Developing Applications With

Objective Caml
The original edition of this book (ISBN 2-84177-121-0) was published in France by O’REILLY & Associates under the title Développement d’applications avec Objective Caml.

Historique :
• Version 19990324??????????

© O’REILLY & ASSOCIATES, 2000

Cover concept by Ellie Volckhausen.

Édition : Xavier CAZIN.

Les programmes figurant dans ce livre ont pour but d’illustrer les sujets traités. Il n’est donné aucune garantie quant à leur fonctionnement une fois compilés, assemblés ou interprétés dans le cadre d’une utilisation professionnelle ou commerciale.

© ÉDITIONS O’REILLY, Paris, 2000
ISBN

Toute représentation ou reproduction, intégrale ou partielle, faite sans le consentement de l’auteur, de ses ayants droit, ou ayants cause, est illicite (loi du 11 mars 1957, alinéa 1er de l’article 40). Cette représentation ou reproduction, par quelque procédé que ce soit, constituerait une contrefaçon sanctionnée par les articles 425 et suivants du Code pénal. La loi du 11 mars 1957 autorise uniquement, aux termes des alinéas 2 et 3 de l’article 41, les copies ou reproductions strictement réservées à l’usage privé du copiste et non destinées à une utilisation collective d’une part et, d’autre part, les analyses et les courtes citations dans un but d’exemple et d’illustration.
Preface

The desire to write a book on Objective Caml sprang from the authors’ pedagogical experience in teaching programming concepts through the Objective Caml language. The students in various majors and the engineers in continuing education at Pierre and Marie Curie University have, through their dynamism and their critiques, caused our presentation of the Objective Caml language to evolve greatly. Several examples in this book are directly inspired by their projects.

The implementation of the Caml language has been ongoing for fifteen years. Its development comes from the Formel and then Cristal projects at INRIA, in collaboration with Denis Diderot University and the École Normale Supérieure. The continuous efforts of the researchers on these teams, as much to develop the theoretical underpinnings as the implementation itself, have produced over the span of years a language of very high quality. They have been able to keep pace with the constant evolution of the field while integrating new programming paradigms into a formal framework. We hope through this exposition to contribute to the widespread diffusion which this work deserves.

The form and the foundation of this book wouldn’t be what they are without the help of numerous colleagues. They were not put off by rereading our first manuscripts. Their remarks and their comments have allowed this exposition to improve throughout the course of its development. We wish particularly to thank Maria-Virginia Aponte, Sylvain Baro, Christian Codognet, Hélène Cottier, Guy Cousineau, Pierre Crégut, Titou Durand, Christophe Gonzales, Michelle Morcrette, Christian Queinnec, Attila Raksany and Didier Rémy.

The HTML version of this book would not have seen the light of day without the tools hevea and VideoC. A big thank you to their respective authors, Luc Maranget and Christian Queinnec, who have always responded in the briefest intervals to our questions and our demands for changes.
Contents

Preface v

Table of contents vii

Introduction xxi

1: How to obtain Objective Caml 1

Description of the CD-ROM ............................. 1
Downloading ................................................. 2
Installation ................................................. 2
   Installation under Windows ......................... 2
   Installation under LINUX ......................... 4
   Installation under MacOS ......................... 4
   Installation from source under Unix ............. 5
   Installation of the HTML documentation .......... 5
Testing the installation ............................... 5

I Language Core 7

2: Functional programming 11

Functional core of Objective Caml ................... 12
   Primitive values, functions, and types ........... 12
   Conditional control structure ................... 18
Value declarations ............................................. 19  
Function expressions, functions .................................. 21  
Polymorphism and type constraints ................................. 28  
Examples ................................................................ 31  
Type declarations and pattern matching ............................ 34  
Pattern matching .......................................................... 34  
Type declaration .............................................................. 41  
Records .................................................................... 43  
Sum types .................................................................. 45  
Recursive types .............................................................. 47  
Parametrized types .......................................................... 48  
Scope of declarations ...................................................... 49  
Function types .............................................................. 49  
Example: representing trees ............................................ 50  
Recursive values which are not functions ......................... 52  
Typing, domain of definition, and exceptions ..................... 54  
Partial functions and exceptions ..................................... 54  
Definition of an exception .............................................. 55  
Raising an exception ...................................................... 56  
Exception handling ....................................................... 56  
Polymorphism and return values of functions ..................... 58  
Desktop Calculator ....................................................... 59  
Exercises .................................................................. 62  
Merging two lists ............................................................ 62  
Lexical trees ................................................................. 63  
Graph traversal .............................................................. 64  
Summary .................................................................. 64  
To learn more .................................................................. 64  

3: Imperative Programming ................................. 67  
Modifiable Data Structures ............................................. 68  
Vectors .................................................................. 68  
Character Strings .......................................................... 72  
Mutable Fields of Records ............................................ 73  
References .................................................................. 74  
Polymorphism and Modifiable Values ............................... 74  
Input-Output ................................................................. 76  
Channels .................................................................. 77  
Reading and Writing ..................................................... 77  
Example: Higher/Lower ................................................ 78  
Control Structures .......................................................... 79  
Sequence .................................................................... 79  
Loops ........................................................................... 81  
Example: Implementing a Stack ...................................... 82  
Example: Calculations on Matrices ................................. 84  
Order of Evaluation of Arguments ................................. 85
4: Functional and Imperative Styles

Comparison between Functional and Imperative ........................................ 92
  The Functional Side ........................................................................... 93
  The Imperative Side ........................................................................... 93
  Recursive or Iterative ....................................................................... 95
Which Style to Choose? ........................................................................ 96
  Sequence or Composition of Functions .............................................. 97
  Shared or Copy Values ..................................................................... 99
  How to Choose your Style ................................................................ 101
Mixing Styles ...................................................................................... 103
  Closures and Side Effects ................................................................. 103
  Physical Modifications and Exceptions ............................................. 105
  Modifiable Functional Data Structures ............................................. 105
  Lazy Modifiable Data Structures ....................................................... 107
Streams of Data .................................................................................. 110
  Construction ..................................................................................... 110
  Destruction and Matching of Streams .............................................. 111
Exercises .............................................................................................. 114
  Binary Trees ...................................................................................... 114
  Spelling Corrector ........................................................................... 115
  Set of Prime Numbers ..................................................................... 115
Summary ............................................................................................... 115
To Learn More ..................................................................................... 116

5: The Graphics Interface

Using the Graphics Module ................................................................. 118
Basic notions ........................................................................................ 118
Graphical display ................................................................................ 119
  Reference point and graphical context ............................................. 119
  Colors ................................................................................................. 120
  Drawing and filling .......................................................................... 121
  Text .................................................................................................... 123
  Bitmaps ............................................................................................. 125
  Example: drawing of boxes with relief patterns ................................ 126
Animation ............................................................................................. 130
Events ................................................................................................... 132
  Types and functions for events ......................................................... 132
  Program skeleton .............................................................................. 133
<table>
<thead>
<tr>
<th>6: Applications</th>
<th>147</th>
</tr>
</thead>
<tbody>
<tr>
<td>Database queries</td>
<td>148</td>
</tr>
<tr>
<td>Data format</td>
<td>148</td>
</tr>
<tr>
<td>Reading a database from a file</td>
<td>150</td>
</tr>
<tr>
<td>General principles for database processing</td>
<td>151</td>
</tr>
<tr>
<td>Selection criteria</td>
<td>153</td>
</tr>
<tr>
<td>Processing and computation</td>
<td>156</td>
</tr>
<tr>
<td>An example</td>
<td>157</td>
</tr>
<tr>
<td>Further work</td>
<td>159</td>
</tr>
<tr>
<td>BASIC interpreter</td>
<td>159</td>
</tr>
<tr>
<td>Abstract syntax</td>
<td>160</td>
</tr>
<tr>
<td>Program pretty printing</td>
<td>162</td>
</tr>
<tr>
<td>Lexing</td>
<td>163</td>
</tr>
<tr>
<td>Parsing</td>
<td>165</td>
</tr>
<tr>
<td>Evaluation</td>
<td>169</td>
</tr>
<tr>
<td>Finishing touches</td>
<td>173</td>
</tr>
<tr>
<td>Further work</td>
<td>176</td>
</tr>
<tr>
<td>Minesweeper</td>
<td>176</td>
</tr>
<tr>
<td>The abstract mine field</td>
<td>177</td>
</tr>
<tr>
<td>Displaying the Minesweeper game</td>
<td>182</td>
</tr>
<tr>
<td>Interaction with the player</td>
<td>188</td>
</tr>
<tr>
<td>Exercises</td>
<td>192</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>II Development Tools</th>
<th>193</th>
</tr>
</thead>
<tbody>
<tr>
<td>7: Compilation and Portability</td>
<td>197</td>
</tr>
<tr>
<td>Steps of Compilation</td>
<td>198</td>
</tr>
<tr>
<td>The Objective Caml Compilers</td>
<td>198</td>
</tr>
<tr>
<td>Description of the Bytecode Compiler</td>
<td>199</td>
</tr>
<tr>
<td>Compilation</td>
<td>201</td>
</tr>
<tr>
<td>Command Names</td>
<td>201</td>
</tr>
<tr>
<td>Compilation Unit</td>
<td>201</td>
</tr>
<tr>
<td>Naming Rules for File Extensions</td>
<td>202</td>
</tr>
<tr>
<td>The Bytecode Compiler</td>
<td>202</td>
</tr>
<tr>
<td>Native Compiler</td>
<td>204</td>
</tr>
</tbody>
</table>
# Table of Contents

- Toplevel Loop ........................................... 205
- Construction of a New Interactive System ............... 206
- Standalone Executables .................................. 207
- Portability and Efficiency ................................ 208
- Standalone Files and Portability ......................... 208
- Efficiency of Execution ................................ 208
- Exercises .................................................. 209
- Creation of a Toplevel and Standalone Executable ....... 209
- Comparison of Performance ............................... 209
- Summary .................................................... 210
- To Learn More ............................................. 210

8: Libraries ................................................. 213

- Categorization and Use of the Libraries .................. 214
- Preloaded Library ......................................... 215
- Standard Library ......................................... 215
- Utilities .................................................... 216
- Linear Data Structures .................................. 217
- Input-output .............................................. 223
- Persistence ............................................... 228
- Interface with the System ................................ 234
- Other Libraries in the Distribution ....................... 239
- Exact Math ............................................... 239
- Dynamic Loading of Code ................................ 241
- Exercises .................................................. 244
- Resolution of Linear Systems .............................. 244
- Search for Prime Numbers ................................ 244
- Displaying Bitmaps ....................................... 245
- Summary .................................................... 246
- To Learn More ............................................. 246

9: Garbage Collection ....................................... 247

- Program Memory .......................................... 248
- Allocation and Deallocation of Memory .................... 249
- Explicit Allocation ....................................... 249
- Explicit Reclamation ..................................... 250
- Implicit Reclamation ..................................... 251
- Automatic Garbage Collection ............................. 252
- Reference Counting ...................................... 252
- Sweep Algorithms ........................................ 253
- Mark & Sweep ............................................. 254
- Stop & Copy ................................................ 256
- Other Garbage Collectors ................................ 259
- Memory Management by Objective Caml ................. 261
- Module Gc ................................................. 263
## 10: Program Analysis Tools

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dependency Analysis</td>
<td>272</td>
</tr>
<tr>
<td>Debugging Tools</td>
<td>273</td>
</tr>
<tr>
<td>Trace</td>
<td>273</td>
</tr>
<tr>
<td>Debug</td>
<td>278</td>
</tr>
<tr>
<td>Execution Control</td>
<td>279</td>
</tr>
<tr>
<td>Profiling</td>
<td>281</td>
</tr>
<tr>
<td>Compilation Commands</td>
<td>281</td>
</tr>
<tr>
<td>Program Execution</td>
<td>282</td>
</tr>
<tr>
<td>Presentation of the Results</td>
<td>283</td>
</tr>
<tr>
<td>Exercises</td>
<td>285</td>
</tr>
<tr>
<td>Tracing Function Application</td>
<td>285</td>
</tr>
<tr>
<td>Performance Analysis</td>
<td>285</td>
</tr>
<tr>
<td>Summary</td>
<td>286</td>
</tr>
<tr>
<td>To Learn More</td>
<td>286</td>
</tr>
</tbody>
</table>

## 11: Tools for lexical analysis and parsing

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lexicon</td>
<td>288</td>
</tr>
<tr>
<td>Module Genlex</td>
<td>288</td>
</tr>
<tr>
<td>Use of Streams</td>
<td>289</td>
</tr>
<tr>
<td>Regular Expressions</td>
<td>290</td>
</tr>
<tr>
<td>The Str Library</td>
<td>292</td>
</tr>
<tr>
<td>The ocamllex Tool</td>
<td>293</td>
</tr>
<tr>
<td>Syntax</td>
<td>295</td>
</tr>
<tr>
<td>Grammar</td>
<td>295</td>
</tr>
<tr>
<td>Production and Recognition</td>
<td>296</td>
</tr>
<tr>
<td>Top-down Parsing</td>
<td>297</td>
</tr>
<tr>
<td>Bottom-up Parsing</td>
<td>299</td>
</tr>
<tr>
<td>The ocamlyacc Tool</td>
<td>303</td>
</tr>
<tr>
<td>Contextual Grammars</td>
<td>305</td>
</tr>
<tr>
<td>Basic Revisited</td>
<td>307</td>
</tr>
<tr>
<td>File basic_parser.mly</td>
<td>307</td>
</tr>
<tr>
<td>File basic_lexer.mll</td>
<td>310</td>
</tr>
<tr>
<td>Compiling, Linking</td>
<td>311</td>
</tr>
<tr>
<td>Exercises</td>
<td>312</td>
</tr>
<tr>
<td>Filtering Comments Out</td>
<td>312</td>
</tr>
<tr>
<td>Evaluator</td>
<td>312</td>
</tr>
<tr>
<td>Summary</td>
<td>313</td>
</tr>
</tbody>
</table>
To Learn More .................................................. 313

12: Interoperability with C 315

Communication between C and Objective Caml ................. 317
  External declarations ........................................ 318
  Declaration of the C functions ............................... 318
  Linking with C .............................................. 320
  Mixing input-output in C and in Objective Caml ............. 323
Exploring Objective Caml values from C .......................... 323
  Classification of Objective Caml representations .......... 324
  Accessing immediate values ................................. 325
  Representation of structured values ......................... 326
Creating and modifying Objective Caml values from C .......... 335
  Modifying Objective Caml values ............................ 336
  Allocating new blocks ...................................... 337
  Storing C data in the Objective Caml heap .................. 338
  Garbage collection and C parameters and local variables .. 341
  Calling an Objective Caml closure from C ................. 343
Exception handling in C and in Objective Caml ................. 344
  Raising a predefined exception ............................ 344
  Raising a user-defined exception .......................... 345
  Catching an exception ..................................... 345
Main program in C ............................................ 347
  Linking Objective Caml code with C ....................... 347
Exercises ................................................................ 348
  Polymorphic Printing Function ............................... 348
  Matrix Product ............................................... 348
  Counting Words: Main Program in C ....................... 348
Summary .................................................................. 349
To Learn More .................................................. 349

13: Applications 351

Constructing a Graphical Interface ............................... 351
  Graphics Context, Events and Options ....................... 352
  Components and Containers .................................. 356
  Event Handling ................................................ 360
  Defining Components ........................................ 364
  Enriched Components ....................................... 376
  Setting up the Awi Library .................................. 377
  Example: A Franc-Euro Converter ........................... 378
  Where to go from here ....................................... 380
Finding Least Cost Paths ........................................ 381
  Graph Representations ..................................... 382
  Dijkstra’s Algorithm ......................................... 386
  Introducing a Cache ......................................... 390
A Graphical Interface ............................................. 392
Creating a Standalone Application ............................ 398
Final Notes ....................................................... 400

III Application Structure ......................................... 401

14: Programming with Modules ................................. 405

Modules as Compilation Units .................................. 406
  Interface and Implementation .................................. 406
  Relating Interfaces and Implementations ....................... 408
  Separate Compilation .......................................... 409
The Module Language ........................................... 410
  Two Stack Modules ............................................ 411
  Modules and Information Hiding ............................. 414
  Type Sharing between Modules ............................... 416
  Extending Simple Modules .................................... 418
Parameterized Modules ........................................... 418
  Functors and Code Reuse ...................................... 420
  Local Module Definitions .................................... 422
Extended Example: Managing Bank Accounts .................. 423
  Organization of the Program .................................. 423
  Signatures for the Module Parameters ........................ 424
  The Parameterized Module for Managing Accounts ............ 426
  Implementing the Parameters .................................. 427
Exercises .......................................................... 431
  Association Lists ............................................. 431
  Parameterized Vectors ....................................... 431
  Lexical Trees .................................................. 432
Summary .......................................................... 432
To Learn More .................................................... 433

15: Object-Oriented Programming ............................... 435

Classes, Objects, and Methods ................................. 436
  Object-Oriented Terminology .................................. 436
  Class Declaration .............................................. 437
  Instance Creation .............................................. 440
  Sending a Message ............................................. 440
Relations between Classes ...................................... 441
  Aggregation ..................................................... 441
  Inheritance Relation .......................................... 443
Other Object-oriented Features ............................... 445
  References: self and super .................................... 445
  Delayed Binding ............................................... 446
  Object Representation and Message Dispatch ............... 447
# Table of Contents

- Initialization .................................................. 448
- Private Methods ............................................. 449

## Types and Genericity ........................................
- Abstract Classes and Methods .............................. 450
- Classes, Types, and Objects ............................... 452
- Multiple Inheritance ...................................... 457
- Parameterized Classes .................................... 460

## Subtyping and Inclusion Polymorphism ..................
- Example ....................................................... 465
- Subtyping is not Inheritance .............................. 466
- Inclusion Polymorphism .................................... 468
- Equality between Objects .................................. 469

## Functional Style ...........................................

## Other Aspects of the Object Extension .................
- Interfaces ..................................................... 473
- Local Declarations in Classes ............................. 474

## Exercises ....................................................
- Stacks as Objects .......................................... 477
- Delayed Binding ............................................ 477
- Abstract Classes and an Expression Evaluator .......... 479
- The Game of Life and Objects ............................. 479

## Summary ......................................................

## To Learn More ................................................

## 16: Comparison of the Models of Organisation ........

Comparison of Modules and Objects ...................... 484
- Translation of Modules into Classes .................... 487
- Simulation of Inheritance with Modules ............... 489
- Limitations of each Model ................................ 490

Extending Components ...................................... 492
- In the Functional Model .................................. 493
- In the Object Model ...................................... 493
- Extension of Data and Methods ........................... 495

Mixed Organisations ....................................... 497

Exercises ....................................................... 498
- Classes and Modules for Data Structures .............. 498
- Abstract Types ............................................. 499

Summary ........................................................ 499

## To Learn More ................................................

## 17: Applications .............................................

Two Player Games ............................................. 501
- The Problem of Two Player Games ....................... 502
- Minimax $\alpha/\beta$ ...................................... 503
- Organization of a Game Program ....................... 510
# Table of Contents

Connect Four .......................................................... 515
Stonehenge ................................................................. 527
To Learn More ............................................................ 549

Fancy Robots ............................................................... 550
“Abstract” Robots ....................................................... 551
Pure World ................................................................. 553
Textual Robots ............................................................ 554
Textual World ............................................................. 556
Graphical Robots ........................................................ 559
Graphical World .......................................................... 562
To Learn More ............................................................ 563

IV  Concurrency and distribution 565

18: Communication and Processes 571

The Unix Module ......................................................... 572
Error Handling ............................................................ 573
Portability of System Calls ........................................... 573

File Descriptors ......................................................... 573
File Manipulation ....................................................... 575
Input / Output on Files ................................................ 576

Processes ................................................................. 579
Executing a Program ................................................... 579
Process Creation ........................................................ 581
Creation of Processes by Duplication ............................... 582
Order and Moment of Execution ..................................... 584
Descendence, Death and Funerals of Processes .................. 586

Communication Between Processes ................................. 587
Communication Pipes ................................................... 587
Communication Channels .............................................. 589
Signals under Unix ..................................................... 590

Exercises ................................................................. 595
Counting Words: the `wc` Command ................................. 595
Pipes for Spell Checking .............................................. 595
Interactive Trace ......................................................... 596

Summary ................................................................. 596
To Learn More ........................................................... 596

19: Concurrent Programming 599

Concurrent Processes .................................................. 600
Compilation with Threads ............................................. 601
Module Thread ........................................................... 602

Synchronization of Processes ........................................ 604
Critical Section and Mutual Exclusion ............................ 604
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mutex Module</td>
<td>604</td>
</tr>
<tr>
<td>Waiting and Synchronization</td>
<td>608</td>
</tr>
<tr>
<td>Condition Module</td>
<td>609</td>
</tr>
<tr>
<td>Synchronous Communication</td>
<td>612</td>
</tr>
<tr>
<td>Synchronization using Communication Events</td>
<td>612</td>
</tr>
<tr>
<td>Transmitted Values</td>
<td>612</td>
</tr>
<tr>
<td>Module Event</td>
<td>613</td>
</tr>
<tr>
<td>Example: Post Office</td>
<td>614</td>
</tr>
<tr>
<td>The Components</td>
<td>615</td>
</tr>
<tr>
<td>Clients and Clerks</td>
<td>617</td>
</tr>
<tr>
<td>The System</td>
<td>618</td>
</tr>
<tr>
<td>Exercises</td>
<td>619</td>
</tr>
<tr>
<td>The Philosophers Disentangled</td>
<td>619</td>
</tr>
<tr>
<td>More of the Post Office</td>
<td>619</td>
</tr>
<tr>
<td>Object Producers and Consumers</td>
<td>619</td>
</tr>
<tr>
<td>Summary</td>
<td>620</td>
</tr>
<tr>
<td>To Learn More</td>
<td>621</td>
</tr>
</tbody>
</table>

20: Distributed Programming 623

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>The Internet</td>
<td>624</td>
</tr>
<tr>
<td>The Unix Module and IP Addressing</td>
<td>625</td>
</tr>
<tr>
<td>Sockets</td>
<td>627</td>
</tr>
<tr>
<td>Description and Creation</td>
<td>627</td>
</tr>
<tr>
<td>Addresses and Connections</td>
<td>629</td>
</tr>
<tr>
<td>Client-server</td>
<td>630</td>
</tr>
<tr>
<td>Client-server Action Model</td>
<td>630</td>
</tr>
<tr>
<td>Client-server Programming</td>
<td>631</td>
</tr>
<tr>
<td>Code for the Server</td>
<td>632</td>
</tr>
<tr>
<td>Testing with <code>telnet</code></td>
<td>634</td>
</tr>
<tr>
<td>The Client Code</td>
<td>635</td>
</tr>
<tr>
<td>Client-server Programming with Lightweight Processes</td>
<td>639</td>
</tr>
<tr>
<td>Multi-tier Client-server Programming</td>
<td>642</td>
</tr>
<tr>
<td>Some Remarks on the Client-server Programs</td>
<td>642</td>
</tr>
<tr>
<td>Communication Protocols</td>
<td>643</td>
</tr>
<tr>
<td>Text Protocol</td>
<td>644</td>
</tr>
<tr>
<td>Protocols with Acknowledgement and Time Limits</td>
<td>646</td>
</tr>
<tr>
<td>Transmitting Values in their Internal Representation</td>
<td>646</td>
</tr>
<tr>
<td>Interoperating with Different Languages</td>
<td>647</td>
</tr>
<tr>
<td>Exercises</td>
<td>647</td>
</tr>
<tr>
<td>Service: Clock</td>
<td>648</td>
</tr>
<tr>
<td>A Network Coffee Machine</td>
<td>648</td>
</tr>
<tr>
<td>Summary</td>
<td>649</td>
</tr>
<tr>
<td>To Learn More</td>
<td>649</td>
</tr>
</tbody>
</table>

21: Applications 651
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Client-server Toolbox</td>
<td>651</td>
</tr>
<tr>
<td>Protocols</td>
<td>652</td>
</tr>
<tr>
<td>Communication</td>
<td>652</td>
</tr>
<tr>
<td>Server</td>
<td>653</td>
</tr>
<tr>
<td>Client</td>
<td>655</td>
</tr>
<tr>
<td>To Learn More</td>
<td>656</td>
</tr>
<tr>
<td>The Robots of Dawn</td>
<td>656</td>
</tr>
<tr>
<td>World-Server</td>
<td>657</td>
</tr>
<tr>
<td>Observer-client</td>
<td>661</td>
</tr>
<tr>
<td>Robot-Client</td>
<td>663</td>
</tr>
<tr>
<td>To Learn More</td>
<td>665</td>
</tr>
<tr>
<td>HTTP Servlets</td>
<td>665</td>
</tr>
<tr>
<td>HTTP and CGI Formats</td>
<td>666</td>
</tr>
<tr>
<td>HTML Servlet Interface</td>
<td>671</td>
</tr>
<tr>
<td>Dynamic Pages for Managing the Association Database</td>
<td>674</td>
</tr>
<tr>
<td>Analysis of Requests and Response</td>
<td>676</td>
</tr>
<tr>
<td>Main Entry Point and Application</td>
<td>676</td>
</tr>
<tr>
<td>22: Developing applications with Objective Caml</td>
<td>679</td>
</tr>
<tr>
<td>Elements of the evaluation</td>
<td>680</td>
</tr>
<tr>
<td>Language</td>
<td>680</td>
</tr>
<tr>
<td>Libraries and tools</td>
<td>681</td>
</tr>
<tr>
<td>Documentation</td>
<td>682</td>
</tr>
<tr>
<td>Other development tools</td>
<td>682</td>
</tr>
<tr>
<td>Editing tools</td>
<td>683</td>
</tr>
<tr>
<td>Syntax extension</td>
<td>683</td>
</tr>
<tr>
<td>Interoperability with other languages</td>
<td>683</td>
</tr>
<tr>
<td>Graphical interfaces</td>
<td>683</td>
</tr>
<tr>
<td>Parallel programming and distribution</td>
<td>684</td>
</tr>
<tr>
<td>Applications developed in Objective Caml</td>
<td>685</td>
</tr>
<tr>
<td>Similar functional languages</td>
<td>686</td>
</tr>
<tr>
<td>ML family</td>
<td>686</td>
</tr>
<tr>
<td>Scheme</td>
<td>687</td>
</tr>
<tr>
<td>Languages with delayed evaluation</td>
<td>688</td>
</tr>
<tr>
<td>Communication languages</td>
<td>690</td>
</tr>
<tr>
<td>Object-oriented languages: comparison with Java</td>
<td>691</td>
</tr>
<tr>
<td>Main characteristics</td>
<td>691</td>
</tr>
<tr>
<td>Differences with Objective Caml</td>
<td>691</td>
</tr>
<tr>
<td>Future of Objective Caml development</td>
<td>693</td>
</tr>
<tr>
<td>Conclusion</td>
<td>695</td>
</tr>
</tbody>
</table>
# Table of Contents

## V Appendices

### A: Cyclic Types

Cyclic types 699
Option `-rectypes` 701

### B: Objective Caml 3.04

Language Extensions 703
Labels 704
Optional arguments 706
Labels and objects 708
Polymorphic variants 709

LablTk Library 712
OCamlBrowser 712

**Bibliography** 715

**Index of concepts** 719

**Index of language elements** 725
**Introduction**

**Objective Caml** is a programming language. One might ask why yet another language is needed. Indeed there are already numerous existing languages with new ones constantly appearing. Beyond their differences, the conception and genesis of each one of them proceeds from a shared motivation: the desire to abstract.

**To abstract from the machine** In the first place, a programming language permits one to neglect the “mechanical” aspect of the computer; it even lets one forget the microprocessor model or the operating system on which the program will be executed.

**To abstract from the operational model** The notion of function which most languages possess in one form or another is borrowed from mathematics and not from electronics. In a general way, languages substitute formal models for purely computational viewpoints. Thus they gain expressivity.

**To abstract errors** This has to do with the attempt to guarantee execution safety; a program shouldn’t terminate abruptly or become inconsistent in case of an error. One of the means of attaining this is strong static typing of programs and having an exception mechanism in place.

**To abstract components** (i) Programming languages make it possible to subdivide an application into different software components which are more or less independent and autonomous. Modularity permits higher-level structuring of the whole of a complex application.

**To abstract components** (ii) The existence of programming units has opened up the possibility of their reuse in contexts other than the ones for which they were developed. Object-oriented languages constitute another approach to reusability permitting rapid prototyping.

Objective Caml is a recent language which takes its place in the history of programming languages as a distant descendant of Lisp, having been able to draw on the lessons
of its cousins while incorporating the principal characteristics of other languages. It is
developed at INRIA and is supported by long experience with the conception of the
languages in the ML family. Objective Caml is a general-purpose language for the
expression of symbolic and numeric algorithms. It is object-oriented and has a param-
eterized module system. It supports the development of concurrent and distributed
applications. It has excellent execution safety thanks to its static typing, its exception
mechanism and its garbage collector. It is high-performance while still being portable.
Finally, a rich development environment is available.

Objective Caml has never been the subject of a presentation to the “general public”.
This is the task to which the authors have set themselves, giving this exposition three
objectives:

1. To describe in depth the Objective Caml language, its libraries and its develop-
   ment environment.
2. To show and explain what are the concepts hidden behind the programming
   styles which can be used with Objective Caml.
3. To illustrate through numerous examples how Objective Caml can serve as the
   development language for various classes of applications.

The authors’ goal is to provide insight into how to choose a programming style and
structure a program, consistent with a given problem, so that it is maintainable and
its components are reusable.

Description of the language

Objective Caml is a functional language: it manipulates functions as values in
the language. These can in turn be passed as arguments to other functions or returned
as the result of a function call.

Objective Caml is statically typed: verification of compatibility between the
types of formal and actual parameters is carried out at program compilation time.
From then on it is not necessary to perform such verification during the execution of
the program, which increases its efficiency. Moreover, verification of typing permits the
elimination of most errors introduced by typos or thoughtlessness and contributes to
execution safety.

Objective Caml has parametric polymorphism: a function which does not tra-
verse the totality of the structure of one of its arguments accepts that the type of this
argument is not fully determined. In this case this parameter is said to be polymorphic.
This feature permits development of generic code usable for different data structures,

---
1. Institut National de Recherche en Informatique et Automatique (National Institute for Research
   in Automation and Information Technology).
such that the exact representation of this structure need not be known by the code in question. The typing algorithm is in a position to make this distinction.

**Objective Caml has type inference:** the programmer need not give any type information within the program. The language alone is in charge of deducing from the code the most general type of the expressions and declarations therein. This inference is carried out jointly with verification, during program compilation.

**Objective Caml is equipped with an exception mechanism:** it is possible to interrupt the normal execution of a program in one place and resume at another place thanks to this facility. This mechanism allows control of exceptional situations, but it can also be adopted as a programming style.

**Objective Caml has imperative features:** I/O, physical modification of values and iterative control structures are possible without having recourse to functional programming features. Mixture of the two styles is acceptable, and offers great development flexibility as well as the possibility of defining new data structures.

**Objective Caml executes (threads):** the principal tools for creation, synchronization, management of shared memory, and interthread communication are predefined.

**Objective Caml communicates on the Internet:** the support functions needed to open communication channels between different machines are predefined and permit the development of client-server applications.

**Numerous libraries are available for Objective Caml:** classic data structures, I/O, interfacing with system resources, lexical and syntactic analysis, computation with large numbers, persistent values, etc.

**A programming environment is available for Objective Caml:** including interactive toplevel, execution trace, dependency calculation and profiling.

**Objective Caml interfaces with the C language:** by calling C functions from an Objective Caml program and vice versa, thus permitting access to numerous C libraries.

**Three execution modes are available for Objective Caml:** interactive by means of an interactive toplevel, compilation to bytecodes interpreted by a virtual machine, compilation to native machine code. The programmer can thus choose between
flexibility of development, portability of object code between different architectures, or performance on a given architecture.

**Structure of a program**

Development of important applications requires the programmer or the development team to consider questions of organization and structure. In Objective Caml, two models are available with distinct advantages and features.

**The parameterized module model:** data and procedures are gathered within a single entity with two facets: the code proper, and its interface. Communication between modules takes place via their interface. The description of a type may be hidden, not appearing in the module interface. These abstract data types facilitate modifications of the internal implementation of a module without affecting other modules which use it. Moreover, modules can be parameterized by other modules, thus increasing their reusability.

**The object model:** descriptions of procedures and data are gathered into entities called *classes*; an object is an instance (value) of a class. Interobject communication is implemented through “message passing”, the receiving object determines upon execution (late binding) the procedure corresponding to the message. In this way, object-oriented programming is “data-driven”. The program structure comes from the relationships between classes; in particular inheritance lets one class be defined by extending another. This model allows concrete, abstract and parameterized classes. Furthermore, it introduces polymorphism of inclusion by defining the subtyping relationship between classes.

The choice between these two models allows great flexibility in the logical organization of an application and facilitates its maintenance and evolution. There is a duality between these two models. One cannot add data fields to a module type (no extensibility of data), but one can add new procedures (extensibility of procedures) acting on data. In the object model, one can add subclasses of a class (extensibility of data) for dealing with new cases, but one cannot add new procedures visible from the ancestor class (no extensibility of procedures). Nevertheless the combination of the two offers new possibilities for extending data and procedures.

**Safety and efficiency of execution**

Objective Caml bestows excellent execution safety on its programs without sacrificing their efficiency. Fundamentally, static typing is a guarantee of the absence of runtime type errors and makes useful static information available to the compiler without burdening performance with dynamic type tests. These benefits also extend to the object-oriented language features. Moreover, the built-in garbage collector adds to the safety of the language system. Objective Caml’s is particularly efficient. The exception
mechanism guarantees that the program will not find itself in an inconsistent state after a division by zero or an access outside the bounds of an array.

Outline of the book

The present work consists of four main parts, bracketed by two chapters and enhanced by two appendices, a bibliography, an index of language elements and an index of programming concepts.

Chapter 1: This chapter describes how to install version 2.04 of the Objective Caml language on the most current systems (Windows, Unix and MacOS).

Part I: Core of the language The first part is a complete presentation of the basic elements of the Objective Caml language. Chapter 2 is a dive into the functional core of the language. Chapter 3 is a continuation of the previous one and describes the imperative part of the language. Chapter 4 compares the “pure” functional and imperative styles, then presents their joint use. Chapter 5 presents the graphics library. Chapter 6 exhibits three applications: management of a simple database, a mini-Basic interpreter and a well-known single-player game, minesweeper.

Part II: Development tools The second part of the book describes the various tools for application development. Chapter 7 compares the various compilation modes, which are the interactive toplevel and command-line bytecode and native code compilers. Chapter 8 presents the principal libraries provided with the language distribution. Chapter 9 explains garbage collection mechanisms and details the one used by Objective Caml. Chapter 10 explains the use of tools for debugging and profiling programs. Chapter 11 addresses lexical and syntactic tools. Chapter 12 shows how to interface Objective Caml programs with C. Chapter 13 constructs a library and an application. This library offers tools for the construction of GUIs. The application is a search for least-cost paths within a graph, whose GUI uses the preceding library.

Part III: Organization of applications The third part describes the two ways of organizing a program: with modules, and with objects. Chapter 14 is a presentation of simple and parameterized language modules. Chapter 15 introduces Objective Caml object-oriented extension. Chapter 16 compares these two types of organization and indicates the usefulness of mixing them to increase the extensibility of programs. Chapter 17 describes two substantial applications: two-player games which put to work several parameterized modules used for two different games, and a simulation of a robot world demonstrating interobject communication.

Part IV: Concurrence and distribution The fourth part introduces concurrent and distributed programs while detailing communication between processes, lightweight or not, and on the Internet. Chapter 18 demonstrates the direct link between the language and the system libraries, in particular the notions of process and
communication. Chapter 19 leads to the lack of determinism of concurrent programming while presenting Objective Caml’s threads. Chapter 20 discusses interprocess communication via sockets in the distributed memory model. Chapter 21 presents first of all a toolbox for client-server applications. It is subsequently used to extend the robots of the previous part to the client-server model. Finally, we adapt some of the programs already encountered in the form of an HTTP server.

Chapter 22 This last chapter takes stock of application development in Objective Caml and presents the best-known applications of the ML language family.

Appendices The first appendix explains the notion of cyclic types used in the typing of objects. The second appendix describes the language changes present in the new version 3.00. These have been integrated in all following versions of Objective Caml (3.xx).

Each chapter consists of a general presentation of the subject being introduced, a chapter outline, the various sections thereof, statements of exercises to carry out, a summary, and a final section entitled “To learn more” which indicates bibliographic references for the subject which has been introduced.