Spacchetti Documentation

Justin Woo

Oct 19, 2018

Contents

This is a guide for the Package Set project Spacchetti, which provides a way to work with package definitions for Psc-Package using the Dhall programming language. This guide will also try to guide you through some of the details of how Psc-Package itself works, and some details about the setup of this project and how to use Dhall.

Note: If there is a topic you would like more help with that is not in this guide, open a issue in the Github repo for it to request it.

CHAPTER 1

Pages

1.1 Introduction to Psc-Package

1.1.1 What is Psc-Package?

Psc-Package is a package manager for PureScript that works essentially by running a bunch of git commands. Its distinguishing feature from most package managers is that it uses a **package set**.

1.1.2 What is a Package Set?

Many users trying to rush into using Psc-Package don't slow down enough to learn what package sets are. They are a **set** of packages, such that the package set only contains **one** entry for a given package in a set. This means that

- Whichever package you want to install must be in the package set
- The dependencies and the transitive dependencies of the package you want to install must be in the package set

Package sets are defined in `packages.json` in the root of any package set repository, like in https://github.com/justinwoo/spacchetti/blob/master/packages.json.

1.1.3 How are package sets used?

Package sets are consumed by having a `psc-package.json` file in the root of your project, where the contents are like below:

```
{
  "name": "project-name",
  "set": "set-name",
  "source": "https://github.com/justinwoo/spacchetti.git",
  "depends": [
    "aff",
    "console",
```

(continues on next page)

(continued from previous page)

```
    "prelude"
  ]
}
```

So the way this file works is that

- `"set"` matches the tag or branch of the git repository of the package set
- `"source"` is the URL for the git repository of the package set
- `"depends"` is an array of strings, where the strings are names of packages you depend on

When you run `psc-package install`, psc-package will perform the steps so that the following directory has the package set cloned to it:

```
.psc-package/set-name/.set
```

And the package set will be available in

```
.psc-package/set-name/.set/packages.json
```

When you install a package in your given package set, the directory structure will be used, such that if you install `aff` from your package set at version `v5.0.0`, you will have the contents of that package in the directory

```
.psc-package/set-name/aff/v5.0.0
```

Once you understand these three sections, you'll be able to solve any problems you run into with Psc-Package.

1.2 Why/How Dhall?

Dhall is a programming language that guarantees termination. Its most useful characteristics for uses in this project are

- Static typing with correct inference: unlike the packages.json file, we have the compiler check that we correctly define packages
- Functions: we can use functions to create simple functions for defining packages
- Local and remote path importing: we can use this to mix and match local and remote sources as necessary to build package sets
- Typed records with directed merging: we can use this to split definitions into multiple groupings and apply patching of existing packages as needed

Let's look at the individual parts for how this helps us make a package set.

1.2.1 Files

The files in this package set are prepared as such:

```
-- Package type definition
src/Package.dhall

-- function to define packages
src/mkPackage.dhall
```

(continues on next page)

(continued from previous page)

```
-- packages to be included when building package set
src/packages.dhall

-- package "groups" where packages are defined in records
src/groups/[...].dhall
```

Package.dhall

This contains the simple type that is the definition of a package:

```
{ dependencies : List Text, repo : Text, version : Text }
```

So a given package has a list of dependencies, the git url for the repository, and the tag or branch that it can be pulled from.

mkPackage.dhall

This contains a function for creating `Package` values easily

```
  λ(dependencies : List Text)
→ λ(repo : Text)
→ λ(version : Text)
→   { dependencies = dependencies, repo = repo, version = version }
  : ./Package.dhall
```

While this function is unfortunately stringly typed, this still lets us conveniently define packages without having to clutter the file with record definitions.

packages.dhall

This is the main file used to generate `packages.json`, and is defined by taking package definitions from the groups and joining them with a right-sided merge.

```
  ./groups/purescript.dhall
 ./groups/purescript-contrib.dhall
 ./groups/purescript-web.dhall
 ./groups/purescript-node.dhall
-- ...
 ./groups/justinwoo.dhall
 ./groups/patches.dhall
```

1.2.2 Definitions and overrides

As `patches.dhall` is last, its definitions override any existing definitions. For example, you can put an override for an existing definition of `string-parsers` with such a definition:

```
    let mkPackage = ./../mkPackage.dhall

in  { string-parsers =
        mkPackage
        [ "arrays"
```

(continues on next page)

(continued from previous page)

```
        , "bifunctors"
        , "control"
        , "either"
        , "foldable-traversable"
        , "lists"
        , "maybe"
        , "prelude"
        , "strings"
        , "tailrec"
        ]
        "https://github.com/justinwoo/purescript-string-parsers.git"
        "no-code-points"
    }
```

1.2.3 Video

I recorded a demo video of how adding a package to Spacchetti works: https://www.youtube.com/watch?v=4Rh-BY-7sMI

1.3 How to use this package set

1.3.1 Requirements

This project requires that you have at least:

- Linux/OSX. I do not support Windows. You will probably be able to do everything using WSL, but I will not support any issues (you will probably barely run into any with WSL). I also assume your distro is from the last decade, any distributions older than 2008 are not supported.
- Dhall-Haskell and Dhall-JSON installed. You can probably install them from Nix or from source.
- Psc-Package installed, with the release binary in your PATH in some way.
- jq installed.

1.3.2 How to generate the package set after editing Dhall files

First, test that you can actually run `make`:

```
> make
./format.sh
formatted dhall files
./generate.sh
generated to packages.json
./validate.pl
validated packages' dependencies
```

This is how you format Dhall files in the project, generate the `packages.json` that needs to be checked in, and validate that all dependencies declared in package definitions are at least valid. Unless you plan to consume only the `packages.dhall` file in your repository, you must check in `packages.json`.

To actually use your new package set, you must create a new git tag and push it to your **fork of spacchetti**. Then set your package set in your **project** repository accordingly, per EXAMPLE:

```
{
  "name": "EXAMPLE",
  "set": "160618", // GIT TAG NAME
  "source": "https://github.com/justinwoo/spacchetti.git", // PACKAGE SET REPO URL
  "depends": [
    "console",
    "prelude"
  ]
}
```

When you set this up correctly, you will see that running `psc-package install` will create the file `.psc-package/{GIT TAG NAME HERE}/.set/packages.json`.

1.3.3 Testing changes to package set

To set up a test project, run `make setup`. Then you can test individual packages with `psc-package verify PACKAGE`.

1.3.4 Using Perl scripts in this repository

You will only need the following scripts:

- `verify.pl` - to install a given package and validate the entire compiled output.
- `from-bower.pl` - to add/update a package that is registered on Bower.

These each take an argument of a package, e.g. `./update-from-bower.pl behaviors`.

1.4 Project-Local setup

There's now a CLI for the repetitive boilerplate generation and task running parts here: https://github.com/justinwoo/spacchetti-cli

In psc-package, there is nothing like "extra-deps" from Stack. Even though editing a package set isn't hard, it can be fairly meaningless to have a package set that differs from package sets that you use for your other projects. While there's no real convenient way to work with it with standard purescript/package-sets, this is made easy with Dhall again where you can define a packages.dhall file in your repo and refer to remote sources for mkPackage and some existing packages.dhall.

1.4.1 With the CLI

With the Spacchetti CLI, you can automate the manual setup below and run a single command to update your package set.

To use the CLI, you will first need fulfill the requirements.

Then, install the Spacchetti CLI in a manner you prefer:

- npm: you can use `npm install --global spacchetti-cli-bin-simple` to install via npm.
- Github releases: You can go to the release page on Github, download the archive with your platform's binary, and put it somewhere on your PATH https://github.com/justinwoo/spacchetti-cli/releases

- stack install: You can clone the repository and run `stack install`: https://github.com/justinwoo/spacchetti-cli

When you have installed the CLI, you can run `spacchetti` to be shown the help message:

```
Spacchetti CLI

Usage: spacchetti (local-setup | insdhall)

Available options:
  -h,--help                Show this help text

Available commands:
  local-setup              run project-local Spacchetti setup
  insdhall                 insdhall the local package set
```

Local setup

First, run the local-setup command to get the setup generated.

```
spacchetti local-setup
```

This will generate two files:

- `packages.dhall`: this is your local package set file, which will refer to the upstream package set and also assign a `upstream` variable you can use to modify your package set.
- `psc-package.json`: this is the normal psc-package file, with the change that it will refer to a "local" set.

Before you try to run anything else, make sure you run `spacchetti insdhall`:

InsDhall

Now you can run the ins-dhall-ation of your package set:

```
spacchetti insdhall
```

This will generate the package set JSON file from your package set and place it in the correct path that psc-package will be able to use. You can now use `psc-package install` and other normal psc-package commands.

Updating the local package set

For example, you may decide to use some different versions of packages defined in the package set. This can be achieved easily with record merge updates in Dhall:

```
    let mkPackage =
          https://raw.githubusercontent.com/justinwoo/spacchetti/140918/src/mkPackage.
↪dhall

in  let upstream =
          https://raw.githubusercontent.com/justinwoo/spacchetti/140918/src/packages.
↪dhall

in  let overrides =
          { halogen =
```

(continues on next page)

(continued from previous page)

```
              upstream.halogen  { version = "master" }
          , halogen-vdom =
              upstream.halogen-vdom  { version = "v4.0.0" }
          }

in  upstream  overrides
```

If you have already fetched these packages, you will need to remove the `.psc-package/` directory, but you can otherwise proceed.

Run the ins-dhall-ation one more time:

```
spacchetti insdhall
```

Now you can install the various dependencies you need by running `psc-package install` again, and you will have a locally patched package set you can work with without upstreaming your changes to a package set.

You might still refer to the manual setup notes below to see how this works and how you might remove the Spacchetti CLI from your project workflow should you choose to.

With CI

You can install everything you need on CI using some kind of setup like the following.

These examples come from vidtracker: https://github.com/justinwoo/vidtracker/tree/37c511ed82f209e0236147399e8a91999aaf754c

Azure

```
pool:
  vmImage: 'Ubuntu 16.04'

steps:
- script: |
    DHALL=https://github.com/dhall-lang/dhall-haskell/releases/download/1.17.0/dhall-
↪1.17.0-x86_64-linux.tar.bz2
    DHALL_JSON=https://github.com/dhall-lang/dhall-json/releases/download/1.2.3/dhall-
↪json-1.2.3-x86_64-linux.tar.bz2

    wget -O $HOME/dhall.tar.gz $DHALL
    wget -O $HOME/dhall-json.tar.gz $DHALL_JSON

    tar -xvf $HOME/dhall.tar.gz -C $HOME/
    tar -xvf $HOME/dhall-json.tar.gz -C $HOME/

    chmod a+x $HOME/bin

    npm set prefix ~/.npm
    npm i -g purescript psc-package-bin-simple spacchetti-cli-bin-simple

  displayName: 'Install deps'
- script: |
    export PATH=~/.npm/bin:./bin:$HOME/bin:$PATH
```

(continues on next page)

(continued from previous page)

```
    which spacchetti
    which dhall
    which dhall-to-json

    make
  displayName: 'Make'
```

Travis

```
language: node_js
sudo: required
dist: trusty
node_js: stable
env:
  - PATH=./bin:$HOME/bin:$PATH
install:
  - DHALL=https://github.com/dhall-lang/dhall-haskell/releases/download/1.17.0/dhall-
↪1.17.0-x86_64-linux.tar.bz2
  - DHALL_JSON=https://github.com/dhall-lang/dhall-json/releases/download/1.2.3/dhall-
↪json-1.2.3-x86_64-linux.tar.bz2
  - SPACCHETTI=https://github.com/justinwoo/spacchetti-cli/releases/download/0.2.0.0/
↪linux.tar.gz
  - wget -O $HOME/dhall.tar.gz $DHALL
  - wget -O $HOME/dhall-json.tar.gz $DHALL_JSON
  - wget -O $HOME/spacchetti.tar.gz $SPACCHETTI
  - tar -xvf $HOME/dhall.tar.gz -C $HOME/
  - tar -xvf $HOME/dhall-json.tar.gz -C $HOME/
  - tar -xvf $HOME/spacchetti.tar.gz -C $HOME/bin
  - chmod a+x $HOME/bin
  - npm install -g purescript pulp psc-package-bin-simple
script:
  - which dhall
  - which dhall-to-json
  - which spacchetti
  - make
```

1.4.2 Manual setup

See the moved notes here

1.5 Manual setup

1.5.1 `packages.dhall`

For example, we could patch `typelevel-prelude` locally in such a way in a project-local `packages.dhall` file:

```
    let mkPackage =
          https://raw.githubusercontent.com/justinwoo/spacchetti/190618/src/mkPackage.
↪dhall
```

(continues on next page)

(continued from previous page)

```
in  let overrides =
          { typelevel-prelude =
              mkPackage
              [ "proxy", "prelude", "type-equality" ]
              "https://github.com/justinwoo/purescript-typelevel-prelude.git"
              "prim-boolean"
          }

in    https://raw.githubusercontent.com/justinwoo/spacchetti/190618/src/packages.dhall
     overrides
```

1.5.2 psc-package.json

Then we need a `psc-package.json` file, but we will stub the package set information:

```
{
  "name": "my-project",
  "set": "local",
  "source": "",
  "depends": [
    "console",
    "effect",
    "prelude",
    "typelevel-prelude"
  ]
}
```

1.5.3 insdhall.sh

Finally, we will need to create the Psc-Package files and insert our local generated package set:

```
NAME='local'
TARGET=.psc-package/$NAME/.set/packages.json
mkdir -p .psc-package/$NAME/.set
dhall-to-json --pretty <<< './packages.dhall' > $TARGET
echo wrote packages.json to $TARGET
```

Once we run this script, we will now be able to use `psc-package install` and get to work.

1.6 FAQ

1.6.1 What is Spacchetti?

> This is a guide for the Package Set project Spacchetti, which provides a way to work with package definitions for Psc-Package using the Dhall programming language. This guide will also try to guide you through some of the details of how Psc-Package itself works, and some details about the setup of this project and how to use Dhall.

It's a package set for psc-package that uses a language that almost acts like SASS for JSON/YAML, but has types and much more.

1.6.2 Why should I use Spacchetti over normal Psc-Package?

First, make sure to read the short explanation of Psc-Package: https://spacchetti.readthedocs.io/en/latest/intro.html

Then read the explanation of why and how Dhall is used: https://spacchetti.readthedocs.io/en/latest/why-dhall.html

In short, because package sets are annoying to edit when they're only in JSON form, but using Dhall can make working with this information much easier.

1.6.3 Does Spacchetti CLI replace Psc-Package?

No, Spacchetti CLI only does some simple tasks that generate files and use Dhall to prepare Psc-Package package sets. There are no overlapping commands with Psc-Package.