Creating packages for OPAM

(C) OCamlPro SAS — INRIA

18/09/2012

In this tutorial, you will learn how to package an OCaml library or software for OPAM. The first section will introduce you the *ounit* OPAM package. Albeit simple, it is a real example of what an OPAM package is, and actually most packages in the OPAM repository are that simple. The second section will be a comprehensive guide illustrated by more complicated examples of real packaging cases.

An important thing to keep in mind is that OPAM is (at least for now) **only** a package manager, as opposed to — for example — the *oasis/odb* suite, which includes a build system (*oasis*) as well as a package manager (*odb*). This means that OPAM cannot help you to actually build your ocaml software or library, to do so, you need to use dedicated tools such as *ocamlbuild*, *oasis*, *omake* and others. OPAM packages are just built the same way you would build the software yourself (with shell commands).

Contents

The *ounit* OPAM package

Let get started by learning from the *ounit* OPAM package. This is a widely used OCaml library to create unit tests for OCaml projects, and is a perfect example of a minimalistic yet complete package.

By reading about *ounit* on the upstream website, you learn that it is a library to create unit tests, that its last version is *1.1.2*, and that it depends on *ocamlfind*. You know at which URL you can download its source tarball, and you can compute the *md5sum* of the this source tarball (by running `md5sum ounit-1.1.2.tar.gz`).

You also learn that to build and install it, you have to do

```
$ make build
$ make install
```

And that's all the information you need to build an OPAM package.

A minimum OPAM package is a directory containing three files, respectively *descr*, *opam*, and *url*. The name of the directory is `<package-name>.<package-version>`. In our case, the directory will thus be `ounit.1.1.2` and contain the following files:

- `descr`

```
Unit testing framework inspired by the JUnit tool and the HUnit tool
```

- `opam`

```
opam-version: "1"
maintainer: "contact@ocamlpro.com"
build: [
  ["make" "build"]
  ["make" "install"]
]
remove: [
  ["ocamlfind" "remove" "oUnit"]
]
depends: ["ocamlfind"]
```

- `url`

```
archive: "http://forge.ocamlcore.org/frs/download.php/886/ounit-1.1.2.tar.gz"
checksum: "14e4d8ee551004dbcc1607f438ef7d83"
```

Notes

`descr`

This file is pure text. Its first line will be used as a short description of the package, it is what is displayed for each package when you do `opam list`. The whole text contained in there is displayed when you do `opam search <package>`. Therefore you should put a short meaningful description on the first line, and a long description starting from the second line.

`opam`

The full ABNF specification of the syntax for *opam* files is available in OPAM's specification document. In this file, `opam-version` MUST be `1`, and you should put your email in the `maintainer` field. `build` is has OCaml type `string list list`, and represent the build instructions. Here,

```
$ make build
$ make install
```

gets translated into

```
build: [
  ["make" "build"]
  ["make" "install"]
]
```

You should adapt this to the required commands to build your package, each line you have to type on your shell corresponding here to a `string list`.

The `remove` field follows the same syntax as the `build` field. The `depends` field is a `string list` of dependencies, each dependency being another OPAM package. Here *ounit* depends only on *ocamlfind*.

`url`

This file contains at least one `archive` line containing the URL of the source package, and optionally a `checksum` line that must contain the md5sum of the source package if it is present. It is good practice to systematically add a checksum line to your packages, unless the source package has no fixed version (the typical example being a source package hosted on *github* with no tags). This checksum will be checked when creating and installing the package.

Advanced OPAM packaging guide

This section will be as comprehensive as possible on the art of creating OPAM packages, but in case of ambiguities, the ABNF syntax documentation has priority.

Everything has already be said about files `descr` and `url`, the additions of this section involve only the `opam` files.

OPAM variables

OPAM maintains a set of variables (key value pairs) that can be used in `opam` files and that will be substituted by their values on package creation. The list of variables that can be used in `opam` files can be displayed by doing `opam config -list-vars`. The following example shows the `build` section of package *ocamlnet* that use the variable `bin`:

```
build: [
  ["./configure" "-bindir" "%{bin}%"]
  ["make" "all"]
  ["make" "opt"]
  ["make" "install"]
]
```

In this case, `%{bin}%` will be substituted by the value of the `bin` variable.

Optional dependencies

In the OPAM tutorial, we mentionned the ability of OPAM to specify optional dependencies for packages. We said thatif a package has optional dependencies, they will not be installed automatically but they will be taken into account if they are present before the package (that optionally depends on them). Let us see how it works from the point of view of the package. The following example shows the `opam` file of package *lwt*:

```
opam-version: "1"
maintainer: "contact@ocamlpro.com"
build: [
  ["./configure" "--%{conf-libev:enable}%-libev" "--%{react:enable}%-react" "--%{ssl:enable}
  ["make" "build"]
  ["make" "install"]
]
remove: [
  ["ocamlfind" "remove" "lwt"]
]
depends: ["ocamlfind"]
depopts: ["base-threads" "base-unix" "conf-libev" "ssl" "react"]
```

You can notice a new field, `depopts`. It contains the list of optional dependencies, specified in the same format as for the `depends` field.

You can also notice a new syntax for substitutions of the form `%{<package>:enable}%`. The semantic of this is that if *package* is installed, the pattern will be replaced by `enable`, otherwise by `disable`. This ease the building of lines of type `./configure --enable-<feature1> --disable-<feature2>`.

Version constraints

If a package depends (respectively optionally depends) on a specific version of a package, this can be specified by using the following syntax for the field `depends` (respectively `depopts`) of the `opam` file:

```
depends: [ "ocamlfind" "re" "uri" "ounit" ]
depopts: [ "async" {= "108.00.02"} "lwt" "mirage-net" ]
```

The above example shows two fields of the `opam` file of package *cohttp*, that optionally depends of the version 108.00.02 of package *async*.

The OPAM specification document specifies the format used by the `depends` and `depopts` fields. As there is much more to say about version constraints, you should read it to learn how to write very fine grained version constraints.

Manually installing binaries or libaries

Most of the time, when a package is built, binaries and/or libraries that it provides are installed automatically. But sometimes a source package does not install a binary that it produced because it has not been included its `make install` command (or equivalent). For example, the package *xmlm* has this following `opam` file:

```
opam-version: "1"
maintainer: "contact@ocamlpro.com"
build: [
  ["ocaml" "setup.ml" "-configure" "--prefix" "%{prefix}%"]
  ["ocaml" "setup.ml" "-build"]
  ["ocaml" "setup.ml" "-install"]
]
remove: [
  ["ocamlfind" "remove" "xmlm"]
]
depends: ["ocamlfind"]
```

and has additional file `files/xmlm.install`:

```
lib: []
bin: ["_build/test/xmltrip.native" {"xmltrip"}]
toplevel: []
misc: []
```

This has the semantic: *install the file of path `_build/test/xmltrip.native` relatively to the root of the source package into the directory returned by the command `opam config -var bin` under the name `xmltrip`*. You can install additional libraries and toplevels the same way. For comprehensive information about this facility refer to the OPAM specification document.

Compiler version constraints

Some packages require a specific OCaml version to work and thus can only be installed under specific compiler versions. To specify such a constraint in a package, you can add a field `ocaml-version: [ < "4.00.0" ]` to the `opam` file. This particular constraint implies that the package cannot be built or installed under OCaml 4.00.0 or later.

Where to go from here

Although this tutorial covered most packaging cases, there are still packages that requires more tuning that what have been described above. If you find yourself stuck trying to package a software or a library, please read the OPAM specification (you will find it in the `doc` directory in the OPAM tarball) and/or read existing OPAM package descriptions for inspiration.