Psc-Package Documentation

Justin Woo

Feb 02, 2019

Contents

1 Pages **3**
- 1.1 Introduction . . . 3
- 1.2 Installation . . . 4
- 1.3 Using Psc-Package from your project . . . 5
- 1.4 Usage with Nix . . . 6
- 1.5 Working with package sets . . . 6
- 1.6 FAQ . . . 7

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 by forking an existing package set or 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-0.10.2",
    "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)
- The package set source repository Git URL (change this if you want to host your own package sets)
- 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 might install http://npmjs.com/package/psc-package-bin-simple either globally or via project depencies. 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
```

(continues on next page)

(continued from previous page)

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

install:
  - TAG=v0.12.0
  - PSC_PACKAGE_TAG=v0.4.1
  - 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

1.4 Usage with Nix

1.4.1 justinwoo/psc-package2nix

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

1.5 Working with package sets

1.5.1 Add a package to the package set

Adding your package to the package set means that others can easily install it as a dependency.

Please note that your package will be removed from the set if it is not kept up to date. It can be easily re-added later if this happens.

Adding a package is a manual process right now. We would like to add commands to make this process simpler, but for now, please follow these steps:

- go to the package-sets repository and fork the repository.
- open the `packages.json` file and make a new entry to add your package, copying the format used for existing packages. The key will be the package name without the preceding `purescript-` as in Bower packages. It should have three fields defined:
- `dependencies` - a list of the dependencies used for this package
- `repo` - a git url for the package. We most often use the format `https://github.com/{user}/purescript-{project-name}.git`
- `version` - the git tag that will be used, using the format `v{Major}.{Minor}.{Patch}`
- when you have added your package, you will want to test this.

First, you will need to create an empty `psc-package.json` to test the package set in use.

```
echo '{ "name": "test", "set": "testing", "source": "", "depends": [] }' > psc-package.json
```

Make the required directory structure for the package sets:

```
mkdir -p .psc-package/testing/.set
```

Then copy over `packages.json` into the directory:

```
cp packages.json .psc-package/testing/.set/packages.json
```

- after this setup, you can use the `verify` command of psc-package, e.g. `psc-package verify aff`. This will verify the package and its reverse dependencies.

If this builds correctly, you can then push up this branch and make a pull request. Travis will verify your package builds correctly, and then we will try to merge your pull request. Your package will then be available in the next tagged package set.

1.5.2 Formatting the package set

When creating your pull request, make sure to run the `format` command to pretty-print the packages file. This helps us avoid problems in the future with git diffs and so on.

1.5.3 Update a package in the set

Similar to the above, you will need to do the setup. You can then modify the version field to the version you wish to use. Once you have updated the package set, run the copy command and verify the package you have modified.

Then you can make a pull request. Again, once Travis verifies your change, we will merge it into `master` and your change will be available in the next tag.

1.5.4 Alternative package sets

You may also be interested in using Spacchetti, a package set project using Dhall to define and work with package sets: https://github.com/justinwoo/spacchetti.

It also defines ways to define a local package set that inherits from a remote one, so you can define local overrides and additions to your package set easily.

1.6 FAQ

1.6.1 Is there an easier way to manage my package sets than to edit packages.json?

Yes. For example, the Spacchetti project uses Dhall to allow for easily maintaining a fork with separate type-checked sub-sets of packages and to allow for users to have local overrides of package sets.

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 Can I add a dependency which is not in the package set?

Not right now. We might add this feature in future, but for now, consider either:

- Adding your dependency to the package set if possible, or

- Creating your own custom package set

There used to be a feature called "add-from-bower", which could modify a package set from purescript/package-sets. This feature was removed as it proved to be very buggy and confusing for users, who thought that this feature would let them install arbitrary bower dependencies. See https://github.com/purescript/psc-package/issues/121.

1.6.4 Why is Add-From-Bower gone?

The feature named "add-from-bower" never did what users expected it to do: to install packages to a project from Bower.

Instead, what this feature did was to add packages to a package set from purescript/package-sets. However, due to the buggy nature of the feature, it was agreed that the feature should be removed and left up to other tools to handle. See https://github.com/purescript/psc-package/issues/121, where the top post describes how the Spacchetti package set adds Bower dependencies to the package set.

If you really would like to mix Psc-Package and Bower dependencies, you have two "real" options today. However, these will break if your packages are not all compatible, because this defeats the guarantees of having a package set.

1. Install the Bower dependencies anyway, and then only source the globs for individual dependencies, so you do not have conflicting definitions of other common modules. You can do this by supplying pass-through arguments: `psc-package build -- 'bower_componenents/purescript-my-lib/src/**/*.purs'`
2. Use the "Bower style" installation of Psc-Package2Nix: https://github.com/justinwoo/psc-package2nix/tree/master/test-bower-style. This will require that you use a system that can use the Nix package manager. See the link for more details.

1.6.5 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
```

1.6.6 Can I use Psc-Package with Nix?

Yes, now there is a solution for using Psc-Package with Nix: https://github.com/justinwoo/psc-package2nix. Please file issues in that project if you run into any.