The State of the OCaml Platform: September 2016

Louis Gesbert*(speaker) on behalf of the OCaml Platform team

The OCaml Platform combines the OCaml compiler toolchain with a coherent set of tools for build, documentation, testing and IDE integration. The project is a collaborative effort across the OCaml community, tied together by the OCaml Labs group in Cambridge and OCamlPro. The requirements of the Platform are guided by large industrial users such as Jane Street, Citrix, Docker, Facebook and LexiFi, as well accrued feedback from the OPAM project.

We have initially taken direction from major industrial users because these groups have a great deal of experience of using the language at scale. For the Platform to be considered successful, it has to be a viable product for these heavy users of OCaml. However, each of the users also have large codebases with distinct coding styles, and often have built their own extensive libraries to complement the OCaml standard library and open-source ecosystem. Pursuing goals desirable to industrial users has proven to benefit also more casual users.

This talk is a follow up from previous years. We will first update on the status of OPAM 2.0 and associated tools that constitute the Platform and highlight some of the advances around testing and documentation that have made notable progress this year. The goal of the Platform is to develop new tools around OPAM to make using it in development easier, more efficient, and more fun.

The maintainers of the projects described below come from different academic and industrial organisations. The goal of the talk is to place all of these projects into the context of the OCaml Platform, with due credit assigned.

1 OPAM 2.0

OPAM plays a central role in the tooling for the Platform, by providing a frontend that can control a concurrently installed set of compiler versions and package sets. Since its public release, thousands of packages and revisions have been added to the central repository, and growth continues strongly in 2016 (see Figure 1).

OPAM's lifecycle continues, and after the 1.2 branch that provided more flexible development workflows using 'opam pin' and many UI improvements [1], the upcoming

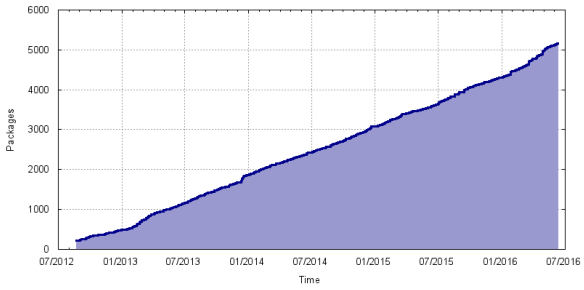

Figure 1: Growth in OPAM packages in the central repository. Note that this plots all unique versions, since OPAM supports installing older revisions of a given package.

OPAM 2.0 is a major rewrite that strives to make it more generic, flexible and robust, while supporting the new development workflows emerging from the expanding needs of the community.

This talk aims to present the bigger changes brought by OPAM 2.0, their impact on the users and the new possibilities offered, along with some technical background and motivation.

1.1 Metadata changes

Previous releases of OPAM used a repository providing collections of definitions of two distinct kinds, with different formats: compiler definitions, and package definitions. While this was convenient and fit well with the idea of "switches" that were initially designed to allow multiple compiler versions, the ultimate purpose of both formats was to provide the information required to install and use some software. OPAM 2.0 unifies these formats by representing compilers as normal package dependencies, thus deprecating the compiler-specific metadata.

Removing compiler definitions is part of a more general change in the paradigm of "switches", which now become generic package prefixes closer to what other package managers call "sandboxes". A few challenges had to be solved (switch creation, compiler upgrade, handling of a system compiler external to OPAM, and setting of the environment), but the solutions have generally led to more expressive package definitions.

New uses of these extensions started popping for spe-

*OCamlPro

cific use-cases even before they were complete: being able to upgrade the compiler became central to the compiler benchmarking system used for the development of *flambda*, and several package bug workarounds (including one Perl-related one!) that were formerly hardcoded in OPAM 1.x are now in the package definitions.

CLI extensions This brings several new possibilities to the command-line interface: one can create an empty switch and have no packages with a special "compiler" status, or conversely define a switch including more packages, *e.g* OCaml and Coq together. It is also more generic and useful in contexts not directly related to OCaml. Compiler hackers can also use 'opam pin' on their compiler to benefit from a development workflow similar to that of usual package developers.

1.1.1 Data layout

Internally, the way OPAM loads its state, updates it, and locks the appropriate parts has been rewritten. Community members such as David Allsopp have been contributing high-quality support for Windows in the core tool.

File formats The general file formats are unchanged, but easier to programmatically handle and extend thanks to a new lens-like parsing/printing implementation. New fields are available:

- they can include `url` and `description`, allowing single-file package definitions
- triple-quote-delimited strings are allowed for avoiding escaping issues in long strings (e.g. package descriptions) or complex commands
- New fields: `setenv`, `extra-sources`, `build-subdir`, `provides`, `synopsis`, `description` and `x-foo` fields for external use
- Dependencies can now be defined conditionally depending on the variables set in the switch and the current package; it is for example possible to express depending on another package "at the same version", making it possible to use generic OPAM files for multiple package versions.
- Upon installation, a package can create a `<pkg>.config` file that indicates a dependency on a system file. The system file will then be checked by OPAM and the package marked for reinstallation if it changes. This allows for better integration with OS package managers such as Homebrew.
- The `remove` field is no longer needed since OPAM now tracks the files installed by every package.

1.1.2 OPAM root filesystem layout

Per-switch remotes Each switch can define define its own selection of repositories, and its universe of available packages. Previous versions of OPAM made repository collections global, which has resulted in poor isolation when using switches with multiple remotes.

Global package caching Similarly, there is no more caching of packages in `~/.opam/packages/`, since the definitions can now be switch specific. There is now a generic cache of archives, stored globally by its hash. The consistency of the switches is also better ensured by always keeping an up-to-date cache of the package definitions as they were used for installing the package.

Local switches Switch directories can now be defined anywhere, including outside of the `opam root`. This separates the local switch prefix data from OPAM's metadata, and places it in a `.opam-switch` in the local project subdirectory. The current state of the switch – *i.e.* what packages are installed, pinned, or part of the compiler – is now in a single human-readable file 'switch-state' in there, rather than scattered in different files.

New Workflows The new features above permit several new workflows that we will describe during the talk, such as local project sandboxes that use development definitions directly from the project, and efficient build/test/documentation cycles directly from an editor.

1.1.3 Tooling

All of the OPAM commands now support structured JSON output to help mechanically script the result of operations. There is a largely rewritten API with 6 smaller libraries (instead of 2), having fewer dependencies and smaller self-contained modules. State loading is also decoupled (global, repositories and switch) and much clearer to use from helper tools.

Tools for automatic migration of the repository (including conversion of compiler definitions to packages) are provided, as well as an extended linting tool. Packages can be checked for correct behaviour using command wrappers (via generic support for command wrappers in OPAM, and shell wrappers restricting write access to commands in the build, install and remove phases).

There are ongoing trees for Windows support and for cross-compilation, but the design has still to be finalised.

2 Documentation

The `codoc` tool has been simplified into `odoc`, which is a command-line tool that takes interface files and outputs cross-referenced module documentation. This is being integrated into OPAM via a plugin that builds per-switch documentation from the universe of installed pack-

ages.

3 Infrastructure

Automated testing has been made significantly easier via a regularly built matrix of Docker containers, available at `https://hub.docker.com/r/ocaml/opam`. These are regularly built on several versions of OCaml (4.00.1, 4.01.0, 4.02.3, 4.03.0 and 4.03.0-flambda), and several distributions (Debian, Ubuntu, CentOS, Fedora, Oracle Linux, RHEL and Alpine), and ARMv7 (Raspbian).

These containers can be used in several hosted continuous integration services such as CircleCI and Travis CI. The `https://github.com/ocaml/ocaml-ci-scripts` repository centralises these scripts for easy integration into individual projects. Windows and FreeBSD support is being worked on for these containers.

4 Conclusion

OPAM 2.0 represents a significant milestone in the OCaml Platform by improving the underlying tooling to support more flexible workflows for industrial users. The familiar existing schemes continue to work, but the compilers-as-packages and more flexible local sandbox support makes it easier to distribute complex OCaml packages that can be run with a single OPAM invocation. The addition of maintained containers makes it easier than ever to build and test across the matrix of supported distributions and compiler revisions.

References

[1] MADHAVAPEDDY, A., CHAUDHRY, A., GAZAGNAIRE, T., YALLOP, J., AND SHEETS, D. The state of the ocaml platform: September 2015. In *Proc. ICFP OCaml User and Developer Workshop* (Sept. 2015).