Building a release

Joop Ringelberg 29-11-19 Version: 2

Introduction

Perspectives imports many packages. Some of these packages belong to the Perspectives project. In this text we describe a workflow that can be followed to produce a new release of Perspectives.

Two types of dependencies

Purescript projects use psc-package for their Purescript package management. For Perspectives, we have a custom package set (kept in the project **package-sets**). For each release, we must check whether any of the Purescript dependencies of **perspectives-core** have been changed; if so, we must bump their version (by creating a Git tag) and add that to package-sets. We then must make perspectives-core depend on that new package set. **Note** that perspectives-core is *the only project* that depends on other Perspectives Purescript packages.

However, to keep things simple, we have *all Perspectives Purescript Packages use the same package-set version*.

However, the projects making up Perspectives also use npm, for node package sets. Some dependencies of Perspectives projects are custom node packages (such as perspectives-proxy). For such dependencies we must bump their Git tag versions, too, and update the versions in the package.json files of projects that depend on them.

If a Purescript Perspectives package **has** Purescript Perspectives dependencies, it **must use** the Purescript Package Set.

If a Purescript Perspectives package **is** a Purescript Perspectives dependency, it **must be in** the Purescript Package Set.

Development between versions

In the period between two releases, imported packages will change, too. To prevent many sub-versions and unnecessary work, we replace the subdirectory of such an imported package in the core project by a symlink to a working directory holding a clone of the package project. To be precise: the .psc-package directory of perspectives-core holds a subdirectory for a particular version of the package-sets (for example: psc-12.5-perspectives). This directory in turn holds subdirectories for packages, that in turn have subdirectories for specific versions of those packages. Schematically:

- .psc-package

- psc-<version>-perspectives
  - <some-package>
    - <some-version>

We replace the <some-version> subdirectory with a symlink to the project root for <some-package>.

Each time we run psc-package:

```
$psc-package install
```

or:

```
$ psc-package build
```

these symlinks must be restored, at least for each package that has been changed since the previous release. We have a script for that: createPerspectivesLinks.sh (this script must be adapted to the new tag of package-sets for each release).

The above holds for Purescript projects. Similar reasoning applies to Node projects.

Releasing a version

Order the packages according to dependencies, such that if B depends on A, the order is A – B[1]. Then create new git tags in that order, because the new versions of a project's dependencies have to be recorded in its package.json before its final commit can be created to which the new tag is attached.

For Purescript projects, the new version of the Perspectives project package-set has to be set in the field "set" in their psc-package.json file, too.

1. Create a tag name for the new release of package-set before actually creating that tag (this is assuming that at least one of the Purescript packages has changed!). The tag name has the form pv<version>.

For each project (for which the dependencies have been updated and tagged!):

2. Check in all changes to Git.
3. Update the dependencies on Node Perspectives packages in package.json
4. **If a Purescript project:** update the field "set" in psc-package.json.
5. **[if a Purescript project with Perspectives dependencies]** Adapt createPerspectivesLinks.sh:
   a. Use the new tag of package-sets to descend into the right subdirectory of .psc-package
   b. Use the new tags of the updated packages to replace the right subdirectories.
6. Commit changes.

[1] An excel file with that ordered list can be found in perspectives-documentation/releaseNotes/package-versions.xlsx.

7. Check if a new release is in order. See *Check if a package has been updated after the previous release* on how to proceed.
   NOTE: InPlace must be released to a new version each time, since at least one of the dependencies will have changed (e.g. the core).
8. If so, and if the project has a package.json file:
   a. set a new version number in the field “version”. Use semantic versioning: 1.1.1.
   b. Commit the new version number to Git. Push to Github.
9. If so, apply the commands in *Create a tag*. Prefix the semantic version number with “v”.
10. Push to the origin!
11. If so, and if the project is a Purescript project **and** a dependency for another Purescript project, update its version in the project package-set in the file packages.json.

After all packages have been released:

12. Check if all Purescript projects with a new tag have that new tag in the project package-set in the file packages.json.
13. Commit package-sets and create a new tagged version, using the tag of step 1.
14. Update the release notes in the documentation project, use the new tag of the core.
15. In the documentation project, save all Word files *with a date later than the previous release* to PDF to the `docs` directory.
16. In the documentation project, save all .md files to .html in the `docs` directory.
17. Commit the documentation project and give it the same version tag as the core.

Check if a package has been updated after the previous release

How can we see whether a package (whether a Purescript or Node package) has been updated, since a previous release? By checking commits to that project and comparing them with the commit hash of the previous release.

```
git log --oneline --decorate --tags --no-walk
```

This command lists all tags for a project, showing the (short) commit hash. The most recent tag is that of the last release.

```
git log -5 --pretty=format:"%h - %cr, %s"
```

This command lists the last five commits in a project. If the commit hash of the last release is not the same as the hash on the top of the list of commits, a new tag is in order.

Create a tag

Create a tag for a Git project with the following two commands:

```
git tag -a v2.5.6 -m "some description"
git push origin v2.5.6
```

Delete a tag

Local:

```
git tag -d tagName
```

Remote:

```
git push --delete origin tagName
```

Move a tag

Created and pushed a tagged version and then discovered some uncommitted changes? Here is how to remedy that:

- Delete the tag on any remote before you push

  ```
  git push origin :refs/tags/<tagname>
  ```

- Commit your stuff.
- Push it to the server
- Replace the tag to reference the most recent commit

  ```
  git tag -fa <tagname> -m "Your message"
  ```

- Push the tag to the remote origin

  ```
  git push -f origin <tagname>
  ```

Building a distribution

We currently have a way to create installers for MacOSx and for Linux.

MacOSx

An installers is created by electron-builder. To prepare for a new version of the installer, create a draft on Github. Open the project `perspectives-react-integrated-client` and click Create a new Release on the right.

- Choose a package.json version and precede it with v. E.g. for version 0.7.0, enter v0.7.0. So choose an existing tag!
- Check "This is a pre-release".
- Save the draft.

Build

First, make sure the project builds correctly:

```
npm run build
```

Then either test or release.

Test

To create a directory of unpacked files (for testing purposes), run

```
npm run pack
```

To testrun the packed application, open dist/mac/InPlace.

Release

In the project `perspectives-react-integrated-client`, run the command:

```
npm run release
```

to create a new default distribution for Mac and push it to Github.

Linux

Careful: do not run ./createPerspectivesLinks.sh, as the dependent projects are not cloned to this Ubuntu disk! If dependencies in packages.json have been changed, just update npm.

- Start VirtualBox, the Ubuntu machine. The password for user 'Joop Ringelberg' is 'geheim'.
- Open a terminal and move into the project directory.

  ```
  cd /home/joop/code/perspectives-react-integrated-client
  ```

- Pull all changes from Github:

  ```
  git pull
  ```

- If dependencies in packages.json have been change:

  ```
  npm update
  ```

- Build:

  ```
  npm run build
  ```

Now test the application using `npm run electron&`. When all is fine, run the command:

```
npm run release
```

to create a new default distribution for Mac and push it to Github.

Finally, publish the release on Github.

A note on Github credentials. The file release.sh contains a Github credential. We need that to be able to push to Github. This file may never be published to Github!