Haskell Development with Nix, GHCJS and Miso

Moritz Kiefer (@cocreature)

July 9, 2018

Terminology

Nix

Language and package manager

NixOS

Linux distribution built on top of Nix

nixpkgs

Official repository for nix expressions

The Life of a Package

- Everything happens in the Nix *store* (`/nix/store`)
- Nix *expressions* produce Nix *derivations*
- A Nix derivation consists of
  - Input derivations
  - Output paths
  - Build instructions
- *Realising* a derivation ensures that the output paths are valid
  - Either by fetching them from a *substitute*
  - or by building the derivation

Nix Language

Basic Values

- Booleans
- Integers
- Floating points
- Null
- Paths

Strings

Normal String Literals

```
"foo bar"
```

Antiquotation

```
"foo ${var} bar"
```

Indented Strings

```
''
    foo
    bar
''
```

Lists

- Enclosed in square brackets
- Items separated by spaces

Example

```
[ 123 ./foo.nix "abc" ]
```

Sets

Example

```
{ a = 1; b = 5; }
```

Attribute Selection

```
{ a = 1; b = 5; }.a
```

Recursive Sets

```
rec {
  x = y;
  y = 123;
}
```

`let`-expressions

Example

```
let
  x = "foo";
  y = "bar";
in x + y
```

Inheriting Attributes

Without `inherit`

```
let x = 123; in
{ x = x;
}
```

With `inherit`

```
let x = 123; in
{ inherit x;
}
```

Functions

General Form

pattern: body

Set Pattern

```
{x, y}: x + y
```

Default Values

```
{x, y ? 0}: x + y
```

Additional Arguments

```
{x, y, ...}: x + y
```

`with`-expressions

Brings all attributes of a set in scope

Example

```
with x = 1; y = 2; x
```

CLI Interface

CLI Overview

nix-instantiate	Create a derivation from a nix expression
nix-store	Manipulate and query the nix store
nix-build	Create a derivation from a nix expression and build it
nix-shell	Setup shell environment for building a derivation

nix-instantiate

- Evaluates nix expression and prints path of derivation
- Defaults to `default.nix`
- Select attribute with `-A`
- Pass arbitrary expression with `-E`

nix-store

- Build a derivation using `--realise`
- Garbage collect the nix store using `--gc`
- Show immediate dependencies using `--query --references`
- Show transitive dependencies using `--query --requisites`
- Show referrers of store path using `--query --referrers`

nix-build

- Combines `nix-instantiate` and `nix-store --realise`
- Symlinks store path to `result`

nix-shell

- Opens shell that has dependencies of derivation in scope
- Clear environment using `--pure`
- Specify packages that should be in scope using `-p`

Miso

App type

```
data App model action = App
  { model :: model
  , update :: action -> model
           -> Effect action model
  , view :: model -> View action
  , subs :: [ Sub action ]
  , events :: M.Map MisoString Bool
  , initialAction :: action
  , mountPoint :: Maybe MisoString
  }
```

Effect

```
data Effect action model =
  Effect model [(action -> IO ()) -> IO ()]

noEff :: model -> Effect action model
noEff m = Effect m []

(<#) :: model -> IO action -> Effect action model
m <# act = Effect m [\sink -> sink =<< a]
```

View

```
div_
  [ class_ "foobar" ]
  [ button_
      [ onClick ButtonClicked ]
      [ "Click me!" ]
  ]
```

MisoString

- Type synonym for
  - `JSString` when compiled with `GHCJS`
  - `Text` when compiled with `GHC`
- Conversion using `ToMisoString` typeclass

```
class ToMisoString str where
  toMisoString :: str -> MisoString
  fromMisoString :: MisoString -> str
```