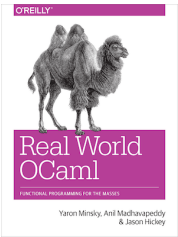



*Buy in print and eBook.*

Table of Contents





# Prologue

## WHY OCAML?

Programming languages matter. They affect the reliability, security, and efficiency of the code you write, as well as how easy it is to read, refactor, and extend. The languages you know can also change how you think, influencing the way you design software even when you're not using them.

We wrote this book because we believe in the importance of programming languages, and that OCaml in particular is an important language to learn. The three of us have been using OCaml in our academic and professional lives for over 15 years, and in that time we've come to see it as a secret weapon for building complex software systems. This book aims to make this secret weapon available to a wider audience, by providing a clear guide to what you need to know to use OCaml effectively in the real world.

What makes OCaml special is that it occupies a sweet spot in the space of programming language designs. It provides a combination of efficiency, expressiveness and practicality that is matched by no other language. That is in large part because OCaml is an elegant combination of a few key language features that have been developed over the last 40 years. These include:

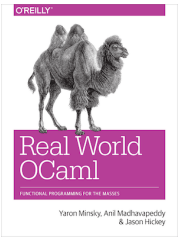
- *Garbage collection* for automatic memory management, now a feature of almost every modern, high-level language.
- *First-class functions* that can be passed around like ordinary values, as seen in JavaScript, Common Lisp, and C#.
- *Static type-checking* to increase performance and reduce the number of runtime errors, as found in Java and C#.
- *Parametric polymorphism*, which enables the construction of abstractions that work across different data types, similar to generics in Java and C# and templates in C++.
- Good support for *immutable programming*, i.e., programming without making destructive updates to data structures. This is present in traditional functional languages like Scheme, and is also found in distributed, big-data frameworks like Hadoop.
- *Automatic type inference* to avoid having to laboriously define the type of every single variable in a program and instead have them inferred based on how a value is used. Available in a limited form in C# with implicitly typed local variables, and in C++11 with its `auto` keyword.
- *Algebraic data types* and *pattern matching* to define and manipulate complex data structures. Available in Scala and F#.

Some of you will know and love all of these features, and for others they will be largely new, but most of you will have seen *some* of them in other languages that you've used. As we'll demonstrate over the course of this book, there is something transformative about having them all together and able to interact in a single language. Despite their importance, these ideas have made only limited inroads into mainstream languages, and when they do arrive there, like first-class functions in C# or parametric polymorphism in Java, it's typically in a limited and awkward form. The only languages that completely embody these ideas are *statically typed, functional programming languages* like OCaml, F#, Haskell, Scala, and Standard ML.

Among this worthy set of languages, OCaml stands apart because it manages to provide a great deal of power while remaining highly pragmatic. The compiler has a straightforward compilation strategy that produces performant code without requiring heavy optimization and without the complexities of dynamic just-in-time (JIT) compilation. This, along with OCaml's strict evaluation model, makes runtime behavior easy to predict. The garbage collector is *incremental*, letting you avoid large garbage collection (GC)-related pauses, and *precise*, meaning it will collect all unreferenced data (unlike many reference-counting collectors), and the runtime is simple and highly portable.

All of this makes OCaml a great choice for programmers who want to step up to a better programming language, and at the same time get practical work done.

### A Brief History




*Buy in print and eBook.*

Table of Contents





OCaml was written in 1996 by Xavier Leroy, Jérôme Vouillon, Damien Doligez, and Didier Rémy at INRIA in France. It was inspired by a long line of research into ML starting in the 1960s, and continues to have deep links to the academic community.

ML was originally the *meta language* of the LCF (Logic for Computable Functions) proof assistant released by Robin Milner in 1972 (at Stanford, and later at Cambridge). ML was turned into a compiler in order to make it easier to use LCF on different machines, and it was gradually turned into a full-fledged system of its own by the 1980s.

The first implementation of Caml appeared in 1987. It was created by Ascánder Suárez and later continued by Pierre Weis and Michel Mauny. In 1990, Xavier Leroy and Damien Doligez built a new implementation called Caml Light that was based on a bytecode interpreter with a fast, sequential garbage collector. Over the next few years useful libraries appeared, such as Michel Mauny's syntax manipulation tools, and this helped promote the use of Caml in education and research teams.

Xavier Leroy continued extending Caml Light with new features, which resulted in the 1995 release of Caml Special Light. This improved the executable efficiency significantly by adding a fast native code compiler that made Caml's performance competitive with mainstream languages such as C++. A module system inspired by Standard ML also provided powerful facilities for abstraction and made larger-scale programs easier to construct.

The modern OCaml emerged in 1996, when a powerful and elegant object system was implemented by Didier Rémy and Jérôme Vouillon. This object system was notable for supporting many common object-oriented idioms in a statically type-safe way, whereas the same idioms required runtime checks in languages such as C++ or Java. In 2000, Jacques Garrigue extended OCaml with several new features such as polymorphic methods, variants, and labeled and optional arguments.

The last decade has seen OCaml attract a significant user base, and language improvements have been steadily added to support the growing commercial and academic codebases. First-class modules, Generalized Algebraic Data Types (GADTs), and dynamic linking have improved the flexibility of the language. There is also fast native code support for x86_64, ARM, PowerPC, and Sparc, making OCaml a good choice for systems where resource usage, predictability, and performance all matter.

### The Core Standard Library

A language on its own isn't enough. You also need a rich set of libraries to base your applications on. A common source of frustration for those learning OCaml is that the standard library that ships with the compiler is limited, covering only a small subset of the functionality you would expect from a general-purpose standard library. That's because the standard library isn't a general-purpose tool; it was developed for use in bootstrapping the compiler and is purposefully kept small and simple.

Happily, in the world of open source software, nothing stops alternative libraries from being written to supplement the compiler-supplied standard library, and this is exactly what the Core distribution is.

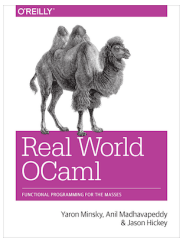
Jane Street, a company that has been using OCaml for more than a decade, developed Core for its own internal use, but designed it from the start with an eye toward being a general-purpose standard library. Like the OCaml language itself, Core is engineered with correctness, reliability, and performance in mind.

Core is distributed with syntax extensions that provide useful new functionality to OCaml, and there are additional libraries such as the Async network communications library that extend the reach of Core into building complex distributed systems. All of these libraries are distributed under a liberal Apache 2 license to permit free use in hobby, academic, and commercial settings.

### The OCaml Platform

Core is a comprehensive and effective standard library, but there's much more OCaml software out there. A large community of programmers has been using OCaml since its first release in 1996, and has generated many useful libraries and tools. We'll introduce some of these libraries in the course of the examples presented in the book.

The installation and management of these third-party libraries is made much easier via a package management tool known as OPAM. We'll explain more about OPAM as the book unfolds, but it forms the basis of the Platform, which is a set of tools and libraries that, along with the OCaml compiler, lets you build real-world applications quickly and effectively.










We'll also use OPAM for installing the **utop** command-line interface. This is a modern interactive tool that supports command history, macro expansion, module completion, and other niceties that make it much more pleasant to work with the language. We'll be using **utop** throughout the book to let you step through the examples interactively.

## ABOUT THIS BOOK

*Real World OCaml* is aimed at programmers who have some experience with conventional programming languages, but not specifically with statically typed functional programming. Depending on your background, many of the concepts we cover will be new, including traditional functional-programming techniques like higher-order functions and immutable data types, as well as aspects of OCaml's powerful type and module systems.

If you already know OCaml, this book may surprise you. Core redefines most of the standard namespace to make better use of the OCaml module system and expose a number of powerful, reusable data structures by default. Older OCaml code will still interoperate with Core, but you may need to adapt it for maximal benefit. All the new code that we write uses Core, and we believe the Core model is worth learning; it's been successfully used on large, multimillion-line codebases and removes a big barrier to building sophisticated applications in OCaml.

Code that uses only the traditional compiler standard library will always exist, but there are other online resources for learning how that works. *Real World OCaml* focuses on the techniques the authors have used in their personal experience to construct scalable, robust software systems.

### What to Expect

*Real World OCaml* is split into three parts:

- Part I covers the language itself, opening with a guided tour designed to provide a quick sketch of the language. Don't expect to understand everything in the tour; it's meant to give you a taste of many different aspects of the language, but the ideas covered there will be explained in more depth in the chapters that follow.

  After covering the core language, Part I then moves onto more advanced features like modules, functors, and objects, which may take some time to digest. Understanding these concepts is important, though. These ideas will put you in good stead even beyond OCaml when switching to other modern languages, many of which have drawn inspiration from ML.

- Part II builds on the basics by working through useful tools and techniques for addressing common practical applications, from command-line parsing to asynchronous network programming. Along the way, you'll see how some of the concepts from Part I are glued together into real libraries and tools that combine different features of the language to good effect.

- Part III discusses OCaml's runtime system and compiler toolchain. It is remarkably simple when compared to some other language implementations (such as Java's or .NET's CLR). Reading this part will enable you to build very-high-performance systems, or to interface with C libraries. This is also where we talk about profiling and debugging techniques using tools such as GNU **gdb**.

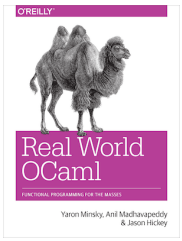
### Installation Instructions

*Real World OCaml* uses some tools that we've developed while writing this book. Some of these resulted in improvements to the OCaml compiler, which means that you will need to ensure that you have an up-to-date development environment (using the 4.01 version of the compiler). The installation process is largely automated through the OPAM package manager. Instructions on how to it set up and what packages to install can be found at this Real World OCaml page.

As of publication time, the Windows operating system is unsupported by Core, and so only Mac OS X, Linux, FreeBSD, and OpenBSD can be expected to work reliably. Please check the online installation instructions for updates regarding Windows, or install a Linux virtual machine to work through the book as it stands.

This book is not intended as a reference manual. We aim to teach you about the language and about libraries tools and techniques that will help you be a more effective OCaml programmer. But it's no replacement for API documentation or the OCaml manual and man pages. You can find documentation for all of the libraries and tools referenced in the book online.

### Code Examples




*Buy in print and eBook.*

Table of Contents





All of the code examples in this book are available freely online under a public-domain-like license. You are most welcome to copy and use any of the snippets as you see fit in your own code, without any attribution or other restrictions on their use.

The code repository is available online at https://github.com/realworldocaml/examples. Every code snippet in the book has a clickable header that tells you the filename in that repository to find the source code, shell script, or ancillary data file that the snippet was sourced from.

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at <permissions@oreilly.com>.

## SAFARI® BOOKS ONLINE

> **Note**
>
> Safari Books Online (www.safaribooksonline.com) is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business. Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

## HOW TO CONTACT US

Please address comments and questions concerning this book to the publisher:

| |
|---|
| O'Reilly Media, Inc. |
| 1005 Gravenstein Highway North |
| Sebastopol, CA 95472 |
| 800-998-9938 (in the United States or Canada) |
| 707-829-0515 (international or local) |
| 707-829-0104 (fax) |

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://oreil.ly/realworldOCaml

To comment or ask technical questions about this book, send email to:

<bookquestions@oreilly.com>

For more information about our books, courses, conferences, and news, see our website at http://www.oreilly.com.

Find us on Facebook: http://facebook.com/oreilly

Follow us on Twitter: http://twitter.com/oreillymedia

Watch us on YouTube: http://www.youtube.com/oreillymedia

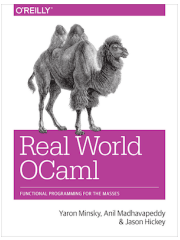
## CONTRIBUTORS

We would especially like to thank the following individuals for improving *Real World OCaml*:

- Leo White contributed greatly to the content and examples in Chapter 11, *Objects* and Chapter 12, *Classes*.
- Jeremy Yallop authored and documented the Ctypes library described in Chapter 19, *Foreign Function Interface*.
- Stephen Weeks is responsible for much of the modular architecture behind Core, and his extensive notes formed the basis of Chapter 20, *Memory Representation of Values* and

Chapter 21, *Understanding the Garbage Collector*.

- Jeremie Dimino, the author of *utop*, the interactive command-line interface that is used throughout this book. We're particularly grateful for the changes that he pushed through to make *utop* work better in the context of the book.
- The many people who collectively submitted over 2400 comments to online drafts of this book, through whose efforts countless errors were found and fixed.






**< Previous** **Next >**

Table of Contents