F# for Scientists

Jon Harrop
Flying Frog Consultancy Ltd.

Foreword by Don Syme

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

This Page Intentionally Left Blank

F# for Scientists

This Page Intentionally Left Blank

F# for Scientists

Jon Harrop
Flying Frog Consultancy Ltd.

Foreword by Don Syme

WILEY

A JOHN WILEY & SONS, INC., PUBLICATION

Published by John Wiley & Sons, Inc., Hoboken, New Jersey.
Published simultaneously in Canada.

Library of Congress Cataloging-in-Publication Data:

Harrop, Jon D.
F# for scientists / Jon Harrop.
p. cm.
Includes index.
ISBN 978-0-470-24211-7 (cloth)
1. F# (Computer program language) 2. Functional programming (Computer science) 3. Science—Data processing. I. Title.
QA76.73.F163H37 2008
005.1'14—dc22 2008009567

Printed in the United States of America.

10 9 8 7 6 5 4 3 2 1

To my family

This Page Intentionally Left Blank

Contents in Brief

This Page Intentionally Left Blank

CONTENTS

This Page Intentionally Left Blank

Foreword

Computational science is one of the wonders of the modern world. In almost all areas of science the use of computational techniques is rocketing, and software has moved from being a supporting tool to being a key site where research activities are performed. This has meant a huge increase in the importance of controlling and orchestrating computers as part of the daily routine of a scientific laboratory, from large teams making and running the computers performing global climate simulations to the individual scientist/programmer working alone. Across this spectrum, the productivity of teams and the happiness of scientists depends dramatically on their overall competency as programmers, as well as on their skills as researchers within their field. So, in the last 30 years we have seen the continued rise of that new profession: the *scientific programmer*. A good scientific programmer will carry both epithets with pride, knowing that programming is a key foundation for a successful publication record.

However, programming cultures differ widely, and, over time, gaping divides can emerge that can be to the detriment of all. In this book, Dr. Harrop has taken great steps forward to bridging three very different cultures: *managed code programming*, *scientific programming* and *functional programming*. At a technical level, each has its unique characteristics. Managed code programming, epitomized by .NET and Java, focuses on the productivity of the (primarily commercial) programmer. Scientific programmers focus on high performance computations, data manipulation, numerical

computing and visualization. Functional programming focuses on crisp, declarative solutions to problems using compositional techniques. The challenge, then, is to bring these disparate worlds together in a productive way.

The language F#, which Dr. Harrop uses in this book, itself bridges two of these cultures by being a functional language for the .NET platform. F# is an incredibly powerful language: the .NET libraries give a rich and solid foundation of software functionality for many tasks, from routine programming to accessing web services and high performance graphics engines. F# brings an approach to programming that routinely makes even short programs powerful, simple, elegant and correct. However Dr. Harrop has gone a step further, showing how managed code functional programming can revolutionize the art of scientific programming itself by being a powerful workhorse tool that unifies and simplifies many of the tasks scientific programmers face.

But what of the future? The next 20 years will see great changes in scientific programming. It is customary to mention the ever-increasing challenges of parallel, concurrent, distributed and reactive programming. It is widely expected that future micro-processors will use ever-increasing transistor counts to host multiple processing cores, rather than more sophisticated microprocessor designs. If computations can be parallelized and distributed on commodity hardware then the computing resources that can be brought can be massively increased. It is well known that successful concurrent and distributed computing requires a combination of intelligent algorithm design, competent programming, and core components that abstract some details of concurrent execution, e.g. databases and task execution libraries. This needs a language that can interoperate with key technologies such as databases, and parallelism engines. Furthermore, the ability to declaratively and crisply describe solutions to concurrent programming problems is essential, and F# is admirably suited to this task.

The future, will, however, bring other challenges as well. Truly massive amounts of data are now being generated by scientific experiments. Web-based programming will become more and more routine for scientific teams: a good web application can revolutionize a scientific field. Shared databases will soon be used in almost every scientific field, and programmatic access to these will be essential. F# lends itself to these challenges: for example, it is relatively easy to perform sophisticated and high-performance analysis of these data sources by bringing them under the static type discipline of functional programming, as shown by some of the samples in this book.

You will learn much about both programming and science through this book. Dr. Harrop has chosen the style of F# programming most suited to the individual scientist: crisp, succinct and efficient, with a discursive presentation style reminiscent of Mathematica. It has been a pleasure to read, and we trust it will launch you on a long and productive career as a managed code, functional scientific programmer.

Preface

The face of scientific computing has changed. Computational scientists are no longer writing their programs in Fortran and competing for time on supercomputers. Scientists are now streamlining their research by choosing more expressive programming languages, parallel processing on desktop machines and exploiting the wealth of scientific information distributed across the internet.

The landscape of programming languages saw a punctuation in its evolution at the end of the 20th century, marked by the advent of a new breed of languages. These new languages incorporate a multitude of features that are all designed to serve a single purpose: to make life easier. Modern programming languages offer so much more expressive power than traditional languages that they even open up new avenues of scientific research that were simply intractable before.

The next few years will usher in a new era of computing, where parallelism becomes ubiquitous. Few approaches to programming will survive this transition, and functional programming is one of them.

Seamlessly interoperating with computers across the world is of pivotal importance not only because of the breadth of information now available on-line but also because this is the only practicable way to interrogate the enormous amount of data available. The amount of genomic and proteinomic data published every year continues to grow exponentially, as each generation of technology fuels the next.

Only one mainstream programming language combines awesome expressive power, interoperability and performance: F#. This book introduces all of the aspects of the F# programming language needed by a working scientist, emphasizing aspects not covered by existing literature. Consequently, this book is the ideal complement to a detailed overview of the language itself, such as the F# manual or the book Expert F#[25].

Chapters 1–5 cover the most important aspects of F# programming needed to start developing useful F# programs. Chapter 6 ossifies this knowledge with a variety of enlightening and yet simple examples. Chapters 7–11 cover advanced topics including real-time visualization, interoperability and parallel computing. Chapter 12 concludes the book with a suite of complete working programs relevant to scientific computing.

The source code from this book is available from the following website:

`http://www.ffconsultancy.com/products/fsharp_for_scientists/`

J. D. HARROP

Cambridge, UK
June, 2008

Acknowledgments

I would like the thank Don Syme, the creator of F#, for pioneering research into programming languages and for thrusting the incredibly powerful ML family of languages into the limelight of mainstream software development.

Xavier Leroy, everyone at projet Cristal and the Debian package maintainers for making OCaml so practically useful.

Stephen Elliott and Sergei Taraskin and their group at the University of Cambridge for teaching me how to be a research scientist and letting me pursue crazy ideas when I should have been working.

Ioannis Baltopoulos and Enrique Nell for proofreading this book and giving essential feedback.

J. D. H.

This Page Intentionally Left Blank

List of Figures