OCaml Batteries Included: An Open-Source Extended Standard Library for OCaml

François Bérenger
Yamanishi lab., Kyushu Institute of Technology, Japan

20/01/2021

Outline

1. An OCaml Crash Course
   - The Language
   - Type Signatures
   - Programming Environment
2. The Stdlib and Extended Standard Libraries
   - Stdlib
   - Batteries
   - Jane Street Core
   - Containers
3. Conclusion

Outline

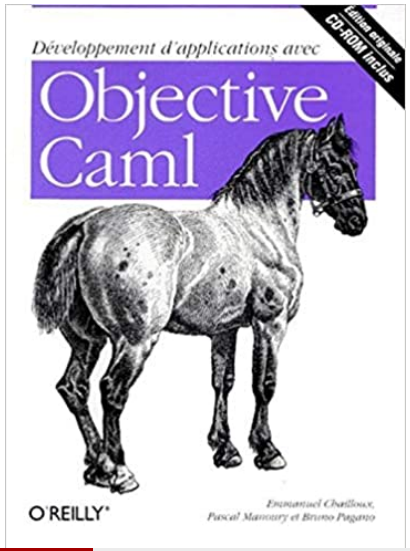
1. An OCaml Crash Course
   - The Language
   - Type Signatures
   - Programming Environment
2. The Stdlib and Extended Standard Libraries
   - Stdlib
   - Batteries
   - Jane Street Core
   - Containers
3. Conclusion

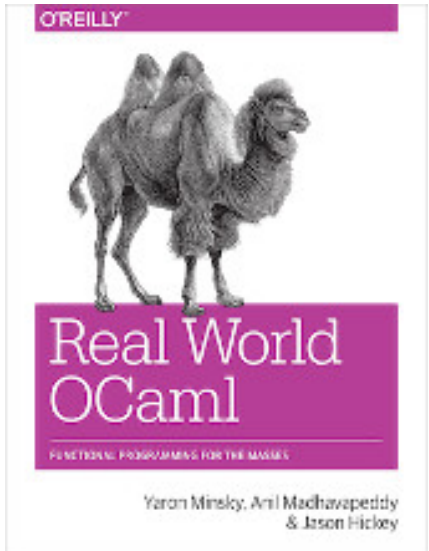
What is OCaml?

- OCaml was born in France at INRIA (now Inria).
- Created by Xavier Leroy *et al.* in 1996.
- OCaml is *strongly-typed* and uses type inference.
- Encourages the functional programming style.
- Is pragmatic: imperative programming is also supported.
- Also supports object-oriented programming.

Where to learn OCaml?

- The online manual
  `https://caml.inria.fr/pub/docs/manual-ocaml/`.
- Chapters: 1 “The core language”, 2 “The module system”, 4 “Labels and variants”, stdlib’s Pervasives and List modules.
- An old book: “Developing Applications With Objective Caml”
  `https://caml.inria.fr/pub/docs/oreilly-book/html/`.
- A more recent book: “Real World OCaml”
  `https://dev.realworldocaml.org/index.html`.





Understanding OCaml type signatures: basic ones

```
(* identity: maybe a complete specification; rare case *)
val identity: 'a -> 'a
val apply: 'a -> 'b
val accumulate: 'acc -> 'b -> 'acc
(* side_effect: 'unit' in type signatures is '()'
   in implementation code *)
val side_effect: 'a -> unit
val predicate: 'a -> bool
(* compare: int is not very precise *)
val compare: 'a -> 'a -> int
```

Understanding OCaml type signatures: more complex ones

```
val cons: 'a -> 'a list -> 'a list
(* ``head''; imprecise signature (empty list) *)
val hd: 'a list -> 'a
val length: 'a list -> int
val map: ('a -> 'b) -> 'a list -> 'b list
val fold: ('acc -> 'b -> 'acc) -> 'acc -> 'b list -> 'acc
val iter: ('a -> unit) -> 'a list ->()
val exists: ('a -> bool) -> 'a list -> bool
val filter: ('a -> bool) -> 'a list -> 'a list
val partition: ('a -> bool) -> 'a list -> ('a list * 'a list)
(* purely functional *)
val sort: ('a -> 'a -> int) -> 'a list -> 'a list
(* imperative *)
val array_sort: ('a -> 'a -> int) -> 'a array -> unit
```

An OCaml programming environment

Suggested tools to get started with OCaml:

- opam: source-based package manager (Debian/Ubuntu: apt?).
- utop: cool top-level interpreter.
- merlin: editor helper.
- Emacs w/ tuareg mode (or Vim or VS Code or Atom).
- dune: build system.
- ocp-browser: command-line documentation browser.
- ocp-indent: integrates with Emacs.
- user-setup: automatic editor configuration.

Outline

1. An OCaml Crash Course
   - The Language
   - Type Signatures
   - Programming Environment
2. The Stdlib and Extended Standard Libraries
   - Stdlib
   - Batteries
   - Jane Street Core
   - Containers
3. Conclusion

The "Standard Library": Stdlib

`https://caml.inria.fr/pub/docs/manual-ocaml/libref/index.html`

- The library that comes with the compiler.
- A Spartan set of features.
- I do not recommend the stdlib for programming in the large.
- Some non tail-recursive functions !
- Very hard and time-consuming to get something accepted into the stdlib...

Batteries: a Community-Maintained Extended Standard Library

- `https://ocaml-batteries-team.github.io/batteries-included/hdoc2/`
- A drop-in replacement for the stdlib.
- 100% compatible, (almost) always tail-recursive.
- Makes new stdlib functions available to older OCaml versions.
- Long history (git logs from 2008), still maintained by several people (Cedric Cellier[1], Gabriel Scherer[1], me).
- New contributors: Florent Monnier, Jakob Krainz.
- 86 contributors in total (github).
- 68 opam packages depend on batteries (as of 20/01/2020).
- Downloaded 2719 times last month.

[1] Long-term contributor

How to use batteries

```
(* in the top-level *)
#use ''batteries'';;
(* I recommend *)
module L = BatList
module A = BatArray
module S = BatString
module SS = BatSet.String
L.map (* ... *)
(* rather than *)
open Batteries (* would shadow some Stdlib things;
                  e.g. the Printf module *)
```

A few batteries modules

I recommend:

- BatArray (there is also BatDynArray).
- BatHashtbl (many functionalities).
- BatInt (e.g. BatInt.Set, BatInt.compare).
- BatList (many functionalities).
- BatMap (a kind of Hashtbl but with ordered keys).
- BatSeq (instead of BatEnum).
- BatSet (many operations).
- BatStream (rather than BatLazyList or BatEnum).
- BatString (e.g. BatString.Set; many operations).

I do not recommend:

- BatEnum (because there is BatSeq now; might be removed).
- BatIO (because slower than the stdlib in_channel/out_channel; might be removed).
- BatLazyList (because of BatSeq).

Batteries have many more modules... I don't know/use all of them.

How to contribute to batteries?

- Open an issue on github.
- Interact with us to check we are interested.
- Send your code as a pull request.
- The code must be (in that order):
  1. Correct
  2. Efficient
  3. Clean / readable / maintainable
  4. Commented
  5. Unit-tested
- Try to satisfy all maintainers (if possible).
- Last but not least: be tenacious, follow through.

Advantages of batteries

- A single OPAM package.
- Takes 47s to install on a 24 threads Mac.
- Actively maintained.
- Stable set of features and API / mature code base.
- Open-source with an open development model.
- Well written documentation; for almost all functions.
- Quite clean, sometimes beautiful, source code.
- Significant code-coverage by unit tests (for modern code).
- Drop-in replacement for the stdlib.
- Feature-parity between BatMap and BatSet.
- Feature-parity between BatArray and BatList.

Disadvantages of batteries

- More maintained than actively developed.
- Still using ocamlbuild (instead of dune).
- Cannot install and compile a single module from opam (e.g. only BatList and dependencies; maybe in the future...).
- ...the audience is welcome to point out some more ...

What is Jane Street Core?

"Industrial strength alternative to OCaml's standard library"

`https://ocaml.janestreet.com/ocaml-core/latest/doc/core/Core/index.html`

- 108 packages in opam depend on core (as of 20/01/2021).
- Installed 9318 times last month.
- 15 contributors on github.

Advantages of Jane Street Core

Warning: I have not used core since 2013 !

- Actively maintained.
- Backed by a wealthy company: Jane Street[2].
- Used in production by mission-critical software.
- Stable set of features and API.
- Mature code base.
- Naming conventions.

[2] Biggest OCaml industrial user; long-term benevolent contributor to the ecosystem.

Disadvantages of Jane Street Core

- Documentation is hard to find; not all functions are documented in natural language.
- Development model: open-source behind closed walls ?
- Not compatible with the stdlib (if you were using the stdlib and switch to core...).
- Mandatory use of labels.
- Heavy use of the type system (not everybody writes soft-real-time, mission-critical server software...).
- An OPAM package with 47 dependencies !
- Which library should I use: Core? Core_kernel? Base?
- Took 240s to install on a 24 threads Mac (batteries: 47s).

What is Containers?

"A modular, clean and powerful extension of the OCaml standard library".

`https://c-cube.github.io/ocaml-containers/last/containers/index.html`.

- Created by Simon Cruanes (OCaml hacker and former batteries contributor).
- 47 packages in opam depend on containers (as of 20/01/2021).
- Installed 1542 times last month.
- 44 contributors on github.

Pros and cons of Containers

Warning: I have never used containers in a project (yet?) !

- Each module is independent (you can copy a file into your project; BSD license).
- A single OPAM package.
- Takes 14s to install on a 24 threads Mac (batteries: 47s).
- Open-source with an open development model.
- Pure OCaml: no dependencies to unix, str or num libraries.
- +/- Minimalist.
- . . . people who know more may add something here . . .

Outline

1. An OCaml Crash Course
   - The Language
   - Type Signatures
   - Programming Environment
2. The Stdlib and Extended Standard Libraries
   - Stdlib
   - Batteries
   - Jane Street Core
   - Containers
3. Conclusion

The opinionated slide

- Batteries is a successful open-source project (given its longevity, number of contributors, users, etc.).
- The OCaml ecosystem (and the language) are still evolving. Since OPAM came out; things are moving faster.
- Almost all the community is on github.
- When programming in Python; one mostly spends his time *debugging*. When programming in OCaml, one spends his time *developing* software.
- Objects in OCaml are like an electric guitar in the middle of a symphonic orchestra. If, and only if, your name is Ennio Morricone should you play the guitar [3].

[3] The cryptokit library only has an object interface... ☹

Bibliography

- Berenger, F., Zhang, K. Y., & Yamanishi, Y. (2019). Chemoinformatics and structural bioinformatics in OCaml. Journal of cheminformatics, 11(1), 1-13.
- E. Chailloux, P. Manoury, B. Pagano. Developing Applications With Objective Caml. O'reilly (2000).
- Leroy, X., Doligez, D., Frisch, A., Garrigue, J., Rémy, D., & Vouillon, J. (2019). The OCaml system release 4.11.
- Minsky, Y., Madhavapeddy, A., & Hickey, J. (2013). Real World OCaml: Functional programming for the masses. "O'Reilly Media, Inc.".

The End

All questions are welcome.

Thank your for your attention.

Do *not* hesitate to contribute to batteries!

`https://github.com/ocaml-batteries-team/batteries-included`

Thanks to all the OCaml community!