Psc-Package Documentation

Justin Woo

Feb 08, 2020

Contents

This is a guide for the tool Psc-Package, which is a tool for managing PureScript dependencies via Git. It can be used directly or by external tools like Pulp.

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

1.1.1 What is a Package Set?

A *package set* is a mapping from package names to:

- the Git repository URL for the package
- the Git ref which should be passed to `git clone` to clone the appropriate version (usually a tag name, but a SHA is also valid)
- the package's transitive dependencies

A package set repository contains a `packages.json` file which contains all mapping information. `psc-package` uses this information to decide which repos need to be cloned.

The default package set is purescript/package-sets, but it is possible to create custom package sets in many ways:

- by preparing a package set with Dhall using the packages.dhall from releases: https://github.com/purescript/package-sets/releases
- forking `purescript/package-sets` to your own repository
- creating a new one from scratch

. One benefit of using the default package set is that it is verified by a continuous integration process.

1.1.2 The `psc-package.json` format

Here is a simple project configuration:

```
{
    "name": "my-project",
    "set": "psc-0.13.6",
```

(continues on next page)

(continued from previous page)

```
    "source": "https://github.com/purescript/package-sets.git",
    "depends": [
        "prelude"
    ]
}
```

It defines:

- The project name
- The package set to use to resolve dependencies. This corresponds to a branch or tag of the package set source repository if you use a Git URL as your source. Otherwise it serves as just an identifier.
- The package set source, which is either a repository Git URL or a path to a package set `packages.json` file.
- Any dependencies of the project, as a list of names of packages from the package set

1.2 Installation

1.2.1 Any platform

You can install Psc-Package on any platform by downloading the binary for your platform from the releases page and copying it somewhere on your PATH.

1.2.2 npm

You can install Psc-Package through the npm package: https://www.npmjs.com/package/psc-package

```
# globally
npm i -g psc-package

# for your project
npm i -S psc-package
```

This should work on Linux, OSX, and Windows. Please report issues at https://github.com/justinwoo/npm-psc-package-bin-simple if this does not work as expected.

1.2.3 Linux/OSX

Nix

You should be able to use the derivation provided in nixpkgs: https://github.com/NixOS/nixpkgs/blob/master/pkgs/development/compilers/purescript/psc-package/default.nix.

If you're not on NixOS, you might use `nix-env -i psc-package`.

1.2.4 Windows

If you're a **Windows Chocolatey** user, then you can install `psc-package` from the official repo:

```
$ choco install psc-package
```

1.2.5 Travis

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

cache:
  directories:
  - .psc-package
  - output

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

install:
  - TAG=v0.13.6
  - PSC_PACKAGE_TAG=v0.6.2
  - 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/
  - chmod a+x $HOME/psc-package

script:
  - ./travis.sh
```

See https://github.com/purescript/package-sets/blob/6f9f0b0eaea5e3718c860bc0cbaa651a554aad21/.travis.yml

1.3 Using Psc-Package from your project

1.3.1 Frequently used commands

```
# install or update the dependencies listed in psc-package.json
$ psc-package install

# install or update the package and add it to psc-package.json if not listed
$ psc-package install <package>

# list available commands
$ psc-package --help
```

1.3.2 Create a project

A new package can be created using `psc-package init`. This will:

- Create a simple `psc-package.json` file based on the current compiler version
- Add the Prelude as a dependency (this can be removed later)
- Sync the local package database (under the `.psc-package/` directory) by cloning any necessary repositories.

1.3.3 Add dependencies

To add a dependency, either:

- Use the `install <package name>` command, which will update the project configuration automatically, or
- Modify the `psc-package.json` file, and sync manually by running the `install` command (previously `update`).

1.3.4 Build a project

Active project dependencies and project source files under `src` can be compiled using the `build` command.

This command is provided as a convenience until external tools add support for `psc-package`. It *might* be removed in future.

1.3.5 Query the local package database

The local package database can be queried using the following commands:

- `sources` - list source directories for active package versions. This can be useful when building a command for, say, running PSCi.
- `dependencies` - list all transitive dependencies

1.3.6 Local package sets

In `psc-package.json`, you can set `"source"` to be a path to a local file:

```
{
  "name": "name",
  "set": "local",
  "source": "packages.json",
  "depends": [
    "aff"
  ]
}
```

From here, you can generate a local packages.json file in any way you please and use this package set directly. Consider if you use Dhall:

```
let upstream =
      https://github.com/purescript/package-sets/releases/download/psc-0.13.5-
↪20200103/packages.dhall␣
↪sha256:0a6051982fb4eedb72fbe5ca4282259719b7b9b525a4dda60367f98079132f30

in    upstream
    { calpis =
        { dependencies = [ "prelude" ]
        , repo = "https://github.com/justinwoo/purescript-calpis.git"
        , version = "v0.1.0"
        }
    }
```

This definition takes an existing release and adds the "calpis" package. Then we can generate a package set from this by running Dhall-JSON:

dhall-to-json –file packages.dhall –output packages.json

Then we can install this package expected:

```
$ psc-package install calpis
Installing calpis
psc-package.json file was updated

$ cat psc-package.json
{
  "name": "name",
  "set": "local",
  "source": "packages.json",
  "depends": [
    "aff",
    "calpis"
  ]
}
```

1.4 Usage with Nix

1.4.1 justinwoo/psc-package2nix (Older)

Justin Woo has made a psc-package2nix project for generating nix derivations from Psc-Package dependencies: https://github.com/justinwoo/psc-package2nix

1.4.2 justinwoo/soba (Newer)

Because of the relative rarity in using psc-package with Nix, there is not excessive documentation on how to use this tool. See mostly the `soba insdhall` and `soba nix` commands in this tool.

1.5 Working with package sets

Please see the README in package-sets: https://github.com/purescript/package-sets

1.6 FAQ

1.6.1 Is there an easier way to manage my package sets than to edit packages.json? / How do I prepare a packages.json in some way other than editing JSON?

Yes. purescript/package-sets itself uses Dhall to programmatically prepare the package set, which are then normalized into `packages.dhall` on release: https://github.com/purescript/package-sets/releases. You can import this file in your own Dhall files and add/override packages to customize your package set.

1.6.2 How come I can't install (some package) from the package set?

You should make sure you're using the correct package-set release and have updated the value of "set" in your `psc-package.json` file. See The `psc-package.json` format section for more details.

1.6.3 Why are my changes not updated in my package set?

Package sets are cached based on a git reference (e.g. tag or branch) to the project directory `.psc-package`. If you are making changes to a package set and reusing the package reference then you will need to clear the cache for the changes to take effect.

```
$ rm -rf .psc-package
$ psc-package install
```

1.6.4 Can I use Psc-Package with Nix?

Yes, please see the page about Nix here: https://psc-package.readthedocs.io/en/latest/nix.html.