OPAM: a Package Management Systems for OCaml Developer Manual (version 1.0)

Thomas GAZAGNAIRE

thomas.gazagnaire@ocamlpro.com

February 20, 2013

Contents

Overview

OPAM is a source-based package manager for OCaml. It supports multiple simultaneous compiler installations, flexible package constraints, and a Git-friendly development workflow.

A package management system has typically two kinds of users: *end-users* who install and use packages for their own projects; and *packagers*, who create and upload packages. End-users want to install on their machine a consistent collection of *packages* – a package being a collection of OCaml libraries and/or programs. Packagers want to take a collection of their own libraries and programs and make them available to other developpers.

This document describes the design of OPAM to answer both of these needs.

Conventions

In this document, `$home`, `$opam`, `$package` and `$path` are assumed to be defined as follows:

- `$home` refers to the end-user home path, typically `/home/thomas/` on linux and `/Users/thomas/` on OSX.

- `$opam` refers to the filesystem subtree containing the client state. Default directory is `$home/.opam`.

- `$package` refers to a path in the packager filesystem, where lives the collection of libraries and programs he wants to package.

- `$path` refers to a list of paths in the packager filesystem, where lives the collection of programs (`ocamlc`, `ocamldep`, `ocamlopt`, `ocamlbuild`, ...).

User variables are written in capital letters, prefixed by `$`. For instance package names will be written `$NAME`, package versions `$VERSION`, and the version of the ocaml compiler currently installed `$SWITCH`.

This document is organized as follows: Section 1 describes the core of OPAM, e.g. the management of packages. Section 2 describes how repositories are handled, Section 3 focus on compiler switches and finally Section 4 explain how packages can define configuration variables (which can be later used by the build system).

1 Managing Packages

1.1 State

The client state is stored on the filesystem, under `$opam`. All the configurations files, libraries and binaries related to a specific instance of the OCaml compiler in `$opam/$SWITCH`, where `$SWITCH` is the name of that specific compiler instance. See Section 3 for more details about compiler switches.

- `$opam/config` is the main configuration file. It defines the OPAM version, the repository addresses and the current compiler version. The file format is described in §1.2.3.

- `$opam/opam/$NAME.$VERSION.opam` is the OPAM specification for the package `$NAME` with version `$VERSION` (which might not be installed). The format of OPAM files is described in §1.2.4.

- `$opam/descr/$NAME.$VERSION` contains the description for the version `$VERSION` of package `$NAME` (which might not be installed). The first line of this file is the package synopsis.

- `$opam/archives/$NAME.$VERSIONopam.tar.gz+` contains the source archives for the version `$VERSION` of package `$NAME`. This archive might be a bit different from the upstream library as it might have been repackaged by OPAM to include additional files.

- `$opam/$SWITCH/installed` is the list of installed packages for the compiler instance `$SWITCH`. The file format is described in §1.2.2.

- `$opam/$SWITCH/config/$NAME.config` is a platform-specific configuration file of for the installed package `$NAME` with the compiler instance `$SWITCH`. The file format is described in §1.2.3. `$opam/$SWITCH/config/` can be shortened to `$config/` for more readability.

- `$opam/$SWITCH/install/$NAME.install` is a platform-specific package installation file for the installed package `$NAME` with the compiler instance `$SWITCH`. The file format is described in §1.2.5. `$opam/$SWITCH/install` can be shortened to `$install/` for more readability.

- `$opam/$SWITCH/lib/$NAME/` contains the libraries associated to the installed package `$NAME` with the compiler instance `$SWITCH`. `$opam/$SWITCH/lib/` can be shortened to `$lib/` for more readability.

- `$opam/$SWITCH/doc/$NAME/` contains the documentation associated to the installed package NAME with the compiler instance `$SWITCH`. `$opam/SWITCH/doc/` can be shortened to `$doc/` for more readability.

- `$opam/$SWITCH/bin/` contains the program files for all installed packages with the compiler instance `$SWITCH`. `$opam/$SWITCH/bin/` can be shortened to `$bin/` for more readability.

- `$opam/$SWITCH/build/$NAME.$VERSION/` is a tempory folder used to build package `$NAME` with version `$VERSION`, with compiler instance `$SWITCH`. `$opam/$SWITCH/build/` can be shortened to `$build/` for more readability.

- `$opam/$SWITCH/reinstall` contains the list of packages which has been changed upstream since the last upgrade. This can happen for instance when a packager uploads a new archive or fix the OPAM file for a specific package version. Every package appearing in

this file will be reinstalled (or upgraded if a new version is available) during the next upgrade when the current instance of the compiler is `$SWITCH`. The file format is similar to the one described in §1.2.2.

- `$opam/$SWITCH/pinned` contains the list of pinned packages. The file format is described in §**??**.

- `$opam/$SWITCH/pinned.cache.` contains cached information for cached packages. OPAM uses it on update to check which package needs to be upgraded.

1.2 Files

1.2.1 General Syntax of OPAM files

Most of the files in the client and server states share the same syntax defined in this section.

Base types The base types for values are:

- `BOOL` is either `true` or `false`
- `STRING` is a doubly-quoted OCaml string, for instance: `"foo"`, `"foo-bar"`, ...
- `SYMBOL` contains only non-letter and non-digit characters, for instance: `=`, `<=`, ... Some symbols have a special meaning and thus are not valid `SYMBOL`s: "`( ) [ ] { } :`".
- `IDENT` starts by a letter and is followed by any number of letters, digit and symbols, for instance: `foo`, `foo-bar`, ...

Compound types Types can be composed together to build more complex values:

- `X Y` is a space-separated pair of value.
- `X | Y` is a value of type either `X` or `Y`.
- `?X` is zero or one occurrence of a value of type `X`.
- `X+` is a space-separated list of values of at least one value of type `X`.
- `X*` is a space-separated list of values of values of type `X` (it might contain no value).

All structured OPAM files share the same syntax:

```
<file>  := <item>*

<item>  := IDENT : <value>
         | ?IDENT: <value>
         | IDENT STRING { <item>+ }

<value> := BOOL
         | INT
         | STRING
         | SYMBOL
         | IDENT
         | [ <value>+ ]
         | value { <value>+ }
```

1.2.2 Package List: `installed`, `reinstall` and `update`

The following configuration files: `$opam/$SWITCH/installed`, `$opam/$SWITCH/reinstall`, and `$opam/repo/$REPO/updated` follow a very simple syntax. The file is a list of lines which contains a space-separated name and a version. Each line `$NAME $VERSION` means that the version `$VERSION` of package `$NAME` has been compiled with the compiler instance `$SWITCH` and has been installed on the system in `$lib/$NAME` and `$bin/`.

For instance, if `batteries` version `1.0+beta` and `ocamlfind` version `1.2` are installed, then `$opam/$SWITCH/installed` will contain:

```
batteries 1.0+beta
ocamlfind 1.2
```

1.2.3 Global Configuration File: `config`

`$opam/config` follows the syntax defined in §1.2.1 with the following restrictions:

```
<file> :=
    opam-version: "1"
    repositories: [ STRING+ ]
    switch: STRING
    cores: INT
```

The field `opam-version` indicates the current OPAM format.
The field `repositories` contains the list of OPAM repositories.
The field `switch` corresponds to the current compiler instance.
The field `cores` is the number of parallel process that OPAM will use when trying to build the packages.

1.2.4 Package Specification files: `*.opam`

`$opam/opam/$NAME.$VERSION.opam` follows the syntax defined in §1.2.1 with the following restrictions:

```
<file> :=
    opam-version: "1"
    ?name:          STRING
    ?version:       STRING
    maintainer:     STRING
    ?homepage:      STRING
    ?authors:       [ STRING+ ]
    ?doc:           STRING
    ?license:       STRING
    ?tags:          [ STRING+ ]
    ?subst:         [ STRING+ ]
    ?patches:       [ (STRING ?<filter>)+ ]
    ?build:         commands
    ?build-doc:     commands
    ?build-test:    commands
    ?remove:        commands
    ?depends:       [ <and-formula(package)>+ ]
    ?depopts:       [ <or-formula(package)>+ ]
    ?depexts:       [ [STRING+] [STRING+ ]+
```

```
      ?conflicts:      [ <package>+ ]
      ?os:             [ <formula(os)>+ ]
      ?ocaml-version:  [ <and-formula(constraint)>+ ]
      ?libraries:      [ STRING+ ]
      ?syntax:         [ STRING+ ]

<argument>         := STRING
                    | IDENT

<command>          := [ (<argument> ?<filter>)+ ] ?<filter>

<commands>         := <command>
                    | [ <command>+ ]

<filter>           := { and-formula(<argument> <comp> <argument>) }

<formula(x)>       := <formula(x)> '&' <formula(x)>
                    | <formula(x)> '|' <formula(x)>
                    | ( <formula(x)> )
                    | <x>

<package>          := STRING
                    | STRING { <and-formula(constraint)> }

<constraint>       := <comp> STRING
<comp>             := '=' | '<' | '>' | '>=' | '<='

<and-formula(x)> := <x> <and-formula(x)>
                    | <formula(x)>

<or-formula(x)>  := <x> <or-formula(x)>
                    | <package(x)>

<os>               := STRING
                    | '!' STRING
```

- The first line specifies the OPAM version.

- The content of `name` is $NAME, the content of `version` is $VERSION. Both fields are optional are they can be inferred from the filename.

- The content of `maintainer` is the contact address of the package maintainer.

- The `license`, `homepage doc` and `authors` fields are optional. `doc` should be the address of the online documentation for the package.

- The `tags` field is optional contains a list of tags to classify the package.

- The content of `subst` is the list of files to substitute variable (see §4.2.1 for the file format and §4 for the semantic of file substitution).

- The content of `patches` is a list of patches to be applied. Substitutions happen before patch application, so patches can contain strings which will substituted.

- The content of `build` is the list of commands to run in order to build the package libraries. The build script should build all the libraries and syntax extensions exported by the

package and it should produce the platform-specific configuration and install files (e.g. `$NAME.config` and `$NAME.install`, see §1.2.3 and §1.2.5).

Each command and command argument is substituted (see §4.2.1 and §4, with the identifier `X` being equivalent to the string `"%{X}%"`) and can be followed by an optional filter, whose evaluation will result in the command (or the command argument) being executed or not. A typical example is OS-related filters, where we can choose to execute commands depending on the current OS:

```
build: [
  ["mv" "Makefile.unix" "Makefile"]  {os != "win32"}
  ["mv" "Makefile.win32" "Makefile"] {os  = "win32"}
  [make]
]
```

- `build-doc` is optional and describes how the documentation is built.

- `build-test` is optional and describes how the tests are built and run.

- The content of `remove` is the command to run before deleting the installed file.

- The `depends`, `depots` and `conflicts` fields contain formulas over package names, optionally parametrized by version constraints. Some examples or package formula:

  - A package name: `"foo"`;
  - A package name with version constraints: `"foo" {>= "1.2" & <= "3.4"}`

  `depends` is an *AND* formula, which means that top-level `&` are not mandatory. For instance, `"foo" {<= "1.2"} ("bar" | "gna" {= "3.14"})` has the following semantic: "*both any version of package* `"foo"` *lesser or equal to* 1.2 *and either any version of package* `"bar"` *or the version* 3.14 *of package* `"gna"`."

  The `optdeps` field contains a *OR* formula over package names, which means that top-level `|` are not mandatory. This field express optional dependencies that OPAM will not try to install. However, when installing a new package it will check if it is an optional dependency of already installed packages. If it is the case, it will re-install the packages (and their transitive forward-dependency closure).

- The `depexts` field is optional and contains tags describing the external dependencies.

- The `os` and `ocaml-version` fields are optional constraints over the supported OS and compiler version for this package. In case the filter is not valid, the package is disabled.

- The `libraries` and `syntax` fields contain the libraries and syntax extensions defined by the package. See Section 4 for more details.

1.2.5   Package installation files: `*.install`

`$opam/$SWITCH/install/NAME.install` follows the syntax defined in §1.2.1 with the following restrictions:

```
<file> :=
    opam-version: "1"
    ?lib:  [ STRING+ ]
    ?bin:  [ <mv>+ ]
    ?doc:  [ STRING+ ]
    ?misc: [ <mv>+ ]

<mv> := STRING
     | STRING { STRING }
```

- Files listed under `lib` are copied to `$lib/$NAME/`.

- Files listed under `bin` are copied to `$bin/` (they can be renamed using `$SRC { $DST }`; in this case `$SRC` should be a simple filename, ie. it should not start with a directory name).

- Files listed under `doc` are copied to `$doc/$NAME/`.

- Files listed under `misc` should be processed as follows: for each pair `$SRC { $DST }`, the tool should ask the user if he wants to install `$SRC` to the absolute path `$DST`.

- OPAM will try to install all the file in sequence, and it will fail in case a source filename is not available. To tell OPAM a source filename might not be generated (because of byte/native constraints or because of optional dependencies) the source filename should start by `?`. Remark: it is much cleaner if the underlying build-system can generate the right `.install` files, containing the existing files only.

1.2.6   Pinned Packages: `pinned`

`$opam/$SWITCH/pinned` contains a list of lines of the form:

`<name> <kind> <path>`

- `<name>` is the name of the pinned package

- `<kind>` is the kind of pinning. This could be `version`, `local`, `git` or `darcs`.

- `<path>` is either the version number (if kind is `version`) or the path to synchronize with.

1.3   Commands

1.3.1   Creating a Fresh Client State

When an end-user starts OPAM for the first time, he needs to initialize `$opam/` in a consistent state. In order to do so, he should run:

```
$ opam init [--kind $KIND] $REPO $ADDRESS [--comp $VERSION]
```

Where:

- `$KIND` is the kind of OPAM repository (default is `http`);

- `$REPO` is the name of the repository (default is `default`); and

- `ADDRESS` is the repository address (default is `http://opam.ocamlpro.com/pub`).

- `$COMP` is the compiler version to use (default is the version of the compiler installed on the system).

This command will:

1. Create the file `$opam/config` (as specified in §1.2.3)

2. Create an empty `$opam/$SWITCH/installed` file, `$SWITCH` is the version from the OCaml used to compile `$opam`. In particular, we will not fail now if there is no `ocamlc` in `$path`.

3. Initialize `$opam/repo/$REPO` by running the appropriate operations (depending on the repository kind).

4. Symlink all OPAM and description files (ie. create a symbolic link from every file in `$opam/repo/$REPO/opam/` to `$opam/opam/` and from every file in `$opam/repo/$REPO/descr/` to `$opam/descr/`).

5. Create `$opam/repo/index` and for each version `$VERSION` of package `$NAME` appearing in the repository, append the line '`$REPO $NAME $VERSION`' to the file.

6. Create the empty directories `$opam/archives`, `$lib/`, `$bin/` and `$doc/`.

1.3.2 Listing Packages

When an end-user wants to have information on all available packages, he should run:

```
$ opam list
```

This command will parse `$opam/$SWITCH/installed` to know the installed packages, and `$opam/opam/*.opam` to get all the available packages. It will then build a summary of each packages. The description of each package will be read in `$opam/descr/` if it exists.

For instance, if `batteries` version `1.1.3` is installed, `ounit` version `2.3+dev` is installed and `camomille` is not installed, then running the previous command should display:

```
batteries   1.1.3   Batteries is a standard library replacement
ounit     2.3+dev   Test framework
camomille      --   Unicode support
```

1.3.3 Getting Package Info

In case the end-user wants a more details view of a specific package, he should run:

```
$ opam info $NAME
```

This command will parse `$opam/$SWITCH/installed` to get the installed version of `$NAME`, will process `$opam/repo/index` to get the repository where the package comes from and will look for `$opam/opam/$NAME.*.opam` to get available versions of `$NAME`. It can then display:

```
package: $NAME
version: $VERSION
versions: $VERSION1, $VERSION2, ...
libraries: $LIB1, $LIB2, ...
syntax: $SYNTAX1, $SYNTAX2, ...
```

```
repository: $REPO
description:
  $SYNOPSIS

  $LINE1
  $LINE2
  $LINE3
  ...
```

1.3.4 Installing a Package

When an end-user wants to install a new package, he should run:

```
$ opam install $NAME
```

This command will:

1. Compute the transitive closure of dependencies and conflicts of packages using the dependency solver (see §1.3.9). If the dependency solver returns more than one answer, the tool will ask the user to pick one, otherwise it will proceed directly. The dependency solver should also mark the packages to recompile.

2. The dependency solver sorts the collections of packages in topological order. Then, for each of them do:

   (a) Check whether the package is already installed by looking for the line `$NAME $VERSION` in `$opam/$SWITCH/installed`. If not, then:

   (b) Look into the archive cache to see whether it has already been downloaded. The cache location is: `$opam/archives/$NAME.VERSION.tar.gz`

   (c) If not, process `$opam/repo/index/` to get the repository `$REPO` where the archive is available and then ask the repository to download the archive if necessary..
   Once this is done, symlink the archive in `$opam/archives`.

   (d) Decompress the archive into `$build/$NAME.$VERSION/`.

   (e) Substitute the required files.

   (f) Run the list of commands to build the package with `$bin` in the path.

   (g) Process `$build/$NAME.$VERSION/$NAME.install` to install the created files. The file format is described in §1.2.5.

   (h) Install the installation file $build/$NAME.$VERSION/$NAME.install in $install/ and the configuration file $build/$NAME.$VERSION/$NAME.config in $config/.

1.3.5 Updating Index Files

When an end-user wants to know what are the latest packages available, he will write:

```
$ opam update
```

This command will follow the following steps:

- Update each repositories in `$opam/config`.

- For each repositories in `$opam/config`, process `$opam/repo/$REPO/updated` and update `$opam/repo/index`, `$opam/opam/` and `$opam/desc` accordingly (ie. add the right lines in `$opam/repo/index` and create the missing symlinks). Here, the order in which the repositories are specified is important: the first repository containing a given version for a package will be the one providing it (this can be changed manually by editing `$opam/repo/index` later).

- For each line `$REPO $NAME $VERSION` in `$opam/repo/index`, if the version `$VERSION` of package `$NAME` has been modified upstream (ie. if the line `$NAME $VERSION` appears in `$opam/repo/$REPO/$updated`) and if the package is already installed (ie. it appears in `opam/$SWITCH/installed`), then update `$opam/$SWITCH/reinstall` accordingly (for each compiler version `$SWITCH`).

  Packages in `$opam/$SWITCH/reinstall` will be reinstalled (or upgraded if a new version is available) on the next `opam upgrade` (see §1.3.6), with `$SWITCH` being the current compiler version when the upgrade command is run.

- Delete each `$opam/repo/$REPO/$updated`

1.3.6 Upgrading Installed Packages

When an end-user wants to upgrade the packages installed on his host, he will write:

```
$ opam upgrade
```

This command will:

- Call the dependency solver (see §1.3.9) to find a consistent state where **most** of the installed packages are upgraded to their latest version. Moreover, packages listed in `$opam/$SWITCH/reinstall` will be reinstalled (or upgraded if a new version is available). It will install each non-installed packages in topological order, similar to what it is done during the install step, See §1.3.4.

- Once this is done the command will delete `$opam/$SWITCH/reinstall`.

1.3.7 Uploading Packages

When a packager wants to create a package, he should:

1. create `$package/$NAME.$VERSION.opam` containing in the format specified in §1.2.4.

2. create a file describing the package

3. make sure the build scripts:

   - build the libraries and packages advertised in `$package/$NAME.$VERSION.opam`
   - generates a valid `$package/$NAME.install` containing the list of files to install (the file format is described in 1.2.5).
   - generates a valid `$package/$NAME.config` containing the configuration flags for libraries exported by this package (the file format is described in 4.2.2).

4. create an archive `$NAME.$VERSION.tar.gz` with the sources he wants to distribute.

5. run the following command:

```
$ opam upload --opam $OPAM --descr $DESCR --archive $ARCHIVE $REPO
```

This command will parse `$OPAM` to get the package name and version and it will:

- move `$OPAM` to `$opam/repo/$REPO/upload/$NAME.$VERSION.opam`
- move `$DESCR` to `$opam/repo/$REPO/descr/$NAME.$VERSION`
- move `$ARCHIVE` to `$opam/repo/$REPO/archives/$NAME.$VERSION.tar.gz`

It will then perform the necessary operation (depending on the repository kind) to upload the files upstream.

1.3.8 Removing Packages

When the user wants to remove a package, he should write:

```
$ opam remove $NAME
```

This command will check whether the package `$NAME` is installed, and if yes, it will display to the user the list packages that will be uninstalled (ie. the transitive closure of all forward-dependencies). If the user accepts the list, all the packages should be uninstalled, and the client state should be let in a consistent state.

1.3.9 Dependency Solver

Dependency solving is a hard problem and we do not plan to start from scratch implementing a new SAT solver. Thus our plan to integrate (as a library) the Debian depency solver for CUDF files, which is written in OCaml.

- the dependency solver should run on the client; and

- the dependency solver should take as input a list of packages (with some optional version information) the user wants to install, upgrade and remove and it should return a consistent list of packages (with version numbers) to install, upgrade, recompile and remove.

2 Managing Repositories

2.1 State

Configuration files for OPAM repositories `REPO` are stored in `$opam/repo/$REPO`. Repositories can be of different kinds (stored on the local filesystem, available via HTTP, stored under git, ...); they all share the same filesystem hierachy, which is updated by different operations, depending on the repository kind.

- `$opam/repo/$REPO/config` contains the configuration off the repository `$REPO`. The format of repository config files is described in §**??**.

- `$opam/repo/$REPO/opam/$NAME.$VERSION.opam` is the OPAM specification for the package `$NAME` with version `$VERSION` (which might not be installed). The format of OPAM files is described in §1.2.4.

- `$opam/repo/$REPO/descr/$NAME.$VERSION` contains the textual description for the version `$VERSION` of package `$NAME` (which might not be installed). The first line of this file is the package synopsis.

- `$opam/repo/$REPO/archives/$NAME.$VERSION.tar.gz` contains the source archives for the version `$VERSION` of package `$NAME`. This folder is populated when a package needs to be downloaded.

- `$opam/repo/$REPO/updated` contains the new available packages which have not yet been synchronized with the client state. This file is created on update.If the file empty, this means that the client state is up-to-date. The file format is the same as the one described in §1.2.2.

- `$opam/repo/$REPO/upload/$NAME.$VERSION/` contains the OPAM, description and archive files to upload to the OPAM repository for the version `$VERSION` of package `$NAME`.

2.2 Files

2.2.1 Index of packages

`$opam/repo/index` follows a very simple syntax: each line of the file contains a space separated list of words `$NAME $REPO` specifying that all the versions of package `$NAME` are available in the OPAM repository `$REPO`. The file contains information on all available packages (e.g. not only on the installed one).

For instance, if `batteries` version `1.0+beta` is available in the `testing` repository and `ocamlfind` version `1.2` is available in the `default` and testing repositories (where `default` is one being used), then `$opam/repo/index` will contain:

```
batteries testing
ocamlfind default
```

2.3 Commands

2.3.1 Managing OPAM repository

When the user wants to manage OPAM repositories, he should write:

```
$ opam repository list # 'opam repository' works as well
$ opam repository add [--kind $KIND] $REPO $ADRESS
$ opam repository remove $REPO
```

- `list` lists the current repositories by looking at `$opam/config`

- `add [--kind $KIND] $REPO $ADDRESS` initializes `$REPO` as described in §1.3.1.

- `remove $REPO` deletes `$opam/repo/$REPO` and removes `$REPO` from the `repositories` list in `$opam/config`. Then, for each package in `$opam/repo/index` it updates the link between packages and repositories (ie. it either deletes packages or symlink them to the new repository containing the package).

3 Managing Compiler Switches

This milestone focus on the support of multiple compiler versions.

3.1 State

The state of OPAM repositories is extended with the directory `$opam/repo/$repo/compiler` containing the compiler description files. When a repository is updated, this directory is updated as well.

3.2 Files

3.2.1 Compiler Description Files

For each compiler switch `SWITCH`, the client state will be extended with the following files:

- `$opam/compilers/SWITCH.comp`

The syntax of `.comp` files follows the one described in §1.2.1 with the following restrictions:

```
<file> :=
    opam-version: "1"
    name:        STRING
    src:         STRING
    make:        [ STRING+ ]
    ?patches:    [ STRING+ ]
    ?configure:  [ STRING+ ]
    ?bytecomp:   [ STRING+ ]
    ?asmcomp:    [ STRING+ ]
    ?bytelink:   [ STRING+ ]
    ?asmlink:    [ STRING+ ]
    ?packages:   <cnf-formula>
    ?requires:   [ STRING+ ]
    ?pp:         [ <ppflag>+ ]
    ?preinstalled: BOOL

<ppflag> := CAMLP4 { STRING+ }
          | STRING+
```

- `name` is the compiler name, it should be identical to the filename.

- `src` is the location where this version can be downloaded. It can be:
  - an archive available in the local filesystem
  - an archive available via `http` or `ftp`
  - a version-controlled repository under `svn` or `git` (with the expectation that these tools are installed on the user host).

- `patches` are optional patch addresses, available via `http`, `ftp` or locally on the filesystem.

- `configure` are the optional flags to pass to the configure script. The order is relevant: `-prefix=$opam/SWITCH/` will be automatically added at the end to these options. Remark that if these flags contain `-bindir`, `-libdir`, and `-mandir`, then every `-prefix` will be ignored by `configure`.

- **make** are the flags to pass to **make**. It must at least contain some target like **world** or **world.opt**.

- **bytecomp**, **asmcomp**, **bytelink** and **asmlink** are the compilation and linking flags to pass to the OCaml compiler. They will be taken into account by the **opam config** command (see §**??**).

- **packages** is the list of packages to install just after the compiler installation finished. These libraries will not consider what is in the **requires** nor **pp** (as **requires** and **pp** might want to use things already installed with **packages**).

- **requires** is a list of libraries and syntax extensions dependencies which will be added to every packages installed with this compiler. The libraries and syntax extensions should be present in packages defined in **packages**, otherwise an error should be thrown.

- **pp** is the command to use with the **-pp** command-line argument. It is either a full command line or a **camlp4** command, such as CAMLP4 ["pp-trace"]: this will look for the compilation flags for the syntax extension "pp-trace" and expand the **camlp4** command-line accordingly. All the syntax extensions used should be present in **packages**.

- **preinstall** is **true** when the version of the compiler available in the path is the same as **name**.

For instance the file, **3.12.1+memprof.comp** describes OCaml, version 3.12.1 with the memory profiling patch enabled:

```
opam-version: "1"
name:         "3.12.1"
src:          "http://caml.inria.fr/pub/distrib/ocaml-3.12/ocaml-3.12.1.tar.gz"
make:         [ "world" "world.opt" ]
patches:      [ "http://bozman.cagdas.free.fr/documents/ocamlmemprof-3.12.0.patch" ]
```

And the file **trunk-g-notk-byte.comp** describes OCaml from SVN trunk, with no *tk* support and only in bytecode, and all the libraries built with **-g**:

```
opam-version: "1"
name:         "trunk-g-notk-byte"
src:          "http://caml.inria.fr/pub/distrib/ocaml-3.12/ocaml-3.12.1.tar.gz"
configure:    [ "-no-tk" ]
make:         [ "world" ]
bytecomp:     [ "-g" ]
bytelink:     [ "-g" ]
```

3.3   Commands

3.3.1   Switching Compiler Version

If the user wants to switch to an other compiler version, he should run:

```
$ opam switch [-clone] [-alias $ALIAS] $SWITCH
```

This command will:

- If $ALIAS is not set, set it to $SWITCH

- Look for an existing $opam/$ALIAS directory.

  - If it exists, then change the `ocaml-version` content to $ALIAS in $opam/config.

  - If it does not exist, look for an existing $opam/compilers/SWITCH.comp. If the file does not exists, the command will fail with a well-defined error.

  - If the file exist, then build the new compiler with the right options (and pass `--prefix $opam/$ALIAS` to `./configure`) and initialize everything in `$opam/` in a consistent state as if "opam init" has just been called.

  - Update the file $opam/aliases with the line $ALIAS $SWITCH

- If the `-clone` option is set, the command will try to install the packages that were installed before switching (that are not currently installed). In case the new version contains installed packages that were not installed before switching, it will try to keep them.

  In short, the heuristic is to install the maximum of previous packages and remove the minimum. The success depends on the compatibility of the existing packages with respect to this new $SWITCH.

4   Managing Configurations

4.1   State

4.2   Files

4.2.1   Substitution files: `*.in`

Any file can be processed using generated using a special mode of `opam` which can perform tests and substitutes variables (see §4 for the exact command to run). Substitution files contains some templates which will be replaced by some contents. The syntax of templates is the following:

- templates such as `%{$NAME:$VAR}%` are replaced by the value of the variable `$VAR` defined at the root of the file `$config/NAME.config`.

- templates such as `%{$NAME.$LIB:$VAR}%` are replaced by the value of the variable `$VAR` defined in the `$LIB` section in the file `$config/PACKAGE.config`

4.2.2   Package configuration files: `*.config`

`$opam/SWITCH/config/NAME.config` follows the syntax defined in §1.2.1, with the following restrictions:

```
<file>    :=
    opam-version: "1"
    <item>*
<item>    := <def> | <section>
<section> :=
    <kind> STRING {
      ?asmcomp:  [ STRING+ ]
      ?bytecomp: [ STRING+ ]
      ?asmlink : [ STRING+ ]
      ?bytelink: [ STRING+ ]
```

```
      ?requires: [ STRING+ ]
      <def>*
    }
<kind>    := library | syntax
<def>     := IDENT: BOOL
           | IDENT: STRING
           | IDENT: [ STRING+ ]
```

`$NAME.config` contains platform-dependent information which can be useful for other libraries or syntax extensions that want to use libraries defined in the package `$NAME`.

Local and global variables   The definitions "`IDENT: BOOL`", "`IDENT: STRING`" and "`IDENT: [ STRING+ ]`", are used to defined variables associated to this package, and are used to substitute variables in template files (see §**??**):

- `%{$NAME:$VAR}%` will refer to the variable `$VAR` defined at the root of the configuration file `$config/NAME.config`.

- `%{$NAME.$LIB:$VAR}%` will refer to the variable `$VAR` defined in the `library` or `syntax` section named `$LIB` in the configuration file `$config/$NAME.config`.

Library and syntax sections   Each `library` and `syntax` section defines an OCaml library and the specific compilation flags to enable when using and linking with this library.

The distinction between libraries and syntax extensions is only useful at compile time to know whether the options should be used as compilation or pre-processing arguments (ie. should they go on the compiler command line or should they be passed to the `-pp` option). This is the responsibility of the build tool to do the right thing and the `<kind>` of sections is only used for documentation purposes in OPAM.

The available options are:

- `asmcomp` are compilation options to give to the native compiler (when using the `-c` option)

- `bytecomp` are compilation options to give to the bytecode compiler (when using the `-c` option)

- `asmlink` are linking options to give to the native compiler

- `bytlink` are linking options to give to the bytecode compiler

- `requires` is the list of libraries and syntax extensions the current block is depending on. The full list of compilation and linking options is built by looking at the transitive closure of dependencies. The contents of `deps` is the list of libraries or syntax extension the current section depends on. Note that we do not refer here to any package name, as multiple packages can expose libraries with the same name and interface and thus we want the user to be able to switch between them easily.

4.3   Commands

4.3.1   Getting Package Configuration

The first version of OPAM contains the minimal information to be able to use installed libraries. In order to do so, the end-user (or the packager) should run:

```
$ opam config list
$ opam config var $NAME:$VAR
$ opam config var $NAME.$LIB:$VAR
$ opam config subst $FILENAME+
$ opam config [-R] include   $NAME+
$ opam config [-R] bytecomp $NAME.$LIB+
$ opam config [-R] asmcomp  $NAME.$LIB+
$ opam config [-R] bytelink $NAME.$LIB+
$ opam config [-R] asmlink  $NAME.$LIB+
```

- `list` will return the list of all variables defined in installed packages (see §4.2.2)

- `var $var` will return the value associated to the variable `$var`

- `subst $FILENAME` replace any occurrence of `%{$NAME:$VAR}%` and `%{$NAME.$LIB:$VAR}%` as specified in §4.2.1 in `$FILENAME.in` to create `$FILENAME`.

- `includes $NAME` will return the list of paths to include when compiling a project using the package `$NAME` (`-R` gives a result taking into account the transitive closure of dependencies).

- `bytecomp`, `asmcomp`, `bytelink` and `asmlink` return the associated value for the section `$LIB` in the file `$config/$NAME.config` (`-R` gives a result taking into account the transitive closure of all dependencies).