())) e s

let fileIter g s =
  fileXiter (fun s -> g (s.ReadLine())) s
```

Table B.3 *The file* TextProcessing.fs *– File functions*

```
// File handling

open System.IO
open System.Runtime.Serialization.Formatters.Binary

let saveValue v path =
  use fsOut = new FileStream(path,FileMode.Create)
  let formatter = new BinaryFormatter()
  formatter.Serialize(fsOut,box v)
  fsOut.Close()

let restoreValue path =
  use fsIn = new FileStream(path,FileMode.Open)
  let formatter = new BinaryFormatter()
  let res = formatter.Deserialize(fsIn)
  fsIn.Close()
  unbox res
```

Table B.4 *The file* TextProcessing.fs *– File handling*

```
// Culture-dependent string ordering

open System.Globalization
open System

exception StringOrderingMismatch

[<CustomEquality;CustomComparison>]
type orderString =
  {Str: string; Cult: string; Cmp: string->string->int}
  override s.ToString() = s.Str
  interface System.IComparable with
    member s1.CompareTo sobj =
      match sobj with
      | :? orderString as s2 ->
        if s1.Cult <> s2.Cult then raise StringOrderingMismatch
        else
        match s1.Cmp s1.Str s2.Str with
        | 0 -> compare s1.Str s2.Str
        | z -> z
      | _ ->
        invalidArg "sobj"
                   "cannot compare values with different types"
  override s1.Equals sobj =
    match sobj with
    | :? orderString as s2 -> s1 =  s2
    | _                    -> false
  override s.GetHashCode() = hash(s.Str)

let orderString (cult: string) =
  let culInfo = CultureInfo cult
  let comp s1 s2 =
    String.Compare(s1,s2,culInfo,CompareOptions.None)
  fun s -> {Str = s; Cult = cult; Cmp = comp}: orderString

let orderCulture s = s.Cult
```

Table B.5 *The file* TextProcessing.fs *– Culture-dependent string ordering*