Implementing Backpack in GHC

Edward Z. Yang <ezyang@cs.stanford.edu>

Simon Peyton Jones    Scott Kilpatrick
Simon Marlow        Derek Dreyer    Duncan Coutts
Why should you care about module systems?
But Edward, that’s a type class!

Have you ever tried to use a type class for this use-case?
$ cabal install hledger-0.18
$ cabal install hledger-0.18
Resolving dependencies...
cabal: Could not resolve dependencies:
rejecting: hledger-0.19.4/installed-402..., 0.18.2, 0.18.1 (global constraint requires =)
trying: hledger-0.18
trying: regexpr-0.5.4/installed-0da...
trying: process-1.1.0.2/installed-7b6...
rejecting: haskeline-0.7.0.3/installed-414..., 0.7.0.0 (conflict: hledger => haskeline==0.6)
rejecting: haskeline-0.6.4.7 (conflict: process_unix==2.6.0.1/installed-ccb..., haskeline =>
rejecting: haskeline-0.6.4.6 (conflict: regexpr
rej...
$ cabal install hledger-0.18
Resolving dependencies...
Downloading hledger-0.18...
Configuring hledger-0.18...
Building hledger-0.18...

Hledger/Cli/Options.hs:20:7:
  Couldn't match expected type `LibType' with
  In the first argument of `libfoo' name
Failed to install hledger-0.18
Yes, Stable Distributions help, but we also need tools to wrangle the wild-wild west.
Backpack

a new module system for Haskell
there a lot of things I could talk about...
but let me tell you how it works...
did you spend too long on the intro again?
module Utils where
  import Data.Text
  blank = putStrLn empty

what a useless function!

module Data.Text where
  data Text = ...
  empty = Text ...
  putStrLn s = ...
name: utils
build-depends:
  text
exposed-modules:
  Utils

name: text
exposed-modules:
  Data.Text
  Data.Text.Lazy
name: utils
build-depends:
  text
exposed-modules:
  Utils

hard-coded!
module Data.Text where

data Text

empty :: Text

putStrLn :: Text -> IO ()
[1 of 1] Checking Utils (Utils.hs, nothing)
Mix-in linking
configuration by convention

name: utils-strict
build-depends:
  utils,
text

or you could write a DSL
Mix-in linking
configuration by convention

name: utils-strict
build-depends:
  utils,
  text
reexported-modules:
  Utils as Utils.Strict
Mix-in linking
configuration by convention

name: utils-strict
build-depends:
  utils,
text
reexported-modules:
  Utils as Utils.Strict
Mix-in linking
configuration by convention

name: utils-lazy
build-depends:
  utils,
  text (Data.Text.Lazy as Data.Text)
reexported-modules:
  Utils as Utils.Lazy
name: my-app
build-depends:
  utils
    (Data.Text, 
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text
name: my-app
build-depends:
  utils
    (Data.Text,
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text

include multiple times
name: my-app
build-depends:
  utils
    (Data.Text,
     Utils as Utils.Strict;
     Data.Text as Data.Text.Lazy
     Utils as Utils.Lazy),
  text

holes can be renamed too!
module Utils.Strict
blank :: IO ()

module Utils.Lazy
blank :: IO ()
name: utils-sig
exported-signatures:
  Utils

module Utils
blank :: IO ()
name: utils
exported-modules:
  Utils
implements: utils-sig
build-requires:
  text-sig
name: utils
exported-modules:
  utils
implements: utils-sig-3
build-requires:
  text-sig-0.11
ok... so what did we just see?
Signature modules
mix-in linking
reexported modules
module renaming
multiple package instances
Signature modules
mix-in linking
reexported modules
module renaming
multiple package instances
Signature modules
mix-in linking
reexported modules
module renaming
multiple package instances

#9252
(Cabal)
#8407
#9375
#9267
on the docket

type classes/families

shaping

lots of refactoring
on the docket

type classes/families
Shaping
lots of refactoring
and you!
Thanks!

- module renaming
- reexported modules
- signature package
- holes
- mix-in linking

See GHC HEAD and blog.ezyang.com for more details.