PureScript-Resources Documentation

Justin Woo

Jul 06, 2019

Contents

1 Pages **3**

1.1 Introduction . . . 3
1.2 Installation . . . 4
1.3 Community . . . 5
1.4 Psc-Package or Spago . . . 5
1.5 Spacchetti . . . 6
1.6 FFI . . . 6
1.7 JSON . . . 6
1.8 HTTP Requests . . . 7
1.9 UI libraries . . . 8
1.10 Testing . . . 8
1.11 Elm-likes . . . 9
1.12 Collections . . . 9
1.13 Node Backends . . . 10
1.14 Databases . . . 10
1.15 Travis CI . . . 11
1.16 Azure Pipelines CI . . . 12
1.17 Purp, the non-Pulp . . . 13
1.18 Type-Level Programming . . . 13
1.19 Datatype Generics via Generics-Rep . . . 14
1.20 Blogs . . . 14
1.21 0.11.7 to 0.12.0 . . . 14
1.22 “Why did PureScript go from Eff to Effect?” . . . 15
1.23 Usage with Nix . . . 16
1.24 Etc . . . 17

**DEPRECATED: this content will be moved to GitHub for ease of maintenance: *https://github.com/justinwoo/purescript-resources*

This is a collection of notes and links to resources to learn and use PureScript as generally recommended by me. This is all biased information. Think of it as "awesome-how-to-purescript" that isn't just a link farm.

This guide contains a lot of . If you'd like to have something changed or explained more, make a PR or an issue.

There is another resources page out there that you might look at here: https://github.com/JordanMartinez/purescript-jordans-reference

---

Note: Do you find this guide useful? Please contribute your own writing to this page if you do!

CHAPTER 1

Pages

1.1 Introduction

1.1.1 "Why should I use PureScript?"

I will guess that you want to learn something where

- You can actually refactor programs without them breaking
- You want the computer to do more things that you shouldn't have to do
- You want to use tools that help you learn more and do more meaningful work in less time
- You want to do things as you see fit, not just as some thoughtleader has dictated

In that case, welcome. If not, maybe you'll still want to just poke around and look at some of the things here.

1.1.2 "How do I manage my dependencies?" / "Ew, do I have to use BOWER?"

You can just use Spago if you want some solution based on package sets: https://github.com/spacchetti/spago

I use Psc-Package for everything at work and at home for project dependency management. See the docs for Psc-Package https://psc-package.readthedocs.io/en/latest/ and the new package-sets https://github.com/purescript/package-sets

For Bower criticisms, you should read and understand http://harry.garrood.me/blog/purescript-why-bower/. Then you can start using Psc-Package for projects like a normal person.

1.1.3 "Why can't I just use NPM"?

Read the link above to read why npm-style dependencies don't work for PureScript.

However, **yes, almost everyone uses npm for their JavaScript dependencies. It is essential to how PureScript interops with existing JavaScript libraries.**

1.1.4 "Where do I find some general documentation on PureScript?"

Documentation is here https://github.com/purescript/documentation

Some of these pages are outdated or stale, so you might ask people to consider updating them in various channels.

1.1.5 "Where do I find libraries?"

Pursuit https://pursuit.purescript.org/

1.1.6 "Where do I find docs for libraries?"

Pursuit https://pursuit.purescript.org/

1.1.7 "What is the `purescript-contrib` org on Github?"

Various donated libraries to be maintained by contributors of `purescript-contrib`, to varying degrees. Use these if you want, don't use them if you don't want.

1.1.8 What is the difference between row type of Type (# Type) and Records ({}/Record ())?

Read here: https://pursuit.purescript.org/builtins/docs/Prim#t:Record

Note the parens vs curly brace

1.2 Installation

I wrote a blog post about this for people who want to install PureScript tooling via npm: https://qiita.com/kimagure/items/570e6f2bbce5b4724564

1.2.1 About npm in general

Prefix

Make sure you have prefix set for npm in `~/.npmrc`:

```
prefix="~/.npm"
```

If you don't do this, npm installations overall in your system will be messed up.

never run npm with `sudo`.

Set your npm paths

```
export PATH="$HOME/.npm/bin:$PATH"
export PATH="./node_modules/.bin:$PATH"
```

1.2.2 If you want to set up PureScript tooling via npm

```
npm i -g purescript pulp psc-package-bin-simple
```

1.2.3 If you don't want to set up tooling via npm

Grab the PureScript binary from Github releases and put it in your path (e.g. `~/.local/bin/`): https://github.com/purescript/purescript/releases

Grab the Psc-Package binary from Github releases and put it in your path: https://github.com/purescript/psc-package/releases

You will have to install `pulp` via npm, but you don't necessarily have to use pulp. Nevertheless, `npm install -g pulp` or `npm i -S pulp` in your project.

You might try this bash script, but if it doesn't work, make a PR: https://github.com/justinwoo/purescript-resources/blob/master/purs-install.bash

1.2.4 Installation of tools through Nix

Unfortunately, the Nix package for PureScript is usually broken, and there is no visible interest in making the package use the binaries from GitHub. As a result, installing the compiler through Nix will usually be broken or irreproducable.

I have started collecting easy ways of installing PureScript-related tools with Nix via the released binaries: https://github.com/justinwoo/easy-purescript-nix. Please try this! If you don't use NixOS, you can quite readily use this like so in https://github.com/justinwoo/dotfiles/commit/3b839ec52cab87df24455987b47fd942b61b4f43.

You can also try this simple derivation for the compiler: https://github.com/srdqty/purescript-project-template/blob/902f3e7c5ec4284a0878cb4806553e3756552231/nix/pkgs/purescript/default.nix

On the other hand, the Psc-Package package on nixpkgs is fine: https://github.com/NixOS/nixpkgs/blob/a6fa300cf7192b61234436dd199f3678b648a096/pkgs/development/compilers/purescript/psc-package/default.nix

1.3 Community

How to actually talk to people in PureScript:

- FP Slack: #purescript/#purescript-beginners https://fpchat-invite.herokuapp.com/
- Discourse: https://discourse.purescript.org/
- Reddit: https://www.reddit.com/r/purescript/

1.4 Psc-Package or Spago

I use Psc-Package, but I realize it might not be what people readily want. Maybe Spago will help: https://github.com/spacchetti/spago

See the docs for Psc-Package here https://psc-package.readthedocs.io/en/latest/

For easy installation, you might try https://www.npmjs.com/package/psc-package-bin-simple.

1.4.1 Spacchetti

I have a Dhall-based package set for Psc-Package that I actively use and base my work and home projects on. See the docs https://spacchetti.readthedocs.io/en/latest/

Repo https://github.com/justinwoo/spacchetti

1.4.2 With Nix

I have put together a Psc-Package2Nix project here: https://github.com/justinwoo/psc-package2nix

1.5 Spacchetti

Spacchetti is now merged in package-sets, so you should see it first: https://github.com/purescript/package-sets

Otherwise, also see the Spacchetti guide here: https://spacchetti.readthedocs.io/en/latest/

1.6 FFI

I wrote about this in a blog post called "User empowerment of FFI in PureScript"

You should read these links at minimum:

- PureScript language FFI documentation https://github.com/purescript/documentation/blob/master/language/FFI.md
- PureScript documentation guide on FFI https://github.com/purescript/documentation/blob/master/guides/FFI.md
- PureScript-Effect uncurried function documentation https://pursuit.purescript.org/packages/purescript-effect/2.0.0/docs/Effect.Uncurried

I have some examples of various ways of doing FFI here: https://github.com/justinwoo/purescript-ffi-intro

1.6.1 "What if I need to validate inputs from FFI?"

Use the `Foreign` type from the `purescript-foreign` library and see the next section, which is not only about JSON but also about foreign JS values.

1.7 JSON

Currently, I only use my own library Simple-JSON.

1.7.1 Simple-JSON

Everything you've dreamed of when defining transport types or wanting to work with inferred serialization types for JSON. See its documentation at http://purescript-simple-json.readthedocs.io/en/latest/

Repo at https://github.com/justinwoo/purescript-simple-json

1.7.2 Foreign

If you work with JS values, you will want to use the Foreign library. https://github.com/purescript/purescript-foreign

1.7.3 Foreign-Generic

A library for working with data types deriving `Generic`. https://github.com/paf31/purescript-foreign-generic

1.7.4 Argonaut-Core

A library for working with Json values but in a different way. If you want automatic decoding of JSON, this is not the set of libraries you want to use. Otherwise, welcome.

Repo at https://github.com/purescript-contrib/purescript-argonaut-core

Also see https://github.com/purescript-contrib/purescript-argonaut-codecs

1.7.5 "None of these libraries do what I want"

Writing your own solution is fairly simple, since you can start by only handling some of the cases you care about most. If you understand how to use single and multiparameter type classes, then you're ready to get started with making your own solution.

First, make sure you understand whether or not you want to use plain `Either`, `Except`, or something else.

Start by reading through this tutorial:

- http://purescript-simple-json.readthedocs.io/en/latest/generics-rep.html

This article alone should give you some ideas on how you might start to implement your own library.

You might want to also read these blog posts:

- https://github.com/justinwoo/my-blog-posts#automatically-deencoding-json-in-purescript-using-generics-rep
- https://github.com/justinwoo/my-blog-posts#writing-a-json-decoder-using-purescripts-rowtolist

1.8 HTTP Requests

I only use libraries that use Aff for requests. Here are the two libraries I actually use:

1.8.1 Milkis

I made this library so I could use Fetch on both the Browser (through window) and Node (through node-fetch). See its docs at https://purescript-milkis.readthedocs.io/en/latest/

See repo at https://github.com/justinwoo/purescript-milkis

1.8.2 Affjax

This is a library for working with XHR by Slamdata. It works, so you can use it. It currently has concepts of request and response data types and does come with some pains of dealing with Argonaut's Json types, but if you don't really care about Argonaut, you can set it to use the String responses and parse the JSON yourself instead. Read its and docs and see it at https://github.com/slamdata/purescript-affjax

1.9 UI libraries

There are effectively two actively maintained and working solutions.

1.9.1 React-Basic

Quite transparent React interop. Start by looking here: https://github.com/f-f/purescript-react-basic-todomvc

See the starter here: https://github.com/lumihq/react-basic-starter

See my fork with all of my opinionated things here: https://github.com/justinwoo/spacchetti-react-basic-starter

And the TodoMVC implementation here: https://github.com/f-f/purescript-react-basic-todomvc

React-Basic

1.9.2 Halogen

Pure PureScript. Fast if you don't write naive code with thousands of unkeyed children. Note that the docs are slow to update, but each major version is very usable. This library can end up being a deep rabbit hole of FP ideas and how FP also models OOP. Consider React-Basic if you want to get started quickly and improve your codebase incrementally.

See the starter projects here: https://github.com/slamdata/purescript-halogen-template, https://github.com/citizennet/purescript-halogen-template

See an example real-world application with authentication, routing, state management, and more: https://github.com/thomashoneyman/purescript-halogen-realworld

Halogen

1.9.3 No library

It might be worthwhile to not actually try to write your entire view in PureScript first, and rather interface in your existing application. You should read through the FFI links.

1.10 Testing

1.10.1 Assert

Simple but flexible

https://github.com/purescript/purescript-assert

1.10.2 Spec

Pretty fully featured

https://github.com/owickstrom/purescript-spec

1.10.3 Test-Unit

Works, but no fancy features

https://github.com/bodil/purescript-test-unit

1.11 Elm-likes

1.11.1 "I mostly need query-updater-html functions"

Use Halogen with one component, then add a few more whenever you find something that makes sense as a component. Or try to jerry-rig some web components setup.

1.11.2 Hedwig

Another library that presents you a fairly Elm-like experience https://github.com/utkarshkukreti/purescript-hedwig

1.11.3 Spork

For an up to date Elm-like that is performant, you should look at https://github.com/natefaubion/purescript-spork

1.11.4 Pux

Even though Pux is well-known, I do not recommend it because of various usability issues, overwhelming performance issues, and lack of maintenance. It also does not actually provide React interop, so if you want React interop, you should use https://github.com/lumihq/purescript-react-basic

If you're new to PureScript and frustrated with Pux, that's completely understandable as it is quite frustrating to work with. Try some of these other options or try using more FFI to do things you want.

1.12 Collections

1.12.1 Arrays, Lists

Use the libraries for extra functions on arrays and the List structure. https://github.com/purescript/purescript-arrays https://github.com/purescript/purescript-lists

1.12.2 Foreign-Object

For working with JS Objects as String-keyed Maps, see this library. https://github.com/purescript/purescript-foreign-object

1.12.3 Ordered Collections

For working with collections in general. https://github.com/purescript/purescript-ordered-collections

1.13 Node Backends

I do not use any libraries to do Node backend development for work, as I have not found any to be useful enough compared to the cost of having the wrong combination of requirements or missing functionality. See the section on FFI to get familiar with how to do things.

1.13.1 Makkori

I have a library for relatively simple Express usage, which can be extended with normal Express middleware as needed. My vidtracker project uses it to prepare the backend, though there are many other techniques involved here https://github.com/justinwoo/vidtracker/blob/master/src/Main.purs

See the repo at https://github.com/justinwoo/purescript-makkori

1.13.2 What about my databases???

Looking on Pursuit alone, you'll find https://github.com/epost/purescript-node-postgres is actively maintained. I use node-sqlite3 for many of my projects and keep it fairly maintained, though the binding itself is completely minimal: https://github.com/justinwoo/purescript-node-sqlite3

Overall, you should learn how to use FFI so you can make wrappers around things you need and improve their types overall.

1.13.3 HTTPure

You might look at this project if you want to try something active written in plain PureScript https://github.com/cprussin/purescript-httpure

1.14 Databases

If you're really going to write a backend in PureScript (like on AWS Lambda and other such offerings), you probably need to talk to a database.

1.14.1 General

Generally, you should learn how to use FFI and bind to the database you actually want to use, then start thinking about how libraries like Simple-JSON can help you decode results from databases.

Once you learn how to use FFI, you will not need anyone else to provide you a library. This is a large part of the reason why many production users of PureScript on Node do not contribute libraries back to the ecosystem, as they make small interfaces to libraries that they build on top of that contain many assumptions.

Warning

If you see a library that contains a `ReadForeign`, `IsForeign`, or `Decode` constraint, you should probably not use that library, as this is a clear sign of someone imposing a specific combination of libraries on you.

1.14.2 SQLite3

I wrote this library and it provides a wrapper for node-sqlite3. I use this personally, and have been using it for over two years.

https://pursuit.purescript.org/packages/purescript-node-sqlite3

1.14.3 Postgres

I use this sometimes, like at work. You can use this if you want, or just make your own wrapper for pg.

https://pursuit.purescript.org/packages/purescript-node-postgres

1.15 Travis CI

Travis, being the problem child that it is, over represents resources available to it by default. One normal workaround is to force usage of a different environment by using `sudo: required` in .travis.yml like so:

```
dist: trusty
sudo: required
```

(from https://github.com/purescript/package-sets/blob/6f9f0b0eaea5e3718c860bc0cbaa651a554aad21/.travis.yml)

1.15.1 Example configuration

```
language: c
dist: trusty
sudo: required

cache:
  directories:
  - .psc-package
  - output

env:
  - PATH=$HOME/purescript:$HOME/psc-package:$PATH

install:
  - TAG=v0.12.0
  - PSC_PACKAGE_TAG=v0.3.2
  - wget -O $HOME/purescript.tar.gz https://github.com/purescript/purescript/releases/
↪download/$TAG/linux64.tar.gz
  - tar -xvf $HOME/purescript.tar.gz -C $HOME/
  - chmod a+x $HOME/purescript
  - wget -O $HOME/psc-package.tar.gz https://github.com/purescript/psc-package/
↪releases/download/$PSC_PACKAGE_TAG/linux64.tar.gz
  - tar -xvf $HOME/psc-package.tar.gz -C $HOME/
```

(continues on next page)

(continued from previous page)

```
  - chmod a+x $HOME/psc-package

script:
  - make setup-only
  - psc-package verify
```

From https://github.com/justinwoo/spacchetti/blob/f6779d19cc0e9bf3cd041966dd14b480f48dbc57/.travis.yml

1.15.2 Telling Haskell RTS the bad news

You can pass runtime system arguments as pass-through arguments to pulp to make Travis build correctly:

```
pulp build -- +RTS -N1 -RTS
```

This will make builds run smoothly most of the time. As with everything Travis-related, godspeed.

1.16 Azure Pipelines CI

Azure Pipelines works surprisingly well, with no hacks needed to get things going.

1.16.1 Example config

```
pool:
  vmImage: 'Ubuntu 16.04'

steps:
- script: |
    PURESCRIPT_TAG=v0.12.0
    PSC_PACKAGE_TAG=v0.3.2

    PURESCRIPT=https://github.com/purescript/purescript/releases/download/$PURESCRIPT_
↪TAG/linux64.tar.gz
    PSC_PACKAGE=https://github.com/purescript/psc-package/releases/download/$PSC_
↪PACKAGE_TAG/linux64.tar.gz

    wget -O $HOME/purescript.tar.gz $PURESCRIPT
    wget -O $HOME/psc-package.tar.gz $PSC_PACKAGE

    tar -xvf $HOME/psc-package.tar.gz -C $HOME/
    tar -xvf $HOME/purescript.tar.gz -C $HOME/

    mv $HOME/purescript/* $HOME/bin
    mv $HOME/psc-package/* $HOME/bin

    chmod a+x $HOME/bin
  displayName: 'Install deps'
- script: |
    export PATH=./bin:$HOME/bin:$PATH

    which purs
    which psc-package
```

(continues on next page)

(continued from previous page)

```
    make
  displayName: 'Make'
```

If you don't mind using npm:

```
pool:
  vmImage: 'Ubuntu 16.04'

steps:
- script: |
    export PATH=~/.npm/bin:$PATH
    npm set prefix ~/.npm
    npm i -g purescript psc-package-bin-simple
    make setup-only
    psc-package verify
  displayName: 'Install deps and run'
```

From https://github.com/justinwoo/vidtracker/blob/520fb4288de13114394f40f2b191553714c6bd5d/azure-pipelines.yml and https://github.com/justinwoo/spacchetti/blob/f6779d19cc0e9bf3cd041966dd14b480f48dbc57/azure-pipelines.yml

1.17 Purp, the non-Pulp

To work with Psc-Package projects, I use purp: https://github.com/justinwoo/purp

You may be interested in using this also, but otherwise, you can always use Pulp if you don't mind using a node CLI.

1.18 Type-Level Programming

In PureScript, type-level programming isn't about being "smart" or "talented", it's only about solving problems. If you're seeking validation or for something to "prove" your "intelligence", you may be better off reading books or arguing politics on Twitter instead.

This is a truly exciting and interesting area of PureScript, but most people wanting to look at this don't understand enough fundamentals to actually solve their problems. Instead of trying to link you to my blog posts, it's more useful for me to write down a list of topics you should read about and know before you try to start doing this:

- Pattern matching (e.g. of Data.List Cons, Nil)
- Type classes, single parameters and their instances
- Proxy, SProxy, etc.
- Multiple parameter type classes
- Functional dependencies
- Overlapping instances
- Row polymorphism in PureScript
- PureScript-Record
- PureScript-Variant
- Datatype Generics/PureScript-Generics-Rep

- PureScript-Typelevel-Prelude

Optional:

- Instance chains in PureScript

The associated literature for some of these topics in the GHC User's Guide make for a good first start to read about, and long-form explanations can also be found on PureScript By Example.

For understanding, you should devote some time to making some examples of each topic and take some notes that you can refer to later, such as blog posts, an ORG file, or written notes.

To tie everything together, you might read my post Type classes and instances are pattern matching for types.

1.19 Datatype Generics via Generics-Rep

If you've ever wanted to work with information about your data types in terms of generic types and representations that can be converted to and from, this is exactly what you've been looking for: https://github.com/purescript/purescript-generics-rep

Here is a tutorial on Generics-Rep in the Simple-JSON docs: https://purescript-simple-json.readthedocs.io/en/latest/generics-rep.html

For much more detailed information, refer to the GHC User guide https://downloads.haskell.org/~ghc/latest/docs/html/users_guide/glasg programming and the Haskell generic-deriving library docs http://hackage.haskell.org/package/generic-deriving-1.12.1/docs/Generics-Deriving-Base.html

1.20 Blogs

I write the most out of anyone, and my posts are in a repo here: https://github.com/justinwoo/my-blog-posts

Also see the subreddit for other posts: https://www.reddit.com/r/purescript/

1.21 0.11.7 to 0.12.0

There are very few changes needed to be made to upgrade to 0.12.0.

Below from https://gist.github.com/justinwoo/c28327abe623c117e938a0b471b2e2a2

There are some changes you will need to make for most applications to be upgraded to PureScript 0.12. With some usage of editor commands, you should be able to convert any 20K LOC codebase in less than an hour.

1.21.1 Libraries

- Remove `eff`, install `effect`
- Remove `dom` and `dom-*`, use `web-dom` and such from purescript-web. Use type holes (`?whatmethod`) to discover new APIs
- Remove `maps`, install `ordered-collections` for Map/Set/etc. and `foreign-object` for `StrMap`

1.21.2 Changes

- `Eff (fx :: # Type) a` -> `Effect a`
- `Aff (fx :: # Type) a` -> `Aff a`
- `Control.Monad.Effect` -> `Effect`
- `id` -> `identity`
- `Data.Record` -> `Record`
- `Data.StrMap` -> `Foreign.Object`
- `Data.Foreign` -> `Foreign`

1.21.3 Updating libraries

- Use a newer package set or use `ncu -uam bower` via npm-check-updates
- If you want a package set that is actively maintained by me, see https://github.com/purescript/package-sets/releases

1.21.4 General

- Use `psc-package build -d` or `pulp build --src-path some-empty-folder` if you want to only build dependencies first (you should)

1.22 "Why did PureScript go from Eff to Effect?"

There are quite many ill-informed "takes" on PureScript's switch from Eff with row types to Effect. This page serves to provide a **minimum** reading list for one to become familiar with the subject. If you just hate PureScript, there are alternatives you can use without having to learn about different ways to represent effects.

There are various resources you can check about this:

- https://github.com/purescript/purescript/issues/3080
- https://github.com/purescript-deprecated/purescript-eff/issues/25
- https://github.com/slamdata/purescript-io/blob/master/README.md

Eff rows were not sufficiently useful in terms of actually guaranteeing what effects were run in a given `Eff row`, and the attempts to track possible exceptions were poor. In addition, these led to many problems with users not knowing how to solve type errors with unification of effect rows.

If you want to actually track effects, find some reading about MTL, Free, Tagless Final, and also read through https://github.com/natefaubion/purescript-run/blob/master/README.md for an implementation of extensible, algebraic effects for PureScript.

If you want an example of an approach where you can work with known errors, read through https://github.com/natefaubion/purescript-checked-exceptions/blob/master/README.md

1.22.1 "There was a huge rewrite to go from Eff to Effect"

There was little involved other than for renaming.

https://twitter.com/jusrin00/status/1021736059040919552

1.22.2 "It's hard to go from Eff to Effect"

Unless you already used many tricks to encode extra information by creating types of kind `Type -> Effect`, there are only renaming changes involved.

https://twitter.com/jusrin00/status/1021737244674154496

1.22.3 "Eff to Effect was decided by Twitter poll"

See links above, and then see the actual poll and thread of explanations: https://twitter.com/paf31/status/908760073303764993

"But the result of the poll isn't vastly positive"

How much did you have faith in Twitter polls anyway?

1.23 Usage with Nix

You might look through this post to see the components involved: https://qiita.com/kimagure/items/aec640d0047d08d2ce90

I've been trying to put a few things together:

1.23.1 easy-purescript-nix

Easily use PureScript with Nix. Doesn't require hard to reproduce builds, but actually prefers binaries that can be downloaded and used readily.

https://github.com/justinwoo/easy-purescript-nix

You can see an example of this in action in the vidtracker repo: https://github.com/justinwoo/vidtracker/blob/f78b3df57eaf5b122f0a0b51cc4e3c246bf96f88/default.nix

1.23.2 psc-package2nix

Generates a series of derivations from a solved dependency set from Psc-Package.

https://github.com/justinwoo/psc-package2nix

You can see this in action being used in the vidtracker repo: https://github.com/justinwoo/vidtracker/blob/f78b3df57eaf5b122f0a0b51cc4e3c246bf96f88/install-deps.nix

1.24 Etc

1.24.1 Where are union types from TypeScript?

First, you should read some useful information from Typed Racket to learn how union types and occurence typing work:

- https://docs.racket-lang.org/ts-guide/types.html?q=T:ts-guide#%28part._.Union_.Types%29
- https://docs.racket-lang.org/ts-guide/occurrence-typing.html?q=T:ts-guide

Once you have read through these, you will want to use

- Regular sum types, where occurence typing is replaced by pattern matching
- Polymorphic Variants via https://github.com/natefaubion/purescript-variant
- Some other approximation by first using Foreign types to safely read to types you want to use

If you fully understand the above and you're looking for a challenge, try taking a look at the implementation the associated blog post here: https://github.com/justinwoo/purescript-Hotteok