nix-info [1]

Eric Bailey

March 18, 2017

[1] a brew info clone for Nix.

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special contents, but the length of words should match the language.

The reason to use Haskell for `nix-info` is so we can have strong, static typing.

```
Hello, here is some text without a meaning. This text
should show what a printed text will look like at this
place. If you read this text, you will get no information.
Really? Is there no information? Is there a difference
between this text and some nonsense like “Huardest
gefburn”? Kjift - not at all! A blind text like this gives
you information about the selected font, how the letters
are written and an impression of the look. This text should
contain all letters of the alphabet and it should be
written in of the original language. There is no need for
special contents, but the length of words should match the
language.
```

Data Types

⟨*Data Types* 1⟩≡ (9 17)
- ⟨*Meta* 2⟩ — `Meta` — "standard meta-attributes" [Con17]
- ⟨*PackageInfo* 3⟩ — `PackageInfo` — `name`, `system` and `meta`
- ⟨*Package* 4⟩ — `Package` — `path` and `info`
- ⟨*PackageList* 5⟩ — `PackageList` — `[Package]`
- ⟨*NixURL* 6⟩ — `NixURL` — `URL`

The standard meta-attributes are documented in the Nixpkgs Contributors Guide [Con17]. nix-env, which is called by nix-info in ⟨*nixQuery* 13⟩, returns a nested **Object**[2], with relationships as described by the following diagram.

[2] http://hackage.haskell.org/package/aeson-1.1.1.0/docs/Data-Aeson-Types.html#t:Object

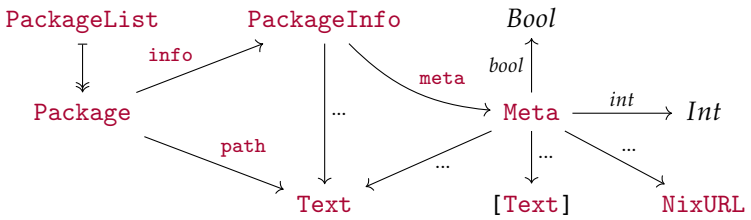

Figure 1: whatever

Flesh this out.

use better types than just **Text** everywhere . . .

⟨*Meta* 2⟩≡ (1)

```
data Meta = Meta
  { description      :: Maybe Text
  , longDescription  :: Maybe Text
  , branch           :: Maybe Text
  , homepage         :: Maybe NixURL
  , downloadPage     :: Maybe NixURL
  , maintainers      :: Maybe [Text]
  , priority         :: Maybe Int
  , platforms        :: Maybe [Text]
  , hydraPlatforms   :: Maybe [Text]
  , broken           :: Maybe Bool
  , updateWalker     :: Maybe Bool
  , outputsToInstall :: Maybe [Text]
  , position         :: Maybe Text
  }
  deriving (Show)
```

describe this

3 ⟨*PackageInfo* 3⟩≡ (1)

```
data PackageInfo = PackageInfo
  { name   :: Text
  , system :: Text
  , meta   :: Meta
  }
  deriving (Show)
```

describe this

4 ⟨*Package* 4⟩≡ (1)

```
data Package = Package
  { path :: Text
  , info :: PackageInfo
  }
  deriving (Show)
```

This **newtype** is a cheap trick to avoid using **FlexibleInstances** for our ⟨*FromJSON Instances* 8⟩.

describe why

5 ⟨*PackageList* 5⟩≡ (1)

```
newtype PackageList = PackageList [Package]
```

Mention the avoidance of the orphan instance.

6 ⟨*NixURL* 6⟩≡ (1)

```
newtype NixURL = NixURL URL deriving (Show)
```

describe this

7 ⟨*magically derive ToJSON and FromJSON instances* 7⟩≡ (8)

```
$(deriveJSON defaultOptions ''NixURL)

$(deriveJSON defaultOptions ''Meta)

$(deriveJSON defaultOptions ''PackageInfo)
```

describe this

8 ⟨*FromJSON Instances* 8⟩≡ (9 17)
⟨*magically derive ToJSON and FromJSON instances* 7⟩

```
instance FromJSON PackageList where
  parseJSON (Object v) =
    PackageList <$> traverse (\(p,y) -> Package p <$> parseJSON y) (HM.toList v)
  parseJSON _          = fail "non-object"

instance FromJSON NixURL where
  parseJSON (String t) = case importURL (T.unpack t) of
                           Just url -> pure $ NixURL url
                           Nothing  -> fail "no parse"
  parseJSON _          = fail "non-string"
```

⟨*src/NixInfo/Types.hs* 9⟩≡
```
- |
- Module      : NixInfo.Types
- Copyright   : (c) 2017, Eric Bailey
- License     : BSD-style (see LICENSE)
-
- Maintainer  : eric@ericb.me
- Stability   : experimental
- Portability : portable
-
- Data types and JSON parsers for nix-info
```

⟨*OverloadedStrings* 18⟩
⟨*TemplateHaskell* 19⟩

```
module NixInfo.Types where
```

⟨*NixInfo.Types Imports* 23⟩

⟨*Data Types* 1⟩

⟨*FromJSON Instances* 8⟩

Helper Functions

⟨*src/NixInfo.hs* 10⟩≡
```
- |
- Module      : NixInfo
- Copyright   : (c) 2017, Eric Bailey
- License     : BSD-style (see LICENSE)
-
- Maintainer  : eric@ericb.me
- Stability   : experimental
- Portability : portable
-
- brew info clone for Nix

module NixInfo (printPackage) where

import           NixInfo.Types
```

⟨*hide Prelude.putStrLn* 20⟩

⟨*import traverse_, catMaybes* 25⟩

```
import qualified Data.Text              as T
```
⟨*import Data.Text.IO* 24⟩

⟨*printPackage* 11⟩

11 ⟨*printPackage* 11⟩≡ (10 17)

```
- printPackage :: MonadIO io => Package -> io ()
printPackage :: Package -> IO ()
printPackage (Package pkgPath (PackageInfo pkgName _pkgSystem pkgMeta)) =
  traverse_ putStrLn $
  catMaybes
  [ Just pkgName
  - , Just pkgSystem
  , description pkgMeta
  , homepage pkgMeta
  - , T.unwords . T.words <$> longDescription pkgMeta
  , T.unwords <$> maintainers pkgMeta
  - , T.unwords <$> outputsToInstall pkgMeta
  - , T.unwords <$> platforms pkgMeta
  , Just pkgPath
  , position pkgMeta
  ]
```

Main Executable

12 ⟨*main* 12⟩≡ (14 17)

```
main :: IO ()
main =
  sh $ arguments »= \case
  [arg] -> nixQuery arg »= \case
           Just (PackageList pkgs) -> liftIO $ traverse_ printPackage pkgs
           Nothing                 -> exit $ ExitFailure 1
  _     -> do echo "TODO: usage"
              exit $ ExitFailure 1
```

13 ⟨*nixQuery* 13⟩≡ (14 17)

```
nixQuery :: Text -> Shell (Maybe PackageList)
nixQuery arg =
  procStrict "nix-env" ["-qa", arg, "-json" ] empty »= \case
  (ExitSuccess,txt) -> pure $ decode (cs txt)
  (status,_)        -> exit status
```

⟨*app/Main.hs* 14⟩≡
```
- |
- Module      : Main
- Copyright   : (c) 2017, Eric Bailey
- License     : BSD-style (see LICENSE)
-
- Maintainer  : eric@ericb.me
- Stability   : experimental
- Portability : portable
-
- Main executable for nix-info.

⟨LambdaCase 16⟩
⟨OverloadedStrings 18⟩

module Main (main) where

import           NixInfo                (printPackage)
import           NixInfo.Types

⟨import Data.Aeson 21⟩
import           Data.Foldable          (traverse_)
import           Data.String.Conversions (cs)
import           Data.Text              (Text)

⟨import Turtle 26⟩

⟨nixQuery 13⟩

⟨main 12⟩
```

As a Script

⟨*shebang* 15⟩≡ (17)
```
#! /usr/bin/env nix-shell
#! nix-shell -i runhaskell -p "haskellPackages.ghcWithPackages (h: [ h.turtle h.aeson h.string-conversions h.url ])"
```

⟨*LambdaCase* 16⟩≡ (14 17)
```
{-# LANGUAGE LambdaCase        #-}
```

17 ⟨*script/nix-info* 17⟩≡
⟨*shebang* 15⟩

⟨*LambdaCase* 16⟩

⟨*OverloadedStrings* 18⟩
⟨*TemplateHaskell* 19⟩

```
module Main (main) where
```

⟨*hide Prelude.putStrLn* 20⟩

⟨*NixInfo.Types Imports* 23⟩

⟨*import traverse_, catMaybes* 25⟩
```
import           Data.String.Conversions (cs)
```

⟨*import Data.Text.IO* 24⟩

⟨*import Turtle* 26⟩

⟨*Data Types* 1⟩

⟨*FromJSON Instances* 8⟩

⟨*printPackage* 11⟩

⟨*nixQuery* 13⟩

⟨*main* 12⟩

Language Extensions

To manage juggling **Text**[3], (lazy) **ByteString**[4], and **Line**[5] values, use the ⟨*OverloadedStrings* 18⟩ language extension [Cha14]. Enable the ⟨*TemplateHaskell* 19⟩ language extension [Wes14] to ⟨*magically derive ToJSON and FromJSON instances* 7⟩ from record definitions via **Data.Aeson.TH**[6]

18 ⟨*OverloadedStrings* 18⟩≡ (9 14 17)
```
{-# LANGUAGE OverloadedStrings #-}
```

19 ⟨*TemplateHaskell* 19⟩≡ (9 17)
```
{-# LANGUAGE TemplateHaskell   #-}
```

[3] https://hackage.haskell.org/package/text/docs/Data-Text.html#t:Text
[4] https://hackage.haskell.org/package/bytestring/docs/Data-ByteString.html#t:ByteString
[5] https://hackage.haskell.org/package/turtle-1.3.2/docs/Turtle-Line.html#t:Line
[6] https://hackage.haskell.org/package/aeson-1.1.1.0/docs/Data-Aeson-TH.html

Imports

Hide `Prelude.putStrLn`[7], so we can ⟨*import Data.Text.IO* 24⟩ `(putStrLn)`[8].

[7] https://hackage.haskell.org/package/base/docs/Prelude.html#v:putStrLn

[8] https://hackage.haskell.org/package/text/docs/Data-Text-IO.html#v:putStrLn

⟨*hide Prelude.putStrLn* 20⟩≡ (10 17)
```
import          Prelude                 hiding (putStrLn)
```

⟨*import Data.Aeson* 21⟩≡ (14 23)
```
import          Data.Aeson
```

⟨*import Data.Aeson.TH* 22⟩≡ (23)
```
import          Data.Aeson.TH           (defaultOptions, deriveJSON)
```

⟨*NixInfo.Types Imports* 23⟩≡ (9 17)
⟨*import Data.Aeson* 21⟩
⟨*import Data.Aeson.TH* 22⟩
```
import qualified Data.HashMap.Lazy       as HM
import          Data.Text               (Text)
import qualified Data.Text               as T

import          Network.URL             (URL, importURL)
```

⟨*import Data.Text.IO* 24⟩≡ (10 17)
```
import          Data.Text.IO            (putStrLn)
```

⟨*import traverse_, catMaybes* 25⟩≡ (10 17)
```
import          Data.Foldable           (traverse_)
import          Data.Maybe              (catMaybes)
```

⟨*import Turtle* 26⟩≡ (14 17)
```
import          Turtle                  (ExitCode (..), Shell, arguments,
                                         echo, empty, exit, liftIO,
                                         procStrict, sh)
```

Other Files

⟨*Setup.hs* 27⟩≡
```
import Distribution.Simple

main :: IO ()
main = defaultMain
```

Chunks

Index

References

[Cha14] Oliver Charles. *24 Days of GHC Extensions: Overloaded Strings*. Dec. 17, 2014. URL: https://ocharles.org.uk/blog/posts/2014-12-17-overloaded-strings.html (visited on 03/18/2017).

[Con17] Nix Contributors. *Nixpkgs Contributors Guide*. 2017. URL: https://nixos.org/nixpkgs/manual/#sec-standard-meta-attributes (visited on 03/18/2017).

[Wes14] Sean Westfall. *24 Days of GHC Extensions: Template Haskell*. Dec. 22, 2014. URL: https://ocharles.org.uk/blog/guest-posts/2014-12-22-template-haskell.html (visited on 03/18/2017).