Haskell Libraries

The GHC Team

Haskell Libraries
by The GHC Team

Table of Contents

Chapter 1. Introduction

Previous versions of GHC (versions 5.02 and older) came with a set of libraries called the `hslibs`, also known as the Hugs-GHC libraries. These libraries are being phased out in favour of the new hierarchical libraries, but for the time being we still provide `hslibs` for backwards compatibility.

The status of each module in `hslibs` can be considered to have three possible values:

Moved

The module has moved to the hierarchical libraries, and its documentation (in this document) will report its new location.

Not moved

The module is waiting to be moved to the new hierarchical libraries, but it hasn't moved yet. Please continue to use it from its current `hslibs` package for the time being. The documentation for the module (if it had any) is still in place in this document.

Deprecated

The module is deprecated and should not be used. A deprecated module will be indicated as such in its documentation, along with an suggested alternative API.

1.1. Usage

If you're using hslibs with GHC[i], then you need to add `-package` *`p`* to the command line for each package from which you're using a module. See the section on packages in the User's Guide (../users_guide/packages.html) for an explanation of packages.

Chapter 2. The `concurrent` package: concurrency support

The concurrency libraries (and the associated documentation) have moved. See the module `Control.Concurrent` (../base/Control.Concurrent.html) in the hierarchical libraries.

Chapter 3. The `data` package: datatypes

3.1. Edison

Edison is a complete package of data structures for Haskell. Documentation is available online (http://www.haskell.org/ghc/docs/edison/).

3.2. The `FiniteMap` type

This module has moved to `Data.FiniteMap` (../base/Data.FiniteMap.html) in the hierarchical libraries.

3.3. `Set`

This module has moved to `Data.Set` (../base/Data.Set.html) in the hierarchical libraries.

Chapter 4. The `lang` package: language support

4.1. `Bits`

This module has moved to `Data.Bits` (../base/Data.Bits.html) in the hierarchical libraries.

4.2. `CError`

This module has moved to `Foreign.C.Error` (../base/Foreign.C.Error.html) in the hierarchical libraries.

4.3. `CForeign`

This module has moved to `Foreign.C` (../base/Foreign.C.html) in the hierarchical libraries.

4.4. `CTypes`

This module has moved to `Foreign.C.Types` (../base/Foreign.C.Types.html) in the hierarchical libraries.

4.5. `CTypesISO`

This module has moved to `Foreign.C.TypesISO` (../base/Foreign.C.TypesISO.html) in the hierarchical libraries.

4.6. `CString`

This module has moved to `Foreign.C.String` (../base/Foreign.C.String.html) in the hierarchical libraries.

4.7. `DiffArray`

This module has moved to `Data.Array.Diff` (../base/Data.Array.Diff.html) in the hierarchical libraries.

4.8. `DirectoryExts`

The `DirectoryExts` module follows the footstep of other '`Exts`' modules and provides functionality that goes beyond what the Haskell 98 module `Directory` offers. That is, functionality that provides access to file/directory operations in an OS-independent manner.

`DirectoryExts` currently exports the following:

```
copyFile  :: FilePath -> FilePath -> IO ()
```

Notes:

- `copyFile` lets you copy a file to another non-existent file.

  File copying is done external to Haskell, and is for natural reasons quicker as a result and, most importantly, file copying handles the number of the OS-specific error conditions that might arise as a result of trying to perform the file copy operation.

  Should the file copying operation for some reason not succeed, the action `copyFile` raises an IO exception to signal the fact.

4.9. `Dynamic`

This module has moved to `Data.Dynamic` (../base/Data.Dynamic.html) in the hierarchical libraries.

4.10. `Exception`

This module has moved to `Control.Exception` (../base/Control.Exception.html) in the hierarchical libraries.

4.11. `Foreign`

This module has moved to `Foreign` (../base/Foreign.html) in the hierarchical libraries.

4.12. `ForeignPtr`

This module has moved to `Foreign.ForeignPtr` (../base/Foreign.ForeignPtr.html) in the hierarchical libraries.

4.13. `GlaExts`

This module has moved to `GHC.Exts` (../base/GHC.Exts.html) in the hierarchical libraries.

4.14. `IArray`

This module has moved to `Data.Array.IArray` (../base/Data.Array.IArray.html) in the hierarchical libraries.

4.15. `Int`

This module has moved to `Data.Int` (../base/Data.Int.html) in the hierarchical libraries.

4.16. `IOExts`

This library is the home for miscellaneous IO-related extensions.

4.16.1. IO monad extensions

```
fixIO :: (a -> IO a) -> IO a
```

`fixIO` allows recursive IO operations to be defined. The first argument to `fixIO` should be a function that takes its own output as an argument (sometimes called "tying the knot").

`unsafePerformIO :: IO a -> a`

This is the "back door" into the `IO` monad, allowing `IO` computation to be performed at any time. For this to be safe, the `IO` computation should be free of side effects and independent of its environment.

If the I/O computation wrapped in `unsafePerformIO` performs side effects, then the relative order in which those side effects take place (relative to the main I/O trunk, or other calls to `unsafePerformIO`) is indeterminate.

However, it is less well known that `unsafePerformIO` is not type safe. For example:

```
test :: IORef [a]
test = unsafePerformIO $ newIORef []

main = do
        writeIORef test [42]
        bang <- readIORef test
        print (bang :: [Char])
```

This program will core dump. This problem with polymorphic references is well known in the ML community, and does not arise with normal monadic use of references. There is no easy way to make it impossible once you use `unsafePerformIO`. Indeed, it is possible to write `coerce :: a -> b` with the help of `unsafePerformIO`. So be careful!

`unsafeInterleaveIO :: IO a -> IO a`

`unsafeInterleaveIO` allows `IO` computation to be deferred lazily. When passed a value of type `IO a`, the `IO` will only be performed when the value of the `a` is demanded. This is used to implement lazy file reading, see `IO.hGetContents`.

4.16.2. Mutable Variables

```
data IORef      -- instance of: Eq
newIORef        :: a -> IO (IORef a)
readIORef       :: IORef a -> IO a
writeIORef      :: IORef a -> a -> IO ()
modifyIORef     :: IORef a -> (a -> a) -> IO ()
mkWeakIORef     :: IORef a -> IO () -> IO (Weak (IORef a))

-- deprecated, use modifyIORef
updateIORef     :: IORef a -> (a -> a) -> IO ()
```

4.16.3. Mutable Arrays

```
data IOArray           -- instance of: Eq
newIOArray             :: Ix ix => (ix,ix) -> elt -> IO (IOArray ix elt)
boundsIOArray       :: Ix ix => IOArray ix elt -> (ix, ix)
readIOArray         :: Ix ix => IOArray ix elt -> ix -> IO elt
writeIOArray        :: Ix ix => IOArray ix elt -> ix -> elt -> IO ()
freezeIOArray       :: Ix ix => IOArray ix elt -> IO (Array ix elt)
thawIOArray         :: Ix ix => Array ix elt -> IO (IOArray ix elt)
unsafeFreezeIOArray :: Ix ix => IOArray ix elt -> IO (Array ix elt)
unsafeThawIOArray   :: Ix ix => Array ix elt -> IO (IOArray ix elt)
```

Note: `unsafeFreezeIOArray` and `unsafeThawIOArray` are not provided by Hugs.

4.16.4. Extended file modes

```
data IOModeEx
 = BinaryMode IOMode
 | TextMode   IOMode
   deriving (Eq, Read, Show)

openFileEx          :: FilePath -> IOModeEx -> IO Handle
hSetBinaryMode      :: Handle -> Bool -> IO Bool
```

GHC's implementation of the IO library distinguishes between binary- and text-mode files. This unfortunate hack is imposed on us by the need to support Win32 platforms.

On Win32, files opened in text mode are subject to CR-LF translation. When reading a handle in text mode, CR-LF sequences in the physical file are translated into lone LFs in the stream presented to the Haskell program. Writes to a text mode handle are subject to the inverse transformation.

On Unix platforms there is no such translation. What you get is exactly the contents of the file, and vice versa.

Unfortunately this behaviour makes it difficult to correctly implement file-positioning operations in text mode on Win32. If you want to use such operations, you must first place the handle in binary mode. Failure to do so results in IO exceptions being raised. This applies only to Win32, and not to any other platforms. If your programs use seek operations and you want them to be portable between Unix and Win32, you need to ensure the relevant handles are in binary mode.

You can get hold of a binary-mode file handle one of two ways. Either open the file with `openFileEx`, which allows the mode to be specified. Or, if you already have an open handle, use `hSetBinaryMode` to change its mode.

Also as a result of this, note that on Win32 there are also several operations which, whist still allowed, may give different results in text mode than their Unix counterparts. These are: changing buffering modes of a handle (`hSetBuffering`), and writing to a read-write handle. In both cases, the read-buffer associated with the handle needs to be flushed, and, due to the Win32 text mode translation, the resulting physical file position following the flush may be wrong.

This issue of seeking in the presence of a non-identity transform between file and buffer contents will need to be revisited when the library is re-done to properly support Unicode. The present arrangement is the least-worst kludge we could come up with at present.

4.16.5. Bulk transfers

```
hGetBuf        :: Handle -> Addr -> Int -> IO Int
hPutBuf        :: Handle -> Addr -> Int -> IO ()
```

These functions read and write chunks of data to/from a handle. They will return only when either the full buffer has been transfered, or the end of file is reached (in the case of `hGetBuf`.

```
hGetBufBA      :: Handle -> MutableByteArray RealWorld a -> Int -> IO Int
hPutBufBA      :: Handle -> MutableByteArray RealWorld a -> Int -> IO ()
```

These functions mirror the previous two functions, but operate on `MutableByteArray`s instead of `Addr`s. This may be more convenient and/or faster, depending on the circumstances.

4.16.6. Terminal control

```
hIsTerminalDevice :: Handle -> IO Bool
hSetEcho          :: Handle -> Bool -> IO ()
hGetEcho          :: Handle -> IO Bool
```

4.16.7. Redirecting handles

```
withHandleFor :: Handle -> Handle -> IO a -> IO a
withStdout    :: Handle -> IO a -> IO a
withStdin     :: Handle -> IO a -> IO a
withStderr    :: Handle -> IO a -> IO a
```

4.16.8. Trace

```
trace :: String -> a -> a
```

When called, `trace` prints the string in its first argument to standard error, before returning the second argument as its result. The `trace` function is not referentially transparent, and should only be used for debugging, or for monitoring execution. Some implementations of `trace` may decorate the string that's output to indicate that you're tracing.

`trace` is implemented using `unsafePerformIO`.

4.16.9. Extra `IOError` Predicates

The `IO` module provides several predicates over the `IOError` type, such as `isEOFError`, `isDoesNotExistError`, and so on. Here we define an extended set of these predicates, taking into account more types of error:

```
isHardwareFault    :: IOError -> Bool
isInappropriateType    :: IOError -> Bool
isInterrupted     :: IOError -> Bool
isInvalidArgument    :: IOError -> Bool
isOtherError     :: IOError -> Bool
isProtocolError    :: IOError -> Bool
isResourceVanished    :: IOError -> Bool
isSystemError     :: IOError -> Bool
isTimeExpired     :: IOError -> Bool
isUnsatisfiedConstraints :: IOError -> Bool
isUnsupportedOperation   :: IOError -> Bool
isDynIOError             :: IOError -> Bool
```

4.16.10. Miscellany

```
unsafePtrEq             :: a -> a -> Bool
slurpFile               :: FilePath -> IO (Addr, Int)
hConnectTo              :: Handle -> Handle -> IO ()
performGC               :: IO ()
freeHaskellFunctionPtr :: Addr -> IO ()

getDynIOError           :: IOError -> Maybe Dynamic.Dynamic
```

`performGC` triggers an immediate garbage collection

`unsafePtrEq` compares two values for pointer equality without evaluating them. The results are not referentially transparent and may vary significantly from one compiler to another or in the face of semantics-preserving program changes. However, pointer equality is useful in creating a number of referentially transparent constructs such as this simplified memoisation function:

```
> cache :: (a -> b) -> (a -> b)
> cache f = \x -> unsafePerformIO (check x)
>  where
>   ref = unsafePerformIO (newIORef (error "cache", error "cache"))
>   check x = readIORef ref >>= \ (x',a) ->
```

```
>           if x `unsafePtrEq` x' then
>             return a
>           else
>             let a = f x in
>             writeIORef ref (x, a) >>
>             return a
```

`getDynIOError` takes an `IOError` as argument. If it is a dynamic IO error, it returns `Just d`, where `d` is the dynamic value. Of (some) use by library providers to provide their own `IOError` types.

4.17. LazyST

The contents of this module can now be found in `Control.Monad.ST.Lazy` (../base/Control.Monad.ST.Lazy.html), and `Data.STRef.Lazy` (../base/Data.STRef.Lazy.html).

4.18. MArray

This module has moved to `Data.Array.MArray` (../base/Data.Array.MArray.html) in the hierarchical libraries.

4.19. MarshalAlloc

This module has moved to `Foreign.Marshal.Alloc` (../base/Foreign.Marshal.Alloc.html) in the hierarchical libraries.

4.20. MarshalArray

This module has moved to `Foreign.Marshal.Array` (../base/Foreign.Marshal.Array.html) in the hierarchical libraries.

4.21. MarshalError

This module has moved to `Foreign.Marshal.Error` (../base/Foreign.Marshal.Error.html) in the hierarchical libraries.

4.22. MarshalUtils

This module has moved to `Foreign.Marshal.Utils` (../base/Foreign.Marshal.Utils.html) in the hierarchical libraries.

4.23. NumExts

The `NumExts` interface collect together various numeric operations that have proven to be commonly useful

```
 -- Going between Doubles and Floats:
doubleToFloat :: Double -> Float
floatToDouble :: Float  -> Double

showHex       :: Integral a => a -> ShowS
showOct       :: Integral a => a -> ShowS
showBin       :: Integral a => a -> ShowS

showIntAtBase :: Integral a
       => a            -- base
       -> (a -> Char)  -- digit to char
       -> a            -- number to show.
       -> ShowS

showListWith  :: (a -> ShowS) -> [a] -> ShowS
```

Notes:

- If `doubleToFloat` is applied to a `Double` that is within the representable range for `Float`, the result may be the next higher or lower representable `Float` value. If the `Double` is out of range, the result is undefined.
- No loss of precision occurs in the other direction with `floatToDouble`, the floating value remains unchanged.
- `showOct`, `showHex` and `showBin` will prefix `0o`, `0x` and `0b`, respectively. Like `Numeric.showInt`, these show functions work on positive numbers only.
- `showIntAtBase` is the more general function for converting a number at some base into a series of characters. The above `show*` functions use it, for instance, here's how `showHex` could be defined

  ```
  showHex :: Integral a => a -> ShowS
  showHex n r =
   showString "0x" $
   showIntAtBase 16 (toChrHex) n r
  ```

```
where
 toChrHex d
   | d < 10    = chr (ord '0' + fromIntegral d)
   | otherwise = chr (ord 'a' + fromIntegral (d - 10))
```

- `showListWith` is strictly speaking not a '`NumExts`' kind of function, but it's sometimes useful in conjunction with the other `show*` functions that `NumExts` exports. It is the non-overloaded version of `showList`, allowing you to supply the `shows` function to use per list element. For instance,

  ```
  putStrLn (NumExts.showListWith NumExts.showHex [0..16])
  ```

  will print out the elements of `[0..16]` in hexadecimal form.

4.24. PackedString

This module has moved to `Data.PackedString` (../base/Data.PackedString.html) in the hierarchical libraries.

4.25. Ptr

This module has moved to `Foreign.Ptr` (../base/Foreign.Ptr.html) in the hierarchical libraries.

4.26. ShowFunctions

This module has moved to `Text.Show.Functions` (../base/Text.Show.Functions.html) in the hierarchical libraries.

4.27. ST

The contents of this module can now be found in `Control.Monad.ST` (../base/Control.Monad.ST.html), `Data.STRef` (../base/Data.STRef.html), and `Data.Array.ST` (../base/Data.Array.ST.html) in the hierarchical libraries.

4.28. StableName

This module has moved to `System.Mem.StableName` (../base/System.Mem.StableName.html) in the hierarchical libraries.

4.29. StablePtr

This module has moved to `Foreign.StablePtr` (../base/Foreign.StablePtr.html) in the hierarchical libraries.

4.30. Storable

This module has moved to `Foreign.Storable` (../base/Foreign.Storable.html) in the hierarchical libraries.

4.31. StorableArray

This module has moved to `Data.Array.Storable` (../base/Data.Array.Storable.html) in the hierarchical libraries.

4.32. SystemExts

The `SystemExts` module contains functionality that goes beyond what the Haskell 98 module `System` provides. That is, functionality that provides access to the underlying OS' facilities in an OS-independent manner.

Notice that `SystemExts` shares the goal of `System`. That is, it aims to provide functionality that's supported by all platforms. So, if you're looking to do serious system programming for a particular (family) of platforms, you really want to check out the libraries provided for the platform in question as well. e.g., The `Posix` library for `POSIX.1`-conforming platforms, the `Win32` library for Win32 platforms.

`SystemExts` exports the following:

```
rawSystem      :: String -> IO ExitCode
withArgs       :: [String] -> IO a -> IO a
withProgName   :: String -> IO a -> IO a
getEnvironment :: IO [(String, String)]
```

Notes:

- `rawSystem` provides the exact same behaviour as `System.system`, except that the system command isn't invoked via a shell / command interpreter.

  Not involving your platform's shell / command interpreter is quicker if you don't need its functionality, and it avoids running into limitations imposed by the shell / command interpreter.

For instance, Win32 command interpreters place a limit on the length of the command they can execute (~4k), which sometimes gets in the way of what you want to do.

- The `withArgs` action lets you change the value returned by `System.getArgs` while executing an IO action.

  When the action has finished executing (or if it raises an exception), the argument vector of `System.getArgs` is restored.

- The `withProgName` action lets you change the program name string returned by `System.getProgName` while executing an IO action.

  As `withArgs`, when the action has finished executing (or if it raises an exception), the program name string `System.getArgs` is restored.

- The `getEnvironment` action returns all the environment values present in your process' environment block.

4.33. `Weak`

This module has moved to `System.Mem.Weak` (../base/System.Mem.Weak.html) in the hierarchical libraries.

4.34. `Word`

This module has moved to `Data.Word` (../base/Data.Word.html) in the hierarchical libraries.

Chapter 5. The `net` package: networking support

(Darren Moffat supplied the initial version of this library.)

5.1. `BSD`: System database info

This module has moved to `Network.BSD` (../network/Network.BSD.html) (package `network`) in the hierarchical libraries.

5.2. `Socket`: The high-level networking interface

This module has moved to `Network` (../network/Network.html) (package `network`) in the hierarchical libraries.

5.3. `SocketPrim`: The low-level socket binding

This module has moved to `Network.Socket` (../network/Network.Socket.html) (package `network`) in the hierarchical libraries.

5.4. `URI`

This module has moved to `Network.URI` (../network/Network.URI.html) (package `network`) in the hierarchical libraries.

Chapter 6. The `num` package: numeric operations

This category is currently empty.

Chapter 7. The `posix` package: POSIX support

The `Posix` interface gives you access to the set of OS services standardised by POSIX 1003.1b (or the *IEEE Portable Operating System Interface for Computing Environments* - IEEE Std. 1003.1). The interface is accessed by `import Posix` and adding `-package posix` on your command-line.

The Posix package is *not* supported under Windows. We've looked into various ways of providing support, and other than using Cygwin, none is particularly attractive. If you want Posix support under Windows, try building GHC for Cygwin; we don't currently do this, but it is mostly supported.

7.1. Posix data types

```
data ByteCount  -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A `ByteCount` is a primitive of type `unsigned`. At a minimum, an conforming implementation must support values in the range `[0, UINT_MAX]`.

```
data ClockTick  -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A `ClockTick` is a primitive of type `clock_t`, which is used to measure intervals of time in fractions of a second. The resolution is determined by `getSysVar ClockTick`.

```
data DeviceID  -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A `DeviceID` is a primitive of type `dev_t`. It must be an arithmetic type.

```
data EpochTime -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A `EpochTime` is a primitive of type `time_t`, which is used to measure seconds since the Epoch. At a minimum, the implementation must support values in the range `[0, INT_MAX]`.

```
data FileID -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A `FileID` is a primitive of type `ino_t`. It must be an arithmetic type.

```
data FileMode -- instance of : Eq
```

A FileMode is a primitive of type mode_t. It must be an arithmetic type.

```
data FileOffset -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A FileOffset is a primitive of type off_t. It must be an arithmetic type.

```
data GroupID -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A GroupID is a primitive of type gid_t. It must be an arithmetic type.

```
data Limit -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A Limit is a primitive of type long. At a minimum, the implementation must support values in the range [LONG_MIN, LONG_MAX].

```
data LinkCount -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A LinkCount is a primitive of type nlink_t. It must be an arithmetic type.

```
data ProcessID -- instances of : Eq Ord Num Real Integral Ix Enum Show
type ProcessGroupID = ProcessID
```

A ProcessID is a primitive of type pid_t. It must be a signed arithmetic type.

```
data UserID -- instances of : Eq Ord Num Real Integral Ix Enum Show
```

A UserID is a primitive of type uid_t. It must be an arithmetic type.

```
data DirStream
```

A DirStream is a primitive of type DIR *.

```
data FileStatus
```

A FileStatus is a primitive of type struct stat.

```
data GroupEntry
```

A `GroupEntry` is a primitive of type `struct group`.

```
data ProcessTimes
```

`ProcessTimes` is a primitive structure containing a `clock_t` and a `struct tms`.

```
data SignalSet
```

An `SignalSet` is a primitive of type `sigset_t`.

```
data SystemID
```

A `SystemID` is a primitive of type `struct utsname`.

```
data TerminalAttributes
```

`TerminalAttributes` is a primitive of type `struct termios`.

```
data UserEntry
```

A `UserEntry` is a primitive of type `struct passwd`.

```
data BaudRate = B0 | B50 | B75 | B110 | B134 | B150 | B200 | B300 | B600
              | B1200 | B1800 | B2400 | B4800 | B9600 | B19200 | B38400
              deriving (Eq, Show)

data Fd

instance Eq Fd
instance Show Fd

intToFd :: Int -> Fd -- use with care.
fdToInt :: Fd -> Int     -- ditto.

data FdOption = AppendOnWrite
              | CloseOnExec
              | NonBlockingRead
```

```
data ControlCharacter = EndOfFile
                      | EndOfLine
                      | Erase
                      | Interrupt
                      | Kill
                      | Quit
                      | Suspend
                      | Start
                      | Stop

type ErrorCode = Int

type FileLock = (LockRequest, SeekMode, FileOffset, FileOffset)
--                            whence    start       length

data FlowAction = SuspendOutput | RestartOutput | TransmitStop | TransmitStart

data Handler = Default | Ignore | Catch (IO ())

data LockRequest = ReadLock | WriteLock | Unlock
                 deriving (Eq, Show)

data OpenMode = ReadOnly | WriteOnly | ReadWrite

data PathVar = LinkLimit
             | InputLineLimit
             | InputQueueLimit
             | FileNameLimit
             | PathNameLimit
             | PipeBufferLimit
             | SetOwnerAndGroupIsRestricted
             | FileNamesAreNotTruncated

data QueueSelector = InputQueue | OutputQueue | BothQueues

type Signal = Int

data SysVar = ArgumentLimit
            | ChildLimit
            | ClockTick
            | GroupLimit
            | OpenFileLimit
            | PosixVersion
            | HasSavedIDs
            | HasJobControl

data TerminalMode = InterruptOnBreak       -- BRKINT
                  | MapCRtoLF              -- ICRNL
                  | IgnoreBreak            -- IGNBRK
```

```
                  | IgnoreCR                  -- IGNCR
                  | IgnoreParityErrors        -- IGNPAR
                  | MapLFtoCR                 -- INLCR
                  | CheckParity               -- INPCK
                  | StripHighBit              -- ISTRIP
                  | StartStopInput            -- IXOFF
                  | StartStopOutput           -- IXON
                  | MarkParityErrors          -- PARMRK
                  | ProcessOutput             -- OPOST
                  | LocalMode                 -- CLOCAL
                  | ReadEnable                -- CREAD
                  | TwoStopBits               -- CSTOPB
                  | HangupOnClose             -- HUPCL
                  | EnableParity              -- PARENB
                  | OddParity                 -- PARODD
                  | EnableEcho                -- ECHO
                  | EchoErase                 -- ECHOE
                  | EchoKill                  -- ECHOK
                  | EchoLF                    -- ECHONL
                  | ProcessInput              -- ICANON
                  | ExtendedFunctions         -- IEXTEN
                  | KeyboardInterrupts        -- ISIG
                  | NoFlushOnInterrupt        -- NOFLSH
                  | BackgroundWriteInterrupt -- TOSTOP

data TerminalState = Immediately | WhenDrained | WhenFlushed

data ProcessStatus = Exited ExitCode
                   | Terminated Signal
                   | Stopped Signal
                   deriving (Eq, Show)
```

7.2. Posix Process Primitives

```
forkProcess :: IO (Maybe ProcessID)
```

`forkProcess` calls `fork`, returning `Just pid` to the parent, where `pid` is the ProcessID of the child, and returning `Nothing` to the child.

```
executeFile :: FilePath                   -- Command
            -> Bool                       -- Search PATH?
            -> [String]                   -- Arguments
```

```
           -> Maybe [(String, String)]   -- Environment
           -> IO ()
```

`executeFile cmd args env` calls one of the `execv*` family, depending on whether or not the current PATH is to be searched for the command, and whether or not an environment is provided to supersede the process's current environment. The basename (leading directory names suppressed) of the command is passed to `execv*` as `arg[0]`; the argument list passed to `executeFile` therefore begins with `arg[1]`.

```
Search PATH?    Supersede environ?      Call
~~~~~~~~~~~~    ~~~~~~~~~~~~~~~~~~      ~~~~~~~
False           False                   execv
False           True                    execve
True            False                   execvp
True            True                    execvpe*
```

Note that `execvpe` is not provided by the POSIX standard, and must be written by hand. Care must be taken to ensure that the search path is extracted from the original environment, and not from the environment to be passed on to the new image.

NOTE: In general, sharing open files between parent and child processes is potential bug farm, and should be avoided unless you really depend on this 'feature' of POSIX' `fork()` semantics. Using Haskell, there's the extra complication that arguments to `executeFile` might come from files that are read lazily (using `hGetContents`, or some such.) If this is the case, then for your own sanity, please ensure that the arguments to `executeFile` have been fully evaluated before calling `forkProcess` (followed by `executeFile`.) Consider yourself warned :-)

A successful `executeFile` overlays the current process image with a new one, so it only returns on failure.

```
runProcess :: FilePath                    -- Command
           -> [String]                    -- Arguments
           -> Maybe [(String, String)]    -- Environment (Nothing -> Inherited)
           -> Maybe FilePath              -- Working directory (Nothing -> inherited)
           -> Maybe Handle                -- stdin  (Nothing -> inherited)
           -> Maybe Handle                -- stdout (Nothing -> inherited)
           -> Maybe Handle                -- stderr (Nothing -> inherited)
           -> IO ()
```

`runProcess` is our candidate for the high-level OS-independent primitive.

`runProcess cmd args env wd inhdl outhdl errhdl` runs **cmd** (searching the current `PATH`) with arguments `args`. If `env` is `Just pairs`, the command is executed with the environment specified by `pairs` of variables and values; otherwise, the command is executed with the current environment. If `wd` is `Just dir`, the command is executed with working directory `dir`; otherwise, the command is executed in the current working directory. If `{in,out,err}hdl` is `Just handle`, the command is executed with the `Fd` for `std{in,out,err}` attached to the specified `handle`; otherwise, the `Fd` for `std{in,out,err}` is left unchanged.

```
getProcessStatus :: Bool              -- Block?
                 -> Bool              -- Stopped processes?
                 -> ProcessID
                 -> IO (Maybe ProcessStatus)
```

`getProcessStatus blk stopped pid` calls `waitpid`, returning `Just tc`, the `ProcessStatus` for process `pid` if it is available, `Nothing` otherwise. If `blk` is `False`, then `WNOHANG` is set in the options for `waitpid`, otherwise not. If `stopped` is `True`, then `WUNTRACED` is set in the options for `waitpid`, otherwise not.

```
getGroupProcessStatus :: Bool         -- Block?
                      -> Bool         -- Stopped processes?
                      -> ProcessGroupID
                      -> IO (Maybe (ProcessID, ProcessStatus))
```

`getGroupProcessStatus blk stopped pgid` calls `waitpid`, returning `Just (pid, tc)`, the `ProcessID` and `ProcessStatus` for any process in group `pgid` if one is available, `Nothing` otherwise. If `blk` is `False`, then `WNOHANG` is set in the options for `waitpid`, otherwise not. If `stopped` is `True`, then `WUNTRACED` is set in the options for `waitpid`, otherwise not.

```
getAnyProcessStatus :: Bool           -- Block?
                    -> Bool           -- Stopped processes?
                    -> IO (Maybe (ProcessID, ProcessStatus))
```

`getAnyProcessStatus blk stopped` calls `waitpid`, returning `Just (pid, tc)`, the `ProcessID` and `ProcessStatus` for any child process if one is available, `Nothing` otherwise. If `blk` is `False`, then `WNOHANG` is set in the options for `waitpid`, otherwise not. If `stopped` is `True`, then `WUNTRACED` is set in the options for `waitpid`, otherwise not.

```
exitImmediately :: ExitCode -> IO ()
```

`exitImmediately status` calls `_exit` to terminate the process with the indicated exit `status`. The operation never returns.

```
getEnvironment :: IO [(String, String)]
```

`getEnvironment` parses the environment variable mapping provided by `environ`, returning `(variable, value)` pairs. The operation never fails.

```
setEnvironment :: [(String, String)] -> IO ()
```

`setEnvironment` replaces the process environment with the provided mapping of `(variable, value)` pairs.

```
getEnvVar :: String -> IO String
```

`getEnvVar var` returns the value associated with variable `var` in the current environment (identical functionality provided through standard Haskell library function `System.getEnv`).

The operation may fail with:

`NoSuchThing`

The variable has no mapping in the current environment.

```
setEnvVar :: String -> String -> IO ()
```

`setEnvVar var val` sets the value associated with variable `var` in the current environment to be `val`. Any previous mapping is superseded.

```
removeEnvVar :: String -> IO ()
```

`removeEnvVar var` removes any value associated with variable `var` in the current environment. Deleting a variable for which there is no mapping does not generate an error.

```
nullSignal :: Signal
nullSignal = 0

backgroundRead, sigTTIN         :: Signal
backgroundWrite, sigTTOU        :: Signal
continueProcess, sigCONT        :: Signal
floatingPointException, sigFPE :: Signal
illegalInstruction, sigILL     :: Signal
internalAbort, sigABRT         :: Signal
```

```
keyboardSignal, sigINT         :: Signal
keyboardStop, sigTSTP          :: Signal
keyboardTermination, sigQUIT   :: Signal
killProcess, sigKILL           :: Signal
lostConnection, sigHUP         :: Signal
openEndedPipe, sigPIPE         :: Signal
processStatusChanged, sigCHLD  :: Signal
realTimeAlarm, sigALRM         :: Signal
segmentationViolation, sigSEGV :: Signal
softwareStop, sigSTOP          :: Signal
softwareTermination, sigTERM   :: Signal
userDefinedSignal1, sigUSR1    :: Signal
userDefinedSignal2, sigUSR2    :: Signal

signalProcess :: Signal -> ProcessID -> IO ()
```

`signalProcess int pid` calls `kill` to signal process `pid` with interrupt signal `int`.

```
raiseSignal :: Signal -> IO ()
```

`raiseSignal int` calls `kill` to signal the current process with interrupt signal `int`.

```
signalProcessGroup :: Signal -> ProcessGroupID -> IO ()
```

`signalProcessGroup int pgid` calls `kill` to signal all processes in group `pgid` with interrupt signal `int`.

```
setStoppedChildFlag :: Bool -> IO Bool
```

`setStoppedChildFlag bool` sets a flag which controls whether or not the `NOCLDSTOP` option will be used the next time a signal handler is installed for `SIGCHLD`. If `bool` is `True` (the default), `NOCLDSTOP` will not be used; otherwise it will be. The operation never fails.

```
queryStoppedChildFlag :: IO Bool
```

`queryStoppedChildFlag` queries the flag which controls whether or not the `NOCLDSTOP` option will be used the next time a signal handler is installed for `SIGCHLD`. If `NOCLDSTOP` will be used, it returns `False`; otherwise (the default) it returns `True`. The operation never fails.

```
emptySignalSet :: SignalSet
```

```
fullSignalSet  :: SignalSet
addSignal      :: Signal -> SignalSet -> SignalSet
deleteSignal   :: Signal -> SignalSet -> SignalSet
inSignalSet    :: Signal -> SignalSet -> Bool

installHandler :: Signal
               -> Handler
               -> Maybe SignalSet       -- other signals to block
               -> IO Handler            -- old handler
```

`installHandler int handler iset` calls `sigaction` to install an interrupt handler for signal `int`. If `handler` is `Default`, `SIG_DFL` is installed; if `handler` is `Ignore`, `SIG_IGN` is installed; if `handler` is `Catch action`, a handler is installed which will invoke `action` in a new thread when (or shortly after) the signal is received. See for details on how to communicate between threads.

If `iset` is `Just s`, then the `sa_mask` of the `sigaction` structure is set to `s`; otherwise it is cleared. The previously installed signal handler for `int` is returned.

```
getSignalMask :: IO SignalSet
```

`getSignalMask` calls `sigprocmask` to determine the set of interrupts which are currently being blocked.

```
setSignalMask :: SignalSet -> IO SignalSet
```

`setSignalMask mask` calls `sigprocmask` with `SIG_SETMASK` to block all interrupts in `mask`. The previous set of blocked interrupts is returned.

```
blockSignals :: SignalSet -> IO SignalSet
```

`setSignalMask mask` calls `sigprocmask` with `SIG_BLOCK` to add all interrupts in `mask` to the set of blocked interrupts. The previous set of blocked interrupts is returned.

```
unBlockSignals :: SignalSet -> IO SignalSet
```

`setSignalMask mask` calls `sigprocmask` with `SIG_UNBLOCK` to remove all interrupts in `mask` from the set of blocked interrupts. The previous set of blocked interrupts is returned.

```
getPendingSignals :: IO SignalSet
```

`getPendingSignals` calls `sigpending` to obtain the set of interrupts which have been received but are currently blocked.

```
awaitSignal :: Maybe SignalSet -> IO ()
```

`awaitSignal iset` suspends execution until an interrupt is received. If `iset` is `Just s`, `awaitSignal` calls `sigsuspend`, installing `s` as the new signal mask before suspending execution; otherwise, it calls `pause`. `awaitSignal` returns on receipt of a signal. If you have installed any signal handlers with `installHandler`, it may be wise to call `yield` directly after `awaitSignal` to ensure that the signal handler runs as promptly.

```
scheduleAlarm :: Int -> IO Int
```

`scheduleAlarm i` calls `alarm` to schedule a real time alarm at least `i` seconds in the future.

```
sleep :: Int -> IO ()
```

`sleep i` calls `sleep` to suspend execution of the program until at least `i` seconds have elapsed or a signal is received.

7.3. Posix Process Environment

```
getProcessID :: IO ProcessID
```

`getProcessID` calls `getpid` to obtain the `ProcessID` for the current process.

```
getParentProcessID :: IO ProcessID
```

`getProcessID` calls `getppid` to obtain the `ProcessID` for the parent of the current process.

```
getRealUserID :: IO UserID
```

`getRealUserID` calls `getuid` to obtain the real `UserID` associated with the current process.

```
getEffectiveUserID :: IO UserID
```

`getEffectiveUserID` calls `geteuid` to obtain the effective `UserID` associated with the current process.

```
setUserID :: UserID -> IO ()
```

`setUserID uid` calls `setuid` to set the real, effective, and saved set-user-id associated with the current process to `uid`.

```
getLoginName :: IO String
```

`getLoginName` calls `getlogin` to obtain the login name associated with the current process.

```
getRealGroupID :: IO GroupID
```

`getRealGroupID` calls `getgid` to obtain the real `GroupID` associated with the current process.

```
getEffectiveGroupID :: IO GroupID
```

`getEffectiveGroupID` calls `getegid` to obtain the effective `GroupID` associated with the current process.

```
setGroupID :: GroupID -> IO ()
```

`setGroupID gid` calls `setgid` to set the real, effective, and saved set-group-id associated with the current process to `gid`.

```
getGroups :: IO [GroupID]
```

`getGroups` calls `getgroups` to obtain the list of supplementary `GroupID`s associated with the current process.

```
getEffectiveUserName :: IO String
```

`getEffectiveUserName` calls `cuserid` to obtain a name associated with the effective `UserID` of the process.

```
getProcessGroupID :: IO ProcessGroupID
```

`getProcessGroupID` calls `getpgrp` to obtain the `ProcessGroupID` for the current process.

```
createProcessGroup :: ProcessID -> IO ProcessGroupID
```

`createProcessGroup pid` calls `setpgid` to make process `pid` a new process group leader.

```
joinProcessGroup :: ProcessGroupID -> IO ProcessGroupID
```

`joinProcessGroup pgid` calls `setpgid` to set the `ProcessGroupID` of the current process to `pgid`.

```
setProcessGroupID :: ProcessID -> ProcessGroupID -> IO ()
```

`setProcessGroupID pid pgid` calls `setpgid` to set the `ProcessGroupID` for process `pid` to `pgid`.

```
createSession :: IO ProcessGroupID
```

`createSession` calls `setsid` to create a new session with the current process as session leader.

```
systemName :: SystemID -> String
nodeName :: SystemID -> String
release :: SystemID -> String
version :: SystemID -> String
machine :: SystemID -> String

getSystemID :: IO SystemID
```

`getSystemID` calls `uname` to obtain information about the current operating system.

```
> epochTime :: IO EpochTime
```

`epochTime` calls `time` to obtain the number of seconds that have elapsed since the epoch (Jan 01 00:00:00 GMT 1970).

```
elapsedTime     :: ProcessTimes -> ClockTick
userTime        :: ProcessTimes -> ClockTick
systemTime      :: ProcessTimes -> ClockTick
childUserTime   :: ProcessTimes -> ClockTick
childSystemTime :: ProcessTimes -> ClockTick

getProcessTimes :: IO ProcessTimes
```

`getProcessTimes` calls `times` to obtain time-accounting information for the current process and its children.

```
getControllingTerminalName :: IO FilePath
```

`getControllingTerminalName` calls `ctermid` to obtain a name associated with the controlling terminal for the process. If a controlling terminal exists, `getControllingTerminalName` returns the name of the controlling terminal.

The operation may fail with:

`NoSuchThing`

There is no controlling terminal, or its name cannot be determined.

`SystemError`

Various other causes.

```
getTerminalName :: Fd -> IO FilePath
```

`getTerminalName fd` calls `ttyname` to obtain a name associated with the terminal for `Fd fd`. If `fd` is associated with a terminal, `getTerminalName` returns the name of the terminal.

The operation may fail with:

`InappropriateType`

The channel is not associated with a terminal.

`NoSuchThing`

The channel is associated with a terminal, but it has no name.

`SystemError`

Various other causes.

```
queryTerminal :: Fd -> IO Bool
```

`queryTerminal fd` calls `isatty` to determine whether or not `Fd fd` is associated with a terminal.

```
getSysVar :: SysVar -> IO Limit
```

`getSysVar var` calls `sysconf` to obtain the dynamic value of the requested configurable system limit or option. For defined system limits, `getSysVar` returns the associated value. For defined system options, the result of `getSysVar` is undefined, but not failure.

The operation may fail with:

`NoSuchThing`

The requested system limit or option is undefined.

7.4. Posix operations on files and directories

```
openDirStream :: FilePath -> IO DirStream
```

`openDirStream dir` calls `opendir` to obtain a directory stream for `dir`.

```
readDirStream :: DirStream -> IO String
```

`readDirStream dp` calls `readdir` to obtain the next directory entry (`struct dirent`) for the open directory stream `dp`, and returns the `d_name` member of that structure.

The operation may fail with:

`EOF`

End of file has been reached.

`SystemError`

Various other causes.

```
rewindDirStream :: DirStream -> IO ()
```

`rewindDirStream dp` calls `rewinddir` to reposition the directory stream `dp` at the beginning of the directory.

```
closeDirStream :: DirStream -> IO ()
```

`closeDirStream dp` calls `closedir` to close the directory stream `dp`.

```
getWorkingDirectory :: IO FilePath
```

`getWorkingDirectory` calls `getcwd` to obtain the name of the current working directory.

```
changeWorkingDirectory :: FilePath -> IO ()
```

`changeWorkingDirectory dir` calls `chdir` to change the current working directory to `dir`.

```
nullFileMode       :: FileMode       -- ---------
ownerReadMode      :: FileMode       -- r--------
ownerWriteMode     :: FileMode       -- -w-------
ownerExecuteMode   :: FileMode       -- --x------
groupReadMode      :: FileMode       -- ---r-----
groupWriteMode     :: FileMode       -- ----w----
groupExecuteMode   :: FileMode       -- -----x---
otherReadMode      :: FileMode       -- ------r--
otherWriteMode     :: FileMode       -- -------w-
otherExecuteMode   :: FileMode       -- --------x
setUserIDMode      :: FileMode       -- --S------
setGroupIDMode     :: FileMode       -- -----S---

stdFileMode        :: FileMode       -- rw-rw-rw-

ownerModes         :: FileMode       -- rwx------
```

```
groupModes         :: FileMode       -- ---rwx---
otherModes         :: FileMode       -- ------rwx
accessModes        :: FileMode       -- rwxrwxrwx

unionFileModes     :: FileMode -> FileMode -> FileMode
intersectFileModes :: FileMode -> FileMode -> FileMode

stdInput  :: Fd
stdInput  = intToFd 0

stdOutput :: Fd
stdOutput = intToFd 1

stdError  :: Fd
stdError  = intToFd 2

data OpenFileFlags =
 OpenFileFlags {
    append    :: Bool,
    exclusive :: Bool,
    noctty    :: Bool,
    nonBlock  :: Bool,
    trunc     :: Bool
 }

openFd :: FilePath
       -> OpenMode
       -> Maybe FileMode  -- Just x => O_CREAT, Nothing => must exist
       -> OpenFileFlags
       -> IO Fd
```

`openFd path acc mode (OpenFileFlags app excl noctty nonblock trunc)` calls `open` to obtain a `Fd` for the file `path` with access mode `acc`. If `mode` is `Just m`, the `O_CREAT` flag is set and the file's permissions will be based on `m` if it does not already exist; otherwise, the `O_CREAT` flag is not set. The arguments `app`, `excl`, `noctty`, `nonblock`, and `trunc` control whether or not the flags `O_APPEND`, `O_EXCL`, `O_NOCTTY`, `O_NONBLOCK`, and `O_TRUNC` are set, respectively.

```
createFile :: FilePath -> FileMode -> IO Fd
```

`createFile path mode` calls `creat` to obtain a `Fd` for file `path`, which will be created with permissions based on `mode` if it does not already exist.

```
setFileCreationMask :: FileMode -> IO FileMode
```

`setFileCreationMask mode` calls `umask` to set the process's file creation mask to `mode`. The previous file creation mask is returned.

```
createLink :: FilePath -> FilePath -> IO ()
```

`createLink old new` calls `link` to create a new path, `new`, linked to an existing file, `old`.

```
createDirectory :: FilePath -> FileMode -> IO ()
```

`createDirectory dir mode` calls `mkdir` to create a new directory, `dir`, with permissions based on `mode`.

```
createNamedPipe :: FilePath -> FileMode -> IO ()
```

`createNamedPipe fifo mode` calls `mkfifo` to create a new named pipe, `fifo`, with permissions based on `mode`.

```
removeLink :: FilePath -> IO ()
```

`removeLink path` calls `unlink` to remove the link named `path`.

```
removeDirectory :: FilePath -> IO ()
```

`removeDirectory dir` calls `rmdir` to remove the directory named `dir`.

```
rename :: FilePath -> FilePath -> IO ()
```

`rename old new` calls `rename` to rename a file or directory from `old` to `new`.

```
fileMode          :: FileStatus -> FileMode

fileID            :: FileStatus -> FileID
deviceID          :: FileStatus -> DeviceID

linkCount         :: FileStatus -> LinkCount

fileOwner         :: FileStatus -> UserID
fileGroup         :: FileStatus -> GroupID
```

```
fileSize          :: FileStatus -> FileOffset

accessTime        :: FileStatus -> EpochTime
modificationTime  :: FileStatus -> EpochTime
statusChangeTime  :: FileStatus -> EpochTime

isDirectory       :: FileStatus -> Bool
isCharacterDevice :: FileStatus -> Bool
isBlockDevice     :: FileStatus -> Bool
isRegularFile     :: FileStatus -> Bool
isNamedPipe       :: FileStatus -> Bool

getFileStatus     :: FilePath -> IO FileStatus
```

`getFileStatus path` calls `stat` to get the `FileStatus` information for the file `path`.

```
getFdStatus :: Fd -> IO FileStatus
```

`getFdStatus fd` calls `fstat` to get the `FileStatus` information for the file associated with `Fd fd`.

```
queryAccess :: FilePath -> Bool -> Bool -> Bool -> IO Bool
```

`queryAccess path r w x` calls `access` to test the access permissions for file `path`. The three arguments, `r`, `w`, and `x` control whether or not `access` is called with `R_OK`, `W_OK`, and `X_OK` respectively.

```
queryFile :: FilePath -> IO Bool
```

`queryFile path` calls `access` with `F_OK` to test for the existence for file `path`.

```
setFileMode :: FilePath -> FileMode -> IO ()
```

`setFileMode path mode` calls `chmod` to set the permission bits associated with file `path` to `mode`.

```
setOwnerAndGroup :: FilePath -> UserID -> GroupID -> IO ()
```

`setOwnerAndGroup path uid gid` calls `chown` to set the `UserID` and `GroupID` associated with file `path` to `uid` and `gid`, respectively.

```
setFileTimes :: FilePath -> EpochTime -> EpochTime -> IO ()
```

`setFileTimes path atime mtime` calls `utime` to set the access and modification times associated with file `path` to `atime` and `mtime`, respectively.

```
touchFile :: FilePath -> IO ()
```

`touchFile path` calls `utime` to set the access and modification times associated with file `path` to the current time.

```
getPathVar :: PathVar -> FilePath -> IO Limit
```

`getPathVar var path` calls `pathconf` to obtain the dynamic value of the requested configurable file limit or option associated with file or directory `path`. For defined file limits, `getPathVar` returns the associated value. For defined file options, the result of `getPathVar` is undefined, but not failure. The operation may fail with:

`NoSuchThing`

The requested file limit or option is undefined.

`SystemError`

Various other causes.

```
getFdVar :: PathVar -> Fd -> IO Limit
```

`getFdVar var fd` calls `fpathconf` to obtain the dynamic value of the requested configurable file limit or option associated with the file or directory attached to the open channel `fd`. For defined file limits, `getFdVar` returns the associated value. For defined file options, the result of `getFdVar` is undefined, but not failure.

The operation may fail with:

`NoSuchThing`

The requested file limit or option is undefined.

`SystemError`

Various other causes.

7.5. Posix Input and Output Primitives

```
createPipe :: IO (Fd, Fd)
```

`createPipe` calls `pipe` to create a pipe and returns a pair of `Fd`s, the first for reading and the second for writing.

```
dup :: Fd -> IO Fd
```

`dup fd` calls `dup` to duplicate `Fd` `fd` to another `Fd`.

```
dupTo :: Fd -> Fd -> IO ()
```

`dupTo src dst` calls `dup2` to duplicate `Fd` `src` to `Fd` `dst`.

```
fdClose :: Fd -> IO ()
```

`fdClose fd` calls `close` to close `Fd` `fd`.

```
fdRead :: Fd -> ByteCount -> IO (String, ByteCount)
```

`fdRead fd nbytes` calls `read` to read at most `nbytes` bytes from `Fd` `fd`, and returns the result as a string paired with the number of bytes actually read.

The operation may fail with:

`EOF`

End of file has been reached.

`SystemError`

Various other causes.

```
fdWrite :: Fd -> String -> IO ByteCount
```

`fdWrite fd s` calls `write` to write the string `s` to `Fd fd` as a contiguous sequence of bytes. It returns the number of bytes successfully written.

```
queryFdOption :: FdOption -> Fd -> IO Bool
```

`getFdOption opt fd` calls `fcntl` to determine whether or not the flag associated with `FdOption opt` is set for `Fd fd`.

```
setFdOption :: Fd -> FdOption -> Bool -> IO ()
```

`setFdOption fd opt val` calls `fcntl` to set the flag associated with `FdOption opt` on `Fd fd` to `val`.

```
getLock :: Fd -> FileLock -> IO (Maybe (ProcessID, FileLock))
```

`getLock fd lock` calls `fcntl` to get the first `FileLock` for `Fd fd` which blocks the `FileLock lock`. If no such `FileLock` exists, `getLock` returns `Nothing`. Otherwise, it returns `Just (pid, block)`, where `block` is the blocking `FileLock` and `pid` is the `ProcessID` of the process holding the blocking `FileLock`.

```
setLock :: Fd -> FileLock -> IO ()
```

`setLock fd lock` calls `fcntl` with `F_SETLK` to set or clear a lock segment for `Fd fd` as indicated by the `FileLock lock`. `setLock` does not block, but fails with `SystemError` if the request cannot be satisfied immediately.

```
waitToSetLock :: Fd -> FileLock -> IO ()
```

`waitToSetLock fd lock` calls `fcntl` with `F_SETLKW` to set or clear a lock segment for `Fd fd` as indicated by the `FileLock lock`. If the request cannot be satisfied immediately, `waitToSetLock` blocks until the request can be satisfied.

```
fdSeek :: Fd -> SeekMode -> FileOffset -> IO FileOffset
```

`fdSeek fd whence offset` calls `lseek` to position the `Fd fd` at the given `offset` from the starting location indicated by `whence`. It returns the resulting offset from the start of the file in bytes.

7.6. Posix, Device- and Class-Specific Functions

```
terminalMode    :: TerminalMode -> TerminalAttributes -> Bool
withMode        :: TerminalAttributes -> TerminalMode -> TerminalAttributes
withoutMode     :: TerminalAttributes -> TerminalMode -> TerminalAttributes

bitsPerByte     :: TerminalAttributes -> Int
withBits        :: TerminalAttributes -> Int -> TerminalAttributes

controlChar     :: TerminalAttributes -> ControlCharacter -> Maybe Char
withCC          :: TerminalAttributes
                -> (ControlCharacter, Char)
                -> TerminalAttributes
withoutCC       :: TerminalAttributes
                -> ControlCharacter
                -> TerminalAttributes

inputTime       :: TerminalAttributes -> Int
withTime        :: TerminalAttributes -> Int -> TerminalAttributes

minInput        :: TerminalAttributes -> Int
withMinInput    :: TerminalAttributes -> Int -> TerminalAttributes

inputSpeed      :: TerminalAttributes -> BaudRate
withInputSpeed  :: TerminalAttributes -> BaudRate -> TerminalAttributes

outputSpeed     :: TerminalAttributes -> BaudRate
withOutputSpeed :: TerminalAttributes -> BaudRate -> TerminalAttributes

getTerminalAttributes :: Fd -> IO TerminalAttributes
```

`getTerminalAttributes fd` calls `tcgetattr` to obtain the `TerminalAttributes` associated with `Fd fd`.

```
setTerminalAttributes :: Fd
                      -> TerminalAttributes
                      -> TerminalState
                      -> IO ()
```

`setTerminalAttributes fd attr ts` calls `tcsetattr` to change the `TerminalAttributes` associated with `Fd fd` to `attr`, when the terminal is in the state indicated by `ts`.

```
sendBreak :: Fd -> Int -> IO ()
```

`sendBreak fd duration` calls `tcsendbreak` to transmit a continuous stream of zero-valued bits on `Fd fd` for the specified implementation-dependent `duration`.

```
drainOutput :: Fd -> IO ()
```

`drainOutput fd` calls `tcdrain` to block until all output written to `Fd fd` has been transmitted.

```
discardData :: Fd -> QueueSelector -> IO ()
```

`discardData fd queues` calls `tcflush` to discard pending input and/or output for `Fd fd`, as indicated by the `QueueSelector queues`.

```
controlFlow :: Fd -> FlowAction -> IO ()
```

`controlFlow fd action` calls `tcflow` to control the flow of data on `Fd fd`, as indicated by `action`.

```
getTerminalProcessGroupID :: Fd -> IO ProcessGroupID
```

`getTerminalProcessGroupID fd` calls `tcgetpgrp` to obtain the `ProcessGroupID` of the foreground process group associated with the terminal attached to `Fd fd`.

```
setTerminalProcessGroupID :: Fd -> ProcessGroupID -> IO ()
```

`setTerminalProcessGroupID fd pgid` calls `tcsetpgrp` to set the `ProcessGroupID` of the foreground process group associated with the terminal attached to `Fd fd` to `pgid`.

7.7. Posix System Databases

```
groupName    :: GroupEntry -> String
groupID      :: GroupEntry -> GroupID
groupMembers :: GroupEntry -> [String]

getGroupEntryForID :: GroupID -> IO GroupEntry
```

`getGroupEntryForID gid` calls `getgrgid` to obtain the `GroupEntry` information associated with `GroupID gid`.

The operation may fail with:

`NoSuchThing`

There is no group entry for the GroupID.

```
getGroupEntryForName :: String -> IO GroupEntry
```

`getGroupEntryForName name` calls `getgrnam` to obtain the `GroupEntry` information associated with the group called `name`.

The operation may fail with:

`NoSuchThing`

There is no group entry for the name.

```
userName      :: UserEntry -> String
userID        :: UserEntry -> UserID
userGroupID   :: UserEntry -> GroupID
homeDirectory :: UserEntry -> String
userShell     :: UserEntry -> String

getUserEntryForID :: UserID -> IO UserEntry
```

`getUserEntryForID gid` calls `getpwuid` to obtain the `UserEntry` information associated with `UserID uid`. The operation may fail with:

`NoSuchThing`

There is no user entry for the UserID.

```
getUserEntryForName :: String -> IO UserEntry
```

`getUserEntryForName name` calls `getpwnam` to obtain the `UserEntry` information associated with the user login `name`.

The operation may fail with:

`NoSuchThing`

There is no user entry for the name.

7.8. POSIX Errors

```
getErrorCode :: IO ErrorCode
```

`getErrorCode` returns the current value of the external variable `errno`. It never fails.

```
setErrorCode :: ErrorCode -> IO ()
```

`setErrorCode err` sets the external variable `errno` to `err`. It never fails.

```
noError :: ErrorCode
noError = 0

argumentListTooLong, e2BIG               :: ErrorCode
badFd, eBADF                             :: ErrorCode
brokenPipe, ePIPE                        :: ErrorCode
directoryNotEmpty, eNOTEMPTY             :: ErrorCode
execFormatError, eNOEXEC                 :: ErrorCode
fileAlreadyExists, eEXIST                :: ErrorCode
fileTooLarge, eFBIG                      :: ErrorCode
filenameTooLong, eNAMETOOLONG            :: ErrorCode
improperLink, eXDEV                      :: ErrorCode
inappropriateIOControlOperation, eNOTTY :: ErrorCode
```

```
inputOutputError, eIO                     :: ErrorCode
interruptedOperation, eINTR               :: ErrorCode
invalidArgument, eINVAL                   :: ErrorCode
invalidSeek, eSPIPE                       :: ErrorCode
isADirectory, eISDIR                      :: ErrorCode
noChildProcess, eCHILD                    :: ErrorCode
noLocksAvailable, eNOLCK                  :: ErrorCode
noSpaceLeftOnDevice, eNOSPC               :: ErrorCode
noSuchOperationOnDevice, eNODEV           :: ErrorCode
noSuchDeviceOrAddress, eNXIO              :: ErrorCode
noSuchFileOrDirectory, eNOENT             :: ErrorCode
noSuchProcess, eSRCH                      :: ErrorCode
notADirectory, eNOTDIR                    :: ErrorCode
notEnoughMemory, eNOMEM                   :: ErrorCode
operationNotImplemented, eNOSYS           :: ErrorCode
operationNotPermitted, ePERM              :: ErrorCode
permissionDenied, eACCES                  :: ErrorCode
readOnlyFileSystem, eROFS                 :: ErrorCode
resourceBusy, eBUSY                       :: ErrorCode
resourceDeadlockAvoided, eDEADLK          :: ErrorCode
resourceTemporarilyUnavailable, eAGAIN    :: ErrorCode
tooManyLinks, eMLINK                      :: ErrorCode
tooManyOpenFiles, eMFILE                  :: ErrorCode
tooManyOpenFilesInSystem, eNFILE          :: ErrorCode
```

7.9. POpen

POpen provides a convenient way of sending string input to a subprocess and reading output from it lazily.

```
popen :: FilePath   -- Command
      -> [String]   -- Arguments
      -> Maybe String   -- Input
      -> IO (String, String, ProcessID) -- (stdout, stderr, pid)
```

`popen cmd args inp` executes `cmd` with `args` in a forked process. If `inp` is `Just str` then str in sent in a pipe to the standard input of the process. The output and error streams from the process are returned, together with the process id.

```
popenEnvDir :: FilePath    -- Command
     -> [String]    -- Arguments
     -> Maybe String   -- Input
     -> Maybe [(String, String)]  -- Environment
```

```
    -> Maybe FilePath    -- Working directory
    -> IO (String, String, ProcessID) -- (stdout, stderr, pid)
```

`popenEnvDir cmd args inp env dir` like `popen` executes `cmd` with `args` in a forked process. If `inp` is `Just str` then str in sent in a pipe to the standard input of the process. If `env` is `Just pairs`, the command in executed in the environment specified by `pairs`, instead of the current one. If `dir` is `Just d` the command is executed in directory `d` instead of the current directory. The output and error streams from the process are returned, together with the process id.

Chapter 8. The `text` package: text manipulation

8.1. `HaXml`: Handling XML data

HaXml is a library for converting Haskell data structures into XML and vice versa. The documentation hasn't been incorporated into this book as yet, but for now it can be found at The HaXml page (http://www.cs.york.ac.uk/fp/HaXml/).

8.2. `MatchPS`: The Perl-like matching interface

(Sigbjørn Finne supplied the regular-expressions interface.)

The `MatchPS` module provides Perl-like "higher-level" facilities to operate on `PackedStrings` (Section 4.24). The regular expressions in question are in Perl syntax. The "flags" on various functions can include: `i` for case-insensitive, `s` for single-line mode, and `g` for global. (It's probably worth your time to peruse the source code...)

```
matchPS :: PackedString    -- regexp
        -> PackedString    -- string to match
        -> [Char]          -- flags
        -> Maybe REmatch   -- info about what matched and where

searchPS :: PackedString    -- regexp
         -> PackedString    -- string to match
         -> [Char]          -- flags
         -> Maybe REmatch

-- Perl-like match-and-substitute:
substPS :: PackedString     -- regexp
        -> PackedString     -- replacement
        -> [Char]           -- flags
        -> PackedString     -- string
        -> PackedString

-- same as substPS, but no prefix and suffix:
replacePS :: PackedString  -- regexp
          -> PackedString  -- replacement
          -> [Char]        -- flags
          -> PackedString  -- string
          -> PackedString
```

```
match2PS :: PackedString   -- regexp
         -> PackedString   -- string1 to match
         -> PackedString   -- string2 to match
         -> [Char]         -- flags
         -> Maybe REmatch

search2PS :: PackedString  -- regexp
          -> PackedString  -- string to match
          -> PackedString  -- string to match
          -> [Char]        -- flags
          -> Maybe REmatch

-- functions to pull the matched pieces out of an REmatch:

getMatchesNo    :: REmatch -> Int
getMatchedGroup :: REmatch -> Int -> PackedString -> PackedString
getWholeMatch   :: REmatch -> PackedString -> PackedString
getLastMatch    :: REmatch -> PackedString -> PackedString
getAfterMatch   :: REmatch -> PackedString -> PackedString

-- (reverse) brute-force string matching;
-- Perl equivalent is index/rindex:
findPS, rfindPS :: PackedString -> PackedString -> Maybe Int

-- Equivalent to Perl "chop" (off the last character, if any):
chopPS :: PackedString -> PackedString

-- matchPrefixPS: tries to match as much as possible of strA starting
-- from the beginning of strB (handy when matching fancy literals in
-- parsers):
matchPrefixPS :: PackedString -> PackedString -> Int
```

8.3. `Parsec`: Parsing combinators

The `Parsec` library has been moved to the hierarchical libraries; it can be found in `Text.ParserCombinators.Parsec` (../base/Text.ParserCombinators.Parsec.html) in the `base` package.

8.4. `Pretty`: Pretty printing combimators

The `Pretty` library has been moved to the hierarchical libraries; it can be found in `Text.PrettyPrint.HughesPJ` (../base/Text.PrettyPrint.HughesPJ.html) in the `base` package.

8.5. Regex: The low-level regex matching interface

(Sigbjørn Finne supplied the regular-expressions interface.)

The Regex library provides quite direct interface to the GNU regular-expression library, for doing manipulation on PackedStrings (Section 4.24). You probably need to see the GNU documentation if you are operating at this level. Alternatively, you can use the simpler and higher-level RegexString (Section 8.6) interface.

The datatypes and functions that Regex provides are:

```
data PatBuffer  # just a bunch of bytes (mutable)

data REmatch
 = REmatch (Array Int GroupBounds)  -- for $1, ... $n
           GroupBounds              -- for $` (everything before match)
           GroupBounds              -- for $& (entire matched string)
           GroupBounds              -- for $' (everything after)
           GroupBounds              -- for $+ (matched by last bracket)

-- GroupBounds hold the interval where a group
-- matched inside a string, e.g.
--
-- matching "reg(exp)" "a regexp" returns the pair (5,7) for the
-- (exp) group. (PackedString indices start from 0)

type GroupBounds = (Int, Int)

re_compile_pattern
        :: PackedString         -- pattern to compile
        -> Bool                 -- True <=> assume single-line mode
        -> Bool                 -- True <=> case-insensitive
        -> IO PatBuffer

re_match :: PatBuffer           -- compiled regexp
         -> PackedString        -- string to match
         -> Int                 -- start position
         -> Bool                -- True <=> record results in registers
         -> IO (Maybe REmatch)

-- Matching on 2 strings is useful when you're dealing with multiple
-- buffers, which is something that could prove useful for
-- PackedStrings, as we don't want to stuff the contents of a file
-- into one massive heap chunk, but load (smaller chunks) on demand.

re_match2 :: PatBuffer          -- 2-string version
          -> PackedString
          -> PackedString
          -> Int
```

```
          -> Int
          -> Bool
          -> IO (Maybe REmatch)

re_search :: PatBuffer          -- compiled regexp
          -> PackedString       -- string to search
          -> Int                -- start index
          -> Int                -- stop index
          -> Bool               -- True <=> record results in registers
          -> IO (Maybe REmatch)

re_search2 :: PatBuffer         -- Double buffer search
           -> PackedString
           -> PackedString
           -> Int               -- start index
           -> Int               -- range (?)
           -> Int               -- stop index
           -> Bool              -- True <=> results in registers
           -> IO (Maybe REmatch)
```

8.6. `RegexString`: Regex matching made simple

The `RegexString` library has been moved to the hierarchical libraries; it can be found in `Text.Regex` (../base/Text.Regex.html) in the `base` package.

Chapter 9. The `util` package: miscellaneous utilities

9.1. `GetOpt`: Command line parsing

The `GetOpt` library has been moved to the hierarchical libraries; it can be found in `System.Console.GetOpt` (../base/System.Console.GetOpt.html) in the `base` package.

9.2. `Memo`: Fast memo functions

The `Memo` library provides fast polymorphic memo functions using hash tables. The interface is:

```
memo :: (a -> b) -> a -> b
```

So, for example, `memo f` is a version of `f` that caches the results of previous calls.

The searching is very fast, being based on pointer equality. One consequence of this is that the caching will only be effective if *exactly the same argument is passed again to the memoised function*. This means not just a copy of a previous argument, but the same instance. It's not useful to memoise integer functions using this interface, because integers are generally copied a lot and two instances of '27' are unlikely to refer to the same object.

This memoisation library works well when the keys are large (or even infinite).

The memo table implementation uses weak pointers and stable names (see the GHC/Hugs library document) to avoid space leaks and allow hashing for arbitrary Haskell objects. NOTE: while individual memo table entries will be garbage collected if the associated key becomes garbage, the memo table itself will not be collected if the function becomes garbage. We plan to fix this in a future version.

There's another version of `memo` if you want to explicitly give a size for the hash table (the default size is 1001 buckets):

```
memoSized :: Int -> (a -> b) -> a -> b
```

9.3. QuickCheck

The QuickCheck library has been moved to the hierarchical libraries; it can be found in Debug.QuickCheck (../base/Debug.QuickCheck.html) in the base package.

9.4. Readline: Command line editing

(Darren Moffat supplied the initial version of the Readline module.)

The Readline module is a straightforward interface to the GNU Readline library. As such, you will need to look at the GNU documentation (and have a libreadline.a file around somewhere...)

The main function you'll use is:

```
readline :: String{-the prompt-} -> IO (Maybe String)
```

If you want to mess around with Full Readline G(l)ory, we also provide:

```
type KeyCode = Char

type CallbackFunction =
    (Int ->     -- Numeric Argument
     KeyCode -> -- KeyCode of pressed Key
     IO Int)    -- What's this?

initialize      :: IO ()
addHistory      :: String -> IO ()
bindKey         :: KeyCode -> CallbackFunction -> IO ()
addDefun        :: String  -> CallbackFunction -> Maybe KeyCode -> IO ()

getReadlineName :: IO String
setReadlineName :: String -> IO ()
getLineBuffer   :: IO String
setLineBuffer   :: String -> IO ()
getPoint        :: IO Int
setPoint        :: Int -> IO ()
getEnd          :: IO Int
setEnd          :: Int -> IO ()
getMark         :: IO Int
setMark         :: Int -> IO ()
setDone         :: Bool -> IO ()
setPendingInput :: KeyCode -> IO ()
getPrompt       :: IO String
getTerminalName :: IO String

inStream        :: Handle
```

```
outStream        :: Handle
```

(All those names are just Haskellised versions of what you will see in the GNU readline documentation.)

9.5. Select: Synchronous I/O multiplexing

The `Select` interface provides a Haskell wrapper for the `select()` OS call supplied by many modern UNIX variants. `Select` exports the following:

```
type TimeOut = Maybe Int
  -- Nothing => wait indefinitely.
  -- Just x | x >= 0    => block waiting for 'x' micro seconds.
  --        | otherwise => block waiting for '-x' micro seconds.

hSelect :: [Handle]
        -> [Handle]
        -> [Handle]
        -> TimeOut
        -> IO SelectResult

type SelectResult
 = ( [Handle]  -- input  handles ready
   , [Handle]  -- output handles ready
   , [Handle]  -- exc.   handles ready
   )
```

Here's an example of how it could be used:

```
module Main(main) where

import Select
import IO

main :: IO ()
main = do
  hSetBuffering stdin NoBuffering
  putStrLn "waiting for input to appear"
  hSelect [stdin] [] [] Nothing
  putStrLn "input ready, let's try reading"
  x <- getChar
  print x
```

where the call to `hSelect` makes the process go to sleep until there's input available on `stdin`.

9.5.1. Using `hSelect` with Concurrent Haskell

In brief: don't. For two reasons:

- `hSelect` will cause all your Haskell threads to block until the `hSelect` returns, much like any call to a foreign function.
- You don't need to. Concurrent Haskell will let you do I/O on multiple file handles concurrently by forking threads, and if you need to assign a timeout, then this can be done using a combination of `threadDelay` (see) and asynchronous exceptions (see).

Chapter 10. The Win32 package

The win32 package is a thin and incomplete veneer over the Win32 API. Look at the source code to see what is available; the usage should be obvious from the Microsoft's C API documentation.