# Using the GHC API

Carlo Hamalainen

carlo@carlo-hamalainen.net

February 2016

# Motivation

- "Haskell in the Financial District: an Experience Report" (BFPG May 2013 – Sam Roberts)
- Sam said something like:

    *"People learned Haskell (Mu) in a Visual Studio environment; those who used the editor tools learned quicker than those who did not."*

## Motivation

Seems to make sense!

- ▶ Haskell has a great type system. It does a lot.
- ▶ Programming is hard.
- ▶ My brain is small.
- ▶ I'll write a Vim plugin to help!
- ▶ Profit?

# What is this symbol?

- You see `mapM_`. What is it?
- Hoogle says:

      mapM_ :: Monad m => (a -> m b) -> [a] -> m ()
      base Prelude, base Control.Monad

      mapM_ :: (Foldable t, Monad m) => (a -> m b)
                                      -> t a -> m ()
      base Data.Foldable

- Hoogle doesn't mention:

      mapM_ :: Monad m => (a -> m ()) -> Consumer a m ()
      Data.Conduit.List

- Not to mention local packages that are not on Hoogle/Hackage!
- Simple idea: use ghc-mod to run :info on a symbol.

## Use :info

```
*Demo> :info mapM_
mapM_ :: Monad m => (a -> m b) -> [a] -> m ()
    -- Defined in Control.Monad

*Demo> :info map
map :: (a -> b) -> [a] -> [b]  -- Defined in GHC.Base
```

Now find the package that exports Control.Monad:

http://hackage.haskell.org/package/base-4.6.0.1/docs/Control-Monad.html

But I had locally built documentation:

/opt/ghc-7.8.4_build/share/doc/ghc/html/libraries/base-4.7.0.2/Control-Monad.html

## :info gives defined in, not exported from

- ▶ module Hiding where

  ```
  import Data.List hiding (map)

  m = map (+1) [1, 2, 3]
  h = head [1, 2, 3]
  ```
- ▶ In ghci:

  ```
  *Hiding> :info map
  map :: (a -> b) -> [a] -> [b]  -- Defined in GHC.Base

  *Hiding> :info head
  head :: [a] -> a  -- Defined in GHC.List
  ```
- ▶ No page for GHC.Base (it's internal).
- ▶ GHC.List isn't quite right, it should be Data.List.

- Imports in a module, e.g. `import Data.List`
- Names in a module, e.g. `head`
- Module that a name is *imported from*, not defined in.
- Haddock URL to the module where the symbol is imported from.

# Load a module with the GHC API

```
main = runGhc (Just libdir) $ do
  dflags <- getSessionDynFlags
  setSessionDynFlags dflags

  GhcMonad.liftIO $ Packages.initPackages dflags

  target <- guessTarget "Hiding.hs" Nothing
  setTargets [target]
  load LoadAllTargets

  modSummary <- getModSummary (mkModuleName "Hiding")
  p <- parseModule modSummary    :: Ghc ParsedModule
  t <- typecheckModule p         :: Ghc TypecheckedModule
  d <- desugarModule t           :: Ghc DesugaredModule
```

# Dump the guts!

```
let guts = coreModule d               :: ModGuts
    gre  = HscTypes.mg_rdr_env guts :: GlobalRdrEnv

GhcMonad.liftIO $ putStrLn $ showSDoc dflags (ppr gre)
```

## The guts!

```
ihZ :-> [GHC.Base.map
         imported from Prelude at Hiding.hs:1:8-13
         (and originally defined in base:GHC.Base)],

ik7 :-> [GHC.List.head
         imported from Data.List at Hiding.hs:3:1-29
         (and originally defined in base:GHC.List)],
```

1. http://hackage.haskell.org/package/base-4.8.0.0/docs/Prelude.html
2. http://hackage.haskell.org/package/base-4.8.0.0/docs/Data-List.html

# The lookup process

Input: `head` at (11, 17) in `Demo.hs`

1. Partially compile `Demo.hs`: list of qualified names (e.g. `GHC.List.head`).
2. Match `head` to `GHC.List.head` using heuristics. Module load order!?
3. Discover `GHC.List.head` imported from `Data.List`.
4. `ghc-pkg find-module Data.List --simple-output --global --user`
5. Package could be `haskell98-2.0.0.3` or `haskell2010-1.1.2.0`.
6. Final answer could be

   `file:///home/carlo/opt/ghc-7.8.4_build/share/doc/ghc/html/libraries/haskell98-2.0.0.3/Prelude.html`

   or

   `file:///home/carlo/opt/ghc-7.8.4_build/share/doc/ghc/html/libraries/haskell2010-1.1.2.0/Data-List.html`

# DL.length



```
File  Edit  View  Search  Terminal  Help
import Data.Maybe
import qualified Data.List as DL
import qualified Data.Map as DM
import qualified Safe
-- import Data.List hiding (map)

f :: a -> Maybe a
f x = Just x

g :: IO ()
g = do
    let (Just _, _) = (Just 3, Just 4)

    return ()

s = "boo" :: String
s' = head s
t = Just 100 :: Maybe Int
r = DL.length [1, 2, 3]
file:///opt/ghc/7.10.3/share/doc/ghc/html/libraries/base-4.8.2.0/Data-List.html
Press ENTER or type command to continue
```

# Just



```
module Muddle where

import Data.Maybe
import qualified Data.List as DL
import qualified Data.Map as DM
import qualified Safe
-- import Data.List hiding (map)

f :: a -> Maybe a
f x = Just x

g :: IO ()
g = do
    let (Just _, _) = (Just 3, Just 4)

    return ()

s = "boo" :: String
s' = head s
```
file:///opt/ghc/7.10.3/share/doc/ghc/html/libraries/base-4.8.2.0/Prelude.html
Press ENTER or type command to continue

# Safe

```
File  Edit  View  Search  Terminal  Help
g :: IO ()
g = do
    let (Just _, _) = (Just 3, Just 4)

    return ()

s = "boo" :: String
s' = head s
t = Just 100 :: Maybe Int
r = DL.length [1, 2, 3]

main = print "Hello, World!"

h = DM.fromList [("x", "y")]

sh = Safe.headMay []

i = 3 :: Int
i' = 3 :: Integer
https://hackage.haskell.org/package/safe-0.3.9/docs/Safe.html        29,13              Bot
```

# TP.upper

```
File  Edit  View  Search  Terminal  Help
parseHaskellModuleName :: TP.ParsecT String u Data.Functor.Identity.Identity String
parseHaskellModuleName = do
    c <- TP.upper
    cs <- TP.many (TP.choice [TP.lower, TP.upper, TP.char '_', TP.digit])
    return (c:cs)

parseDottedHaskellModuleName :: TP.ParsecT String u Data.Functor.Identity.Identity S
tring
parseDottedHaskellModuleName = do
    TP.char '.'
    cs <- parseHaskellModuleName
    return cs

parseFullHaskellModuleName :: TP.ParsecT String u Data.Functor.Identity.Identity Str
ing
parseFullHaskellModuleName = do
    h <- parseHaskellModuleName
    rest <- many parseDottedHaskellModuleName
https://hackage.haskell.org/package/parsec-3.1.9/docs/Text-Parsec.html
Press ENTER or type command to continue
```

# Conclusion

- Useful for me, especially when using Yesod.
- Useful for others (e.g. Emacs plugin contributed).
- Has some corner cases - Haskell module system more complicated than I realised.
- GHC API wasn't too hard to use. Use ghc-mod and glue things together.
- Latest GHC API has plugins? Might help?

## Links

https://github.com/carlohamalainen/ghc-imported-from

https://github.com/carlohamalainen/ghcimportedfrom-vim

https://github.com/david-christiansen/ghc-imported-from-el

http://www.mew.org/~kazu/proj/ghc-mod/en