nix-info [1]

Eric Bailey

March 18, 2017

[1] a brew info clone for Nix.

write abstract

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

flesh this out

Language Extensions

Since we're going to be juggling **Text**, (lazy) **ByteString**, and **Line**, we use the **OverloadedStrings** extension [Cha14].

add links to data types

1 ⟨*OverloadedStrings* 1⟩≡ (11 17 20)

```
{-# LANGUAGE OverloadedStrings #-}
```

For generic **Aeson** ⟨*magic* 9⟩, we use TemplateHaskell.

cite magic

2 ⟨*TemplateHaskell* 2⟩≡ (11 20)

flesh this out

```
{-# LANGUAGE TemplateHaskell   #-}
```

Do you believe in ⟨*magic* 9⟩?

Data Types

3 ⟨*NixInfo.Types Imports* 3⟩≡ (11 20)

```
import           Data.Aeson
import           Data.Aeson.TH     (defaultOptions, deriveJSON)

import qualified Data.HashMap.Lazy as HM
import           Data.Text         (Text)
```

Data Types

4 ⟨*Data Types* 4⟩≡ (11 20)

```
- ---------------------------------------- [ Data Types ]
```

⟨*Meta* 5⟩

⟨*PackageInfo* 6⟩

⟨*Package* 7⟩

⟨*PackageList* 8⟩

asd

The standard meta-attributes are documented in the Nixpkgs Contributors Guide [Con17].

flesh this out

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

⟨*Meta* 5⟩≡ (4)

```
data Meta = Meta
  { description      :: Maybe Text
  , longDescription  :: Maybe Text
  , branch           :: Maybe Text
  , homepage         :: Maybe Text
  , downloadPage     :: Maybe Text
  , 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

⟨*PackageInfo* 6⟩≡ (4)

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

describe this

⟨*Package* 7⟩≡ (4)

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

This `newtype` is a cheap trick to avoid using `FlexibleInstances` for our ⟨*FromJSON Instances* 10⟩.

describe why

⟨*PackageList* 8⟩≡ (4)

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

describe this

9 ⟨*magic* 9⟩≡ (10)

```
$(deriveJSON defaultOptions ”Meta)

$(deriveJSON defaultOptions ”PackageInfo)
```

describe this

10 ⟨*FromJSON Instances* 10⟩≡ (11 20)

```
- ------------------------------------ [ FromJSON Instances ]

⟨magic 9⟩

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

11 ⟨*src/NixInfo/Types.hs* 11⟩≡

```
- ------------------------------------------- [ Types.hs ]
- |
- 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
------------------------------------------------ [ EOH ]
⟨OverloadedStrings 1⟩
⟨TemplateHaskell 2⟩

module NixInfo.Types where

⟨NixInfo.Types Imports 3⟩

⟨Data Types 4⟩

⟨FromJSON Instances 10⟩

- ------------------------------------------------ [ EOF ]
```

Helper Functions

12 ⟨*Text IO Imports* 12⟩≡ (13 20)

```
import qualified Data.Text      as T
import           Data.Text.IO   (putStrLn)
```

⟨*src/NixInfo.hs* 13⟩≡

```
- ---------------------------------------- [ NixInfo.hs ]
- |
- 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
------------------------------------------------ [ EOH ]
module NixInfo (printPackage) where

import           NixInfo.Types

import           Prelude        ()
import           Prelude.Compat hiding (putStrLn)

import           Data.Foldable  (traverse_)
import           Data.Maybe     (catMaybes)
```

⟨*Text IO Imports* 12⟩

⟨*printPackage :: Package -> IO ()* 14⟩

```
- ------------------------------------------------ [ EOF ]
```

⟨*printPackage :: Package -> IO ()* 14⟩≡ (13 20)

```
- 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

15 ⟨*main :: IO ()* 15⟩≡ (17 20)

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

16 ⟨*nixQuery :: Text -> Shell (Maybe PackageList)* 16⟩≡ (17 20)

```
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* 17⟩≡

```
- ------------------------------------------ [ Main.hs ]
- |
- 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.
------------------------------------------------ [ EOH ]
```

⟨*LambdaCase* 19⟩
⟨*OverloadedStrings* 1⟩

```
module Main (main) where

import           NixInfo                (printPackage)
import           NixInfo.Types

import           Control.Applicative    (empty)

import           Data.Aeson             (decode)
import           Data.Foldable          (for_)
import           Data.String.Conversions (cs)
import           Data.Text              (Text)

import           Turtle                 (ExitCode (..), Shell, arguments, echo,
                                         exit, liftIO, procStrict, sh)

- -------------------------------------- [ Private Parts ]
```

⟨*nixQuery :: Text -> Shell (Maybe PackageList)* 16⟩

```
- ------------------------------------------- [ Main ]
```

⟨*main :: IO ()* 15⟩

```
- ------------------------------------------- [ EOF ]
```

As a Script

⟨*shebang* 18⟩≡ (20)

```
#! /usr/bin/env nix-shell
#! nix-shell -i runhaskell -p "haskellPackages.ghcWithPackages (h: [ h.turtle h.aeson h.string-conversions ])"
```

⟨*LambdaCase* 19⟩≡ (17 20)

```
{-# LANGUAGE LambdaCase        #-}
```

20 ⟨*script/nix-info* 20⟩≡
⟨*shebang* 18⟩

⟨*LambdaCase* 19⟩

⟨*OverloadedStrings* 1⟩
⟨*TemplateHaskell* 2⟩

```
module Main (main) where

import           Prelude                 ()
import           Prelude.Compat          hiding (putStrLn)

import           Control.Applicative     (empty)
```

⟨*NixInfo.Types Imports* 3⟩

```
import           Data.Foldable           (for_, traverse_)
import           Data.Maybe              (catMaybes)
import           Data.String.Conversions (cs)
```

⟨*Text IO Imports* 12⟩

```
import           Turtle                  (ExitCode (..), Shell, arguments, echo,
                                          exit, liftIO, procStrict, sh)
```

⟨*Data Types* 4⟩

⟨*FromJSON Instances* 10⟩

```
- ------------------------------------ [ Helper Functions ]
```

⟨*printPackage :: Package -> IO ()* 14⟩

⟨*nixQuery :: Text -> Shell (Maybe PackageList)* 16⟩

```
- ------------------------------------------- [ Main ]
```

⟨*main :: IO ()* 15⟩

```
- ---------------------------------------------- [ EOF ]
```

Other Files

21 ⟨*Setup.hs* 21⟩≡

```
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).