



# Preface

*This is a book about hacking in ocaml. It's assumed that you already understand the underlying theory. Happy hacking Most parts are filled with code blocks, I will add some comments in the future. Still a book in progress. Don't distribute it.*

☺



# Acknowledgements

---

write  
later



# Contents

<b>Preface</b>	<b>3</b>
<b>Acknowledgements</b>	<b>5</b>
<b>1 Tool Chain</b>	<b>15</b>
1.1 Ocamlbuild . . . . .	16
multiple directories . . . . .	19
grouping targets . . . . .	20
With lex yacc, ocamlfind . . . . .	21
1.1.1 Principles . . . . .	22
1.1.2 Write plugin . . . . .	23
Samples . . . . .	26
Mixing with C stubs . . . . .	28
1.1.3 OCamlbuild in the toplevel . . . . .	34
1.1.4 Building interface files . . . . .	35
Interaction with Git . . . . .	35
1.2 Godi,otags . . . . .	36
1.2.1 CheatSheet . . . . .	36
1.3 Ocamlfind . . . . .	34
1.4 toplevel . . . . .	35
1.4.1 directives . . . . .	35
1.4.2 Module toploop . . . . .	35
1.4.3 Env . . . . .	36

1.5	Ocaml doc . . . . .	38
1.6	ocamlmktop . . . . .	41
1.7	Standard OCaml Develop Tools . . . . .	42
1.8	ocamlmktop . . . . .	44
1.8	Git . . . . .	46
<b>2</b>	<b>Lexing</b>	<b>47</b>
2.1	Lexing . . . . .	48
2.1.1	Ulex interface . . . . .	53
2.2	Ocamlllex . . . . .	61
<b>3</b>	<b>Parsing</b>	<b>65</b>
3.1	Ocamlyacc . . . . .	66
3.2	MENHIR Related . . . . .	80
<b>4</b>	<b>Camlp4</b>	<b>83</b>
4.1	Brief Intro To Parser . . . . .	84
4.2	Basics Structure . . . . .	85
4.2.1	Experimentation Environment . . . . .	85
4.2.2	Command Options . . . . .	88
4.2.3	Module Components . . . . .	89
4.2.4	Simple Experiment . . . . .	90
	Example Options Pa_abstract . . . . .	91
4.3	SourceCode Exploration . . . . .	92
4.3.1	Camlp4 PreCast . . . . .	92
4.3.2	OCamlInitSyntax . . . . .	93
4.3.3	Camlp4.Sig . . . . .	98
4.3.4	Camlp4.Struct.Camlp4Ast.ml last . . . . .	99
4.3.5	AstFilters . . . . .	99
4.3.6	Camlp4.Register . . . . .	100
4.3.7	Camlp4Ast . . . . .	104
4.3.8	TestFile . . . . .	133

4.4	Extensible Parser . . . . .	135
4.4.1	Examples . . . . .	135
4.4.2	Mechanism . . . . .	136
4.4.3	Parsing OCaml using Camlp4 . . . . .	139
	Fully Utilize Camlp4 Parser and Printers . . . . .	139
	Otags Mini . . . . .	141
	Parsing Json AST . . . . .	142
4.5	STREAM PARSER . . . . .	146
4.6	Grammar . . . . .	147
	Example: Expr Parse Tree . . . . .	152
4.7	QuasiQuotations . . . . .	157
	Quotation Expander . . . . .	159
	Lambda Example . . . . .	163
4.7.1	Ast Transformation . . . . .	166
4.7.2	Part8, 9 Quotation . . . . .	169
4.7.3	Part 10 Lexer . . . . .	176
4.8	Revised syntax . . . . .	180
4.9	Filters in camlp4 . . . . .	186
4.9.1	Map Filter . . . . .	186
4.9.2	Filter Examples . . . . .	186
	Example: Map Filter . . . . .	186
	Linking Problem . . . . .	188
	Example: Add Zero . . . . .	189
	Fold filter . . . . .	189
	Meta filter . . . . .	189
	Lift filter . . . . .	191
	Macro Filter . . . . .	191
4.9.3	Example Tuple Map . . . . .	192
4.9.4	Location Strip filter . . . . .	192
4.9.5	Camlp4Profiler . . . . .	193
4.9.6	Camlp4TrashRemover . . . . .	193



4.9.7	Camlp4ExceptionTracer . . . . .	193
4.10	Examples . . . . .	194
4.10.1	Pa_python . . . . .	194
4.10.2	Pa_list . . . . .	199
4.10.3	Pa_abstract . . . . .	201
4.10.4	Pa_apply . . . . .	202
4.10.5	Pa_ctyp . . . . .	202
4.10.6	Pa_exception_wrapper . . . . .	203
4.10.7	Pa_exception_tracer . . . . .	206
4.10.8	Pa_freevars . . . . .	207
4.10.9	Pa_freevars_filter . . . . .	207
4.10.10	Pa_global_handler . . . . .	207
4.10.11	Pa_holes . . . . .	207
4.10.12	Pa_minimm . . . . .	207
4.10.13	Pa_plus . . . . .	207
4.10.14	Pa_zero . . . . .	207
4.10.15	Pa_printer . . . . .	207
4.10.16	Parse_arith . . . . .	208
4.10.17	Pa_estring . . . . .	208
4.10.18	Pa_holes . . . . .	217
4.11	Useful links . . . . .	218
<b>5</b>	<b>Libraries</b>	<b>219</b>
5.1	batteries . . . . .	220
	syntax extension . . . . .	220
5.1.1	Dev . . . . .	220
5.1.2	BOLT . . . . .	221
5.2	Mikmatch . . . . .	222
5.3	pa-do . . . . .	235
5.4	num . . . . .	236
5.5	caml-inspect . . . . .	237

5.6	ocamlgraph . . . . .	242
5.7	pa-monad . . . . .	251
5.8	bigarray . . . . .	255
5.9	sexplib . . . . .	256
5.10	bin-prot . . . . .	259
5.11	fieldslib . . . . .	260
5.12	variantslib . . . . .	261
5.13	delimited continuations . . . . .	262
5.14	shcaml . . . . .	268
5.15	deriving . . . . .	269
5.16	Modules . . . . .	270
<b>6</b>	<b>Runtime</b>	<b>273</b>
6.1	ocamlrun . . . . .	274
6.2	FFI . . . . .	276
6.2.1	Data representation . . . . .	277
6.2.2	Caveats . . . . .	287
<b>7</b>	<b>GC</b>	<b>289</b>
<b>8</b>	<b>Object-oriented</b>	<b>297</b>
8.1	Simple Object Concepts . . . . .	298
8.2	Modules vs Objects . . . . .	302
8.3	More about class . . . . .	303
<b>9</b>	<b>Language Features</b>	<b>305</b>
9.1	Stream Expression . . . . .	306
9.2	GADT . . . . .	310
9.3	First Class Module . . . . .	311
9.4	Pahantom Types . . . . .	315
9.4.1	Useful links . . . . .	322
9.5	Positive types . . . . .	323

9.6	Private Types . . . . .	324
9.7	Subtyping . . . . .	326
9.8	Explicit Nameing Of Type Variables . . . . .	327
9.9	The module Language . . . . .	328
<b>10</b>	<b>subtle bugs</b>	<b>329</b>
10.1	Reload duplicate modules . . . . .	330
10.2	debug . . . . .	332
10.3	Debug Cheat Sheet . . . . .	333
<b>11</b>	<b>Interoperating With C</b>	<b>335</b>
<b>12</b>	<b>Pearls</b>	<b>337</b>
12.1	Write Printf-Like Function With Ksprintf . . . . .	338
12.2	Optimization . . . . .	338
12.3	Weak Hashtbl . . . . .	338
12.4	Bitmatch . . . . .	338
12.5	Interesting Notes . . . . .	339
12.6	Polymorphic Variant . . . . .	340
<b>13</b>	<b>XX</b>	<b>345</b>
13.0.1	tricks . . . . .	346
13.0.2	ocaml blogs . . . . .	350
<b>14</b>	<b>Topics</b>	<b>351</b>
14.1	First Order Unification . . . . .	352
14.1.1	First-order terms . . . . .	352
14.1.2	Substitution . . . . .	353
14.1.3	Unification in Various areas . . . . .	353
14.1.4	Occurs check . . . . .	354
14.1.5	Unification Examples . . . . .	354
14.1.6	Algorithm . . . . .	354
14.2	LLVM . . . . .	358

# Todo list

write later . . . . .	5
mlpack file . . . . .	18
Glob Patterns . . . . .	21
parser-help to coordinate menhir and ulex . . . . .	60
theoretic stuff . . . . .	84
Should be re-written later . . . . .	289
Write later . . . . .	304
read ml 2011 workshop paper . . . . .	310
Read the slides by Jacques Garrigue . . . . .	311
write later with subtyping . . . . .	323
write later . . . . .	328
polymorphic comparison . . . . .	330
Write later . . . . .	335



# Chapter 1

## Tool Chain

## 1.7 Standard OCaml Develop Tools

ocaml	toplevel top
ocamlrun	bytecode interpreter
ocamlc	bytecode batch compiler
ocamlopt	native code batch compiler
ocamlc.opt	<i>optimized</i> bytecode batch compiler
ocamlopt.opt	<i>optimized</i> native code batch compiler
ocamlmktop	new <i>toplevel</i> constructor

Table 1.1: Ocaml Compiler Tools

The optimized compilers are themselves compiled with the Objective Caml native compiler. They compile *faster* but are otherwise *identical* to their unoptimized counterparts.

**Compilation Unit** For the interactive system, the unit of compilation corresponds to a phrase of the language. For the batch compiler, the unit of compilation is two files: the source file, and the interface file. The *compiled interface* is used for both the bytecode and native code compiler.

extension	meaning
.ml	source
.mli	interface
.cmi	compiled interface
.cmo	object file (byte)
.cma	library object file(bytecode)
.cmx	object file (native)
.cmxa	library object file(native)
.c	c source
.o	c object file (native)
.a	c library object file (native)

Table 1.2: ocaml file name extension

<b>-a</b>	construct a runtime library
<b>-annot</b>	save information in <filename>.annot
<b>-c</b>	compile <i>without</i> linking
<b>-o name_of_executable</b>	specify the name of the executable
<b>-linkall</b>	link with <i>all</i> libraries used
<b>-i</b>	<i>display all</i> compiled global declarations, generate .mli file
<b>-pp</b>	preprocessor
<b>-unsafe</b>	<i>turn off</i> index checking
<b>-v</b>	display version
<b>-w list</b>	choose among the list the level of warning message
<b>-impl file</b>	indicate that <i>file</i> is a caml source(.ml)
<b>-intf file</b>	as a caml interface(.mli)
<b>-I dir</b>	add directory in the list of directories
<b>-thread</b>	light process
<b>-g, -noassert</b>	linking with debug information
<b>-custom, -cclib, -ccopt, -cc</b>	standalone executable
<b>-make-runtime, -use-runtime</b>	runtime
<b>-output-obj</b>	output a c object file instead of an executable
<b>-vmthread</b>	VM-level thread support
<b>-dparsetree</b>	generate the parse output
<b>-drawlambda</b>	s-expression
<b>-dlambda</b>	s-expression
<b>-dinstr</b>	generate asm

Table 1.3: ocamlc options

A/a	enable/disable all messages
F/f	partial application in a sequence
P/p	incomplete pattern matching
U/u	missing cases in pattern matching
X/x	enable/disable all other messages
M/m and V/v	for hidden object



Table 1.4: warning option

About warning messages (Table 1.4) , the compiler chooses the **(A)** by default. turn off some warnings sometimes is helpful, for example

```
1 ocamlbuild -cflags -w,aPF top_level.cma
```

-compact	optimize the produced code for space
-S	keeps the assembly code in a file
-inline level	set the aggressiveness of inlining
-S	keep assembly output

Table 1.5: ocamlpt option

-I dir	adds the directory
-unsafe	no bounds checking

Table 1.6: toplevel option

## 1.8 ocamlmktop

OCAMLMKTOP( Table 1.6) is ofen used for *pulling native object* code libraries(typically written in C) into a new toplevel. *-cclib libname, -ccopt optioin, -custom, -I dir -o exectuable*

For example:

```
1 ocamlmktop -custom -o mytoplevel graphics.cma \
2 -cclib -I/usr/X11/lib -cclib -lX11
```

This *standalone* exe(-custom) will be *linked* to the library X11(libX11.a) which in turn will be looked up in the path */usr/X11/lib*

A standalone exe is a program that *does not* depend on OCaml installation to run. The OCaml *native* compiler produces standalone executabulars by default. But *without* -custom option, the *bytecode* compiler produces an executabular which requires the *bytecode interpreter ocamlrun*

---

```

1 ocamlc test.ml -o a
2 ocamlc -custom test.ml -o b
3 -rwxr-xr-x  1 bob  staff   12225 Dec 23 16:31 a
4 -rwxr-xr-x  1 bob  staff  198804 Dec 23 16:31 b

```

---

You can see the size of `b` with the `ocamlrun` is way more bigger than `a`

---

```

1 bash-3.2$ cat a | head -n 1
2 #!/Users/bob/SourceCode/ML/godi/bin/ocamlrun

```

---

Without `-custom`, it depends on `ocamlrun`. With `-custom`, it contains the *Zinc* interpreter as well as the program bytecode, this file can be executed directly or copied to another machine (using the same CPU/Operating System). Still, the inclusion of machine code means that stand-alone executables are not portable to other systems or other architectures.

**Optimization** It is necessary to not create *intermediate closures* in the case of application on several arguments. For example, when the function `add` is applied with two integers, it is not useful to create the first closure corresponding to the function of applying `add` to the first argument. It is necessary to note that the creation of a closure would *allocate* certain memory space for the environment and would require the recovery of that memory space in the future. *Automatic memory recovery* is the second major performance concern, along with environment.

## Chapter 2

### Lexing



# Chapter 3

## Parsing

# Chapter 4

## Camlp4

Camlp4 stands for Preprocess-Pretty-Printer for `OCaml`, it's extremely powerful and hard to grasp as well. It is a source-to-source level translation tool.

# Chapter 5

## Libraries

## 5.16 Modules



# Chapter 6

## Runtime



# Chapter 7

## GC

---

Should  
be re-  
written  
later



## Chapter 8

# Object-oriented

Write  
later



## Chapter 9

# Language Features

## 9.9 The module Language

An orange rectangular box with rounded corners. Inside the box, the words "write" and "later" are stacked vertically in a black, sans-serif font. A small orange L-shaped bracket is positioned to the right of the box, pointing towards the right margin.



# Chapter 10

## subtle bugs



# Chapter 11

## Interoperating With C

---

Write  
later



# Chapter 12

## Pearls



# Chapter 13

XX

# Chapter 14

## Topics