JFP **13** (1): 223–224, January 2003. © 2003 Cambridge University Press
DOI: 10.1017/S095679680300251X Printed in the United Kingdom

Chapter 23

System Functions

```
module System (
    ExitCode(ExitSuccess,ExitFailure),
    getArgs, getProgName, getEnv, system, exitWith, exitFailure
  ) where

data ExitCode = ExitSuccess | ExitFailure Int
                deriving (Eq, Ord, Read, Show)

getArgs                 :: IO [String]
getProgName             :: IO String
getEnv                  :: String -> IO String
system                  :: String -> IO ExitCode
exitWith                :: ExitCode -> IO a
exitFailure             :: IO a
```

This library describes the interaction of the program with the operating system.

Any `System` operation could raise an `isIllegalOperation`, as described in Section 21.1; all other permissible errors are described below. Note that, in particular, if an implementation does not support an operation it must raise an `isIllegalOperation`.

The `ExitCode` type defines the exit codes that a program can return. `ExitSuccess` indicates successful termination; and `ExitFailure` *code* indicates program failure with value *code*. The exact interpretation of $code$ is operating-system dependent. In particular, some values of $code$ may be prohibited (for instance, 0 on a POSIX-compliant system).

Computation `getArgs` returns a list of the program's command line arguments (not including the program name). Computation `getProgName` returns the name of the program as it was invoked. Computation `getEnv` var returns the value of the environment variable var. If variable var is undefined, the `isDoesNotExistError` exception is raised.

Computation `system` cmd returns the exit code produced when the operating system processes the command cmd.

Computation `exitWith` $code$ terminates the program, returning $code$ to the program's caller. Before the program terminates, any open or semi-closed handles are first closed. The caller may interpret the return code as it wishes, but the program should return `ExitSuccess` to mean normal completion, and `ExitFailure` n to mean that the program encountered a problem from which it could not recover. The value `exitFailure` is equal to `exitWith (ExitFailure` $exitfail$`)`, where $exitfail$ is implementation-dependent. `exitWith` bypasses the error handling in the I/O monad and cannot be intercepted by `catch`.

If a program terminates as a result of calling `error` or because its value is otherwise determined to be $\perp$, then it is treated identically to the computation `exitFailure`. Otherwise, if any program p terminates without calling `exitWith` explicitly, it is treated identically to the computation

```
(p >> exitWith ExitSuccess) ‘catch‘ \ _ -> exitFailure
```