

pnm

Claude Heiland-Allen

2012–2015

Contents

1	Codec/PNM.hs	2
2	Codec/PNM/Parse.hs	5
3	.gitignore	8
4	LICENSE	8
5	pnm.cabal	9
6	pnmInfo.hs	9
7	pnmStrip.hs	9
8	Setup.hs	9

1 Codec/PNM.hs

```
{- |
Module      : Codec.PNM
Copyright   : (c) Claude Heiland-Allen 2012
License     : BSD3
5
Maintainer  : claud@mathr.co.uk
Stability   : provisional
Portability : portable

10 PNM image format header parsing and pretty printing.

PNM is PBM + PGM + PPM, a family of lowest common denominator image file formats ↴
↳ .

References: <http://netpbm.sourceforge.net/doc/pnm.html>.
15
-}
module Codec.PNM
  ( PNM(..), pnmRasterBytes, pnmPretty, Parse(..), pnmParse
    , pnms, unpnms, onPNMs, onPNMs'
20   ) where

import Prelude hiding ((++), (**), concat, splitAt)
import qualified Prelude as P

25 import Data.ByteString.Lazy (ByteString, uncons, empty, concat, splitAt)
import Data.Int (Int64)
import Codec.PNM.Parse

-- | PNM image headers.
30 data PNM
  = PBM{ pnmPlain :: Bool, pnmWidth, pnmHeight           :: Integer }
```

```

    | PGM{ pnmPlain :: Bool, pnmWidth, pnmHeight, pnmMaxVal :: Integer }
    | PPM{ pnmPlain :: Bool, pnmWidth, pnmHeight, pnmMaxVal :: Integer }
    deriving (Eq, Ord, Show)
35
-- | Compute the raster size in bytes for binary PNM images.
pnmRasterBytes :: PNM -> Maybe Integer
pnmRasterBytes pnm
    | pnmPlain pnm = Nothing
40    | otherwise = Just $ case pnm of
        PBM _ w h -> ((w + 7) `div` 8) * h
        PGM _ w h m
            | m < 256 -> w * h
            | otherwise -> 2 * w * h
45        PPM _ w h m
            | m < 256 -> 3 * w * h
            | otherwise -> 6 * w * h

-- | Pretty-print a PNM image header without any comments.
50 pnmPretty :: PNM -> ByteString
pnmPretty (PBM True w h) = str $ "P1\n" ** show w ** " " ** show h ** "\n"
pnmPretty (PBM False w h) = str $ "P4\n" ** show w ** " " ** show h ** "\n"
pnmPretty (PGM True w h m) = str $ "P2\n" ** show w ** " " ** show h ** "\n" ** ↵
    ↵ show m ** "\n"
pnmPretty (PGM False w h m) = str $ "P5\n" ** show w ** " " ** show h ** "\n" ** ↵
    ↵ show m ** "\n"
55 pnmPretty (PPM True w h m) = str $ "P3\n" ** show w ** " " ** show h ** "\n" ** ↵
    ↵ show m ** "\n"
pnmPretty (PPM False w h m) = str $ "P6\n" ** show w ** " " ** show h ** "\n" ** ↵
    ↵ show m ** "\n"

(**) :: String -> String -> String
(**) = (P.++)

60
-- | Parse a PNM image header.
pnmParse :: ByteString -> Parse PNM
pnmParse s = case uncons s of
    Nothing -> Empty
65    Just (p, ps)
        | p == 80{-P-} -> case uncons ps of
            Just (d, _)
                | d == d1 -> p1 s
                | d == d2 -> p2 s
70                | d == d3 -> p3 s
                | d == d4 -> p4 s
                | d == d5 -> p5 s
                | d == d6 -> p6 s
            _ -> Wrong s
75    _ -> Wrong s

p1, p2, p3, p4, p5, p6 :: ByteString -> Parse PNM

(p1, p4) = (pbm "P1" (PBM True), pbm "P4" (PBM False))
80 where
    pbm magic ctor s0 = case string (str magic) s0 of
        Empty -> Empty
        Parse () ps s1 -> case number s1 of
            Parse w ws s2 -> case number s2 of

```

```

85         Parse h hs s3 -> case oneSpace s3 of
            Parse () ss s4 -> Parse (ctor w h) (ps ++ ws ++ hs ++ ss) s4
            _ -> Wrong s0
            _ -> Wrong s0
            _ -> Wrong s0
90         _ -> Wrong s0

(p2, p3, p5, p6) = (ppm "P2" (PGM True), ppm "P3" (PPM True), ppm "P5" (PGM ↯
    ↯ False), ppm "P6" (PPM False))
where
    ppm magic ctor s0 = case string (str magic) s0 of
95         Empty -> Empty
        Parse () ps s1 -> case number s1 of
            Parse w ws s2 -> case number s2 of
                Parse h hs s3 -> case number s3 of
                    Parse m ms s4 | m < 65536 -> case oneSpace s4 of
100                     Parse () ss s5 -> Parse (ctor w h m) (ps ++ ws ++ hs ++ ms ++ ss) ↯
                        ↯ s5
                        _ -> Wrong s0
                        _ -> Wrong s0
                        _ -> Wrong s0
                        _ -> Wrong s0
105                     _ -> Wrong s0
                    _ -> Wrong s0

-- | Parse a sequence of binary PNM images.
--
-- Malformed input (including huge raster sizes or plain images)
110 -- is treated as end-of-image-stream.
--
pnms :: ByteString -> [(PNM, ByteString)]
pnms s = case pnmParse s of
    Parse pnm _ rest
115     | Nothing < bytes && bytes <= Just (toInteger (maxBound :: Int64)) ->
        (pnm, raster) : pnms next
    where
        ~(raster, next) = splitAt (fromInteger bytes') rest
        bytes@(~(Just bytes')) = pnmRasterBytes pnm
120     _ -> [] -- FIXME what to do about huge rasters or malformed input?

-- | Pretty-print a sequence of binary PNM images.
--
-- The precondition that the raster is of the correct length is not
125 -- checked, so malformed output is possible.
--
unpnms :: [(PNM, ByteString)] -> ByteString
unpnms = concat . map (\ ~(pnm, raster) -> pnmPretty pnm ++ raster)

130 -- | Process a sequence of binary PNM images.
--
-- Malformed input (including huge raster sizes or plain images)
-- is treated as end-of-image-stream.
--
135 -- The precondition that the raster is of the correct length is not
-- checked, so malformed output is possible.
--
-- Header comments are not preserved.
--

```

```

140 onPNMs :: (PNM -> ByteString -> (PNM, ByteString)) -> ByteString -> ByteString
onPNMs f = unpnmns . map (\ ~(pnm, raster) -> f pnm raster) . pnmns

-- | Process a sequence of binary PNM images.
--
145 -- Malformed input (including huge raster sizes or plain images)
-- is treated as end-of-image-stream.
--
-- The precondition that the raster is of the correct length is not
-- checked, so malformed output is possible.
150 --
-- Header comments are preserved. Assuming well-formed input:
--
-- > onPNMs' (\ - r -> r) = id
--
155 onPNMs' :: (PNM -> ByteString -> ByteString) -> ByteString -> ByteString
onPNMs' f s = case pnmParse s of
  Parse pnm header rest
    | Nothing < bytes && bytes <= Just (toInteger (maxBound :: Int64)) ->
      header ++ raster' ++ onPNMs' f next
160 where
  raster' = f pnm raster
  ~(raster, next) = splitAt (fromInteger bytes') rest
  bytes@ ~(Just bytes') = pnmRasterBytes pnm
  - -> empty -- FIXME what to do about huge rasters or malformed input?

```

2 Codec/PNM/Parse.hs

```

{- |
Module      : Codec.PNM.Parse
Copyright   : (c) Claude Heiland-Allen 2012
License     : BSD3

5
Maintainer  : claud@mathr.co.uk
Stability   : provisional
Portability : portable

10 Lower-level functions for parsing PNM image format headers. Most users
shouldn't need to import this module directly.

-}
module Codec.PNM.Parse where

15
import Prelude hiding ((++), dropWhile)
import Data.ByteString.Lazy (ByteString, append, empty, singleton, uncons, pack, \
  ↪ unpack)
import Data.Word (Word8)
import Data.Char (chr, ord)

20
-- | The result of parsing.
data Parse p
  = Wrong
    { parseMalformed :: ByteString
    }
25
  | Empty
  | Parse
    { parseResult :: p

```

```

    , parseRawResult :: ByteString
30    , parseRemainder :: ByteString
    }
    deriving (Eq, Ord, Show)

-- | The next character.
35 rawChar :: ByteString -> Parse Word8
rawChar s = case uncons s of
    Nothing -> Empty
    Just (c, cs) -> Parse c (singleton c) cs

40 -- | The next non-comment character. Comments can occur in the middle
--   of what might be considered tokens.
char :: ByteString -> Parse Word8
char s = case uncons s of
    Nothing -> Empty
45    Just (c, cs)
        -- handle comments: "# ... EOL" can occur anywhere, including mid-token.
        | isStartComment c -> case dropWhile (not . isEndComment) rawChar s of
            Wrong _ -> Wrong s
            Empty -> Wrong s
50            Parse () ds es -> case uncons es of
                Nothing -> Wrong s
                Just (f, fs)
                    | isEndComment f -> case char fs of
                        Wrong _ -> Wrong s
55                        Empty -> Wrong s
                        Parse g gs hs -> Parse g (ds ++ singleton f ++ gs) hs
                    | otherwise -> Wrong s -- should never happen
        | otherwise -> Parse c (singleton c) cs

60 -- | Drop input while a predicate holds.
dropWhile :: (Word8 -> Bool) -> (ByteString -> Parse Word8) -> ByteString -> ↯
    ↳ Parse ()
dropWhile p get s = case get s of
    Wrong _ -> Wrong s
    Empty -> Empty
65    Parse c cs ds
        | p c -> case dropWhile p get ds of
            Wrong _ -> Wrong s
            Empty -> Parse () cs ds
            Parse () es fs -> Parse () (cs ++ es) fs
70    | otherwise -> Parse () empty s

-- | Take input until a predicate holds.
takeUntil :: (Word8 -> Bool) -> (ByteString -> Parse Word8) -> ByteString -> ↯
    ↳ Parse ByteString
takeUntil p get s = case get s of
75    Wrong _ -> Wrong s
    Empty -> Empty
    Parse c cs ds
        | p c -> Parse empty empty s
        | otherwise -> case takeUntil p get ds of
80            Wrong _ -> Wrong s
            Empty -> Parse (singleton c) cs ds
            Parse e es fs -> Parse (singleton c ++ e) (cs ++ es) fs

```

```

-- | Parse a token.
85 token :: ByteString -> Parse ByteString
token s = case dropWhile isSpace char s of
    Wrong _ -> Wrong s
    Empty -> Empty
    Parse () cs ds -> case takeUntil isSpace char ds of
90     Wrong _ -> Wrong s
    Empty -> Wrong s
    Parse t ts rs -> Parse t (cs ++ ts) rs

-- | Parse a fixed string.
95 string :: ByteString -> ByteString -> Parse ()
string match s = case token s of
    Empty -> Empty
    Parse t ts rs | t == match -> Parse () ts rs
    _ -> Wrong s
100

-- | Parse a positive decimal number.
number :: ByteString -> Parse Integer
number s = case token s of
    Parse t ts rs -> case decimal t of
105     Just n -> Parse n ts rs
    _ -> Wrong s
    _ -> Wrong s

-- | Number conversion.
110 decimal :: ByteString -> Maybe Integer
decimal s = case unpack s of
    ts@(t:_)
        | all isDigit ts && t /= d0 -> case reads (map (chr . fromEnum) ts) of
            [(n, "")] -> Just n
115     _ -> Nothing -- should never happen?
    _ -> Nothing

-- | Parse a single space.
oneSpace :: ByteString -> Parse ()
120 oneSpace s = case char s of
    Parse c cs rs | isSpace c -> Parse () cs rs
    _ -> Wrong s

-- | Convert from a string. Crashes hard on non-ASCII input.
125 str :: String -> ByteString
str = pack . map (toEnum . ord) -- toEnum crashes on out of bounds

-- | Character classes.
isSpace, isDigit, isStartComment, isEndComment :: Word8 -> Bool
130 isSpace c = c `elem` [ht, lf, vt, ff, cr, space]
isDigit c = c `elem` [d0, d1, d2, d3, d4, d5, d6, d7, d8, d9]
isStartComment c = c == hash
isEndComment c = c == lf || c == cr

135 -- | White space characters.
ht, lf, vt, ff, cr, space :: Word8
ht = 9
lf = 10
vt = 11
140 ff = 12

```

```

cr      = 13
space = 32

-- | Comment start character.
145 hash :: Word8
    hash = 35

-- | Decimal digit characters.
d0, d1, d2, d3, d4, d5, d6, d7, d8, d9 :: Word8
150 d0 = 48
    d1 = 49
    d2 = 50
    d3 = 51
    d4 = 52
155 d5 = 53
    d6 = 54
    d7 = 55
    d8 = 56
    d9 = 57
160
-- | Alias for 'append'.
(+++) :: ByteString -> ByteString -> ByteString
(+++) = append

```

3 .gitignore

dist

4 LICENSE

Copyright (c) 2012, Claude Heiland-Allen

All rights reserved.

```

5  Redistribution and use in source and binary forms, with or without
   modification, are permitted provided that the following conditions are met:

   * Redistributions of source code must retain the above copyright
     notice, this list of conditions and the following disclaimer.
10  * Redistributions in binary form must reproduce the above
     copyright notice, this list of conditions and the following
     disclaimer in the documentation and/or other materials provided
     with the distribution.

15  * Neither the name of Claude Heiland-Allen nor the names of other
     contributors may be used to endorse or promote products derived
     from this software without specific prior written permission.

20  THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS
   "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT
   LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR
   A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT
   OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL,
25  SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT
   LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE,
   DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY

```


THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT
 (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE
 30 OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5 pnm.cabal

```

name:                pnm
version:             0.1.0.0
synopsis:            PNM image format header parsing and pretty printing
description:
5   PNM image format header parsing and pretty printing. PNM is PBM + PGM + PPM,
    a family of lowest common denominator image file formats. This library makes
    no attempt to interpret raster data, instead providing a lazy ByteString.

license:            BSD3
10  license-file:    LICENSE
author:            Claude Heiland-Allen
maintainer:        claud@mathr.co.uk
category:          Graphics
build-type:        Simple
15  cabal-version:   >=1.8

library
  exposed-modules:   Codec.PNM, Codec.PNM.Parse
  build-depends:     base < 5, bytestring
20
source-repository head
  type:             git
  location:          http://code.mathr.co.uk/pnm.git

25  source-repository this
  type:             git
  location:          http://code.mathr.co.uk/pnm.git
  tag:              v0.1.0.0

```

6 pnminfo.hs

```

import Codec.PNM (pnms)
import Codec.PNM.Parse (str)
import qualified Data.ByteString.Lazy as BS
main = BS.interact (BS.concat . map (str . (++ "\n") . show . fst) . pnms)

```

7 pnmstrip.hs

```

import Codec.PNM (onPNMs)
import qualified Data.ByteString.Lazy as BS
main = BS.interact (onPNMs (curry id))

```

8 Setup.hs

```

import Distribution.Simple
main = defaultMain

```