OASIS
Architecture for building OCaml libraries and applications

Sylvain Le Gall <sylvain.le-gall@ocamlcore.com>
Presentation at OCaml Meeting 2010
April 16th, 2010
How it started

▪ Started in July 2008:
  ▪ Prototype made of code gathered from 3 other small projects
  ▪ I did a summary of problems that a Debian packager has to deal with when packaging an OCaml library (blog article)
  ▪ Mauricio Fernandez provided a small implementation of Cabal like system

▪ Since then:
  ▪ I added features when new projects needed it
  ▪ The project was renamed from OCamlAutobuild to OASIS

▪ Release 0.1.0 (2010/04/08)
Debian problems

- Non-native architectures
- Not using ocamlfind for libraries
- Custom build system
Requirements

- We need at least the following steps:
  - Configure: checks build environment, allows to disable/enable features
  - Build: creates libraries and executables
  - Install: moves results to the right place

- We can use
  - OCaml as a scripting language
  - Findlib to manage libraries
  - OCamlbuild, OMake, OcamlMakefile

- We should avoid
  - Shell scripts and Unix commands
  - Adding dependencies
  - Forcing projects to change things that work
  - Reinventing the wheel
Cabal is a system for building and packaging Haskell libraries and programs. It defines a common interface for package authors and distributors to easily build their applications in a portable way.

http://www.haskell.org/cabal/

- This is a building brick of Hackage (CPAN for Haskell)
- It makes really easy to use external libraries
- It is based on a single text file: pkg.cabal
- It is probably one of the reason of the Haskell's success
Copy Cabal file format

- Fields
- Sections
- Freeform
- Conditional

Simple text file

- Easy to read and write
- Beginners can understand it

OASIS Format: 0.1
Name: with-c
Version: 0.0.1
Authors: Sylvain Le Gall
License File: LICENSE
License: LGPL with OCaml linking exception
Synopsis: Minimal project with C file.
Plugins: META

Library "with-c"
Path: src
Modules: A
CSources: A_stub.c

Executable "test-with-c"
Path: src
Main Is: main.ml
Compiled Object: byte
Build Depends: with-c
CSources: main_stub.c
It compiles “_oasis” into a build system:
- “setup.ml” is the entry point
- It uses plugins to compile sub systems

External commands when stdlib is not enough:
- ocamlfind
- ocamlc -config
- cp, rm (Sys.os_type dependent)

External libraries only at compile time
How it works

Beginners customize...

_oasis

Advanced users customize...

setup.ml

Configure

Build

Test

Install

setup.data (txt)

setup.log (txt)

src/libfoo.clib

src/foo.mllib

OASIS
Is it working well?

- It scales well from small libraries to projects with several libraries and executables
- Easy to extend through plugins
- It still needs to create a huge “setup.ml” (~120kB)
Back to Debian problems

- Non-native architectures
- Not using ocamlfind for libraries
- Custom build system

- Automatically provided by OASIS
- Plugin META
- If widely adopted
Future projects

- **oasis-selfcontained:**
  - to create .tar.gz containing everything required to build

- **oasis-checkout:**
  - to checkout VCS of a package or a particular version

- **bocage.ocamlcore.org which should enable to:**
  - Upload “_oasis” file
  - Translate it to web pages
  - Translate it to GODI files
Conclusion

- Still a lot of work to do (OMake, OcamlMakefile)
- It creates a standard and portable full build system
- Creating Debian packages is easier
- It is a building brick for an Hackage in OCaml for OCaml
Demonstration
Questions ?
Extras
What are plugins?

- It translates an OASIS package data structure
- There are four kinds:
  - Conf
  - Build
  - Test
  - Doc
  - Install
  - Extra
- It can create extra fields in “_oasis”
  - “XCustomClean: $make clean”
- It can embed code into “setup.ml”
Some plugins

- None (conf, build, doc, test, install)
  - It does nothing and fail

- Custom (conf, build, doc, test, install)
  - It calls a shell command

- OCamlbuild (build)
  - It generates .mllib
  - It calls ocamlbuild with the right targets (e.g. "ocamlbuild test.cma" or "ocamlbuild test.cma test.cmxa")

- OcamlbuildDoc (doc)
  - It generates .odocl

- InternallInstall (install)
  - It installs what has been built using ocamlfind or cp

- META (extra)
  - It creates META files including build dependencies
Generated files

Generated content

Custom content

Allow to check generated content changes

Delimit insertion area for generated content

Generated content

Custom content

# OASIS_START
# DO NOT EDIT (digest: 1478ef...b2e38)
# Library odn
# Library pa_noodn
# Executable test
<tests/test.byte>: use_odn
<tests/test.byte>: pkg_str
<tests/test.byte>: pkg_oUnit
<tests/test.byte>: pkg_fileutils
<tests/*.ml>: use_odn
<tests/*.ml>: pkg_str
<tests/*.ml>: pkg_oUnit
<tests/*.ml>: pkg_fileutils
# Library pa_odn
"src": include
<src/*.ml>: pkg_type-conv
<src/*.ml>: pkg_camlp4.quotations.o
<src/*.ml>: pkg_camlp4.lib
# OASIS_STOP
<src/pa_odn.ml>: syntax_camlp4o
<src/pa_noodn.ml>: syntax_camlp4o

Delimit insertion area for generated content

Allow to check generated content changes

Custom content
The general form is “$var”

It can be recursive:

- $docdir
  - $datarootdir/doc/$pkg_name
  - $prefix/share/doc/ocamlify
  - /usr/local/share/doc/ocamlify

You can use functions to transform it:

- utoh: Unix to host for filename
- ocaml_escaped: String.escaped

Origin:

- Default value
- From file “setup.data” (static after configure step)
- From file “setup.log” (change each time you build something)
- From command line
- Environment
Main goal: Hackage/CPAN for OCaml

Should integrate with forge.ocamlcore.org:
- User accounts and login done through the forge
- When you upload an OASIS enabled package to the forge, it is automatically published into bocage.o.o
- Documentation will be shared with the document section of the forge
- If home web page is not set, redirect to the bocage web page of the package

Tarball won't be stored:
- Link to upstream website (to centralize download count)
- Backup to another website (archives.ocamlcore.org?)

Information about VCS

Should integrate 2 alternate GODI repositories
- Stable: no build problems (how to decide stable -> unstable migration)
- Unstable: everything published
fileutils.cma

findlib.cma

buildSys.ml

Dynamic loading is not portable

Too much customization in OCaml code, which is not easy for beginners.

Users customize build.ml directly

This first version works for small projects but doesn't scale