THE EXPERT'S VOICE® IN .NET

Beginning F# 4.0

Second Edition

Robert Pickering
Kit Eason
Foreword by Don Syme, the inventor of F#

EXTRAS ONLINE

Apress®

Beginning F# 4.0

Second Edition

Robert Pickering
Kit Eason

Apress®

Beginning F# 4.0

Robert Pickering
St. Germain-En-Laye
France

Kit Eason

ISBN-13 (pbk): 978-1-4842-1375-9
ISBN-13 (electronic): 978-1-4842-1374-2
DOI 10.1007/978-1-4842-1374-2

Managing Director: Welmoed Spahr
Lead Editor: James DeWolf
Development Editor: Douglas Pundick
Technical Reviewer: Fabio Claudio Ferracchiati
Editorial Board: Steve Anglin, Pramila Balen, Louise Corrigan, James DeWolf, Jonathan Gennick, Robert Hutchinson, Celestin Suresh John, Michelle Lowman, James Markham, Susan McDermott, Matthew Moodie, Jeffrey Pepper, Douglas Pundick, Ben Renow-Clarke, Gwenan Spearing
Coordinating Editor: Melissa Maldonado
Copy Editor: Mary Behr
Compositor: SPi Global
Indexer: SPi Global
Artist: SPi Global

Distributed to the book trade worldwide by Springer Science+Business Media New York, 233 Spring Street, 6th Floor, New York, NY 10013. Phone 1-800-SPRINGER, fax (201) 348-4505, e-mail orders-ny@springer-sbm.com, or visit www.springer.com. Apress Media, LLC is a California LLC and the sole member (owner) is Springer Science + Business Media Finance Inc (SSBM Finance Inc). SSBM Finance Inc is a Delaware corporation.

For information on translations, please e-mail rights@apress.com, or visit www.apress.com.

Apress and friends of ED books may be purchased in bulk for academic, corporate, or promotional use. eBook versions and licenses are also available for most titles. For more information, reference our Special Bulk Sales–eBook Licensing web page at www.apress.com/bulk-sales.

Any source code or other supplementary material referenced by the author in this text is available to readers at www.apress.com. For detailed information about how to locate your book's source code, go to www.apress.com/source-code/.

Printed on acid-free paper

For Noah and Darwin.

Contents at a Glance

Contents

About the Authors

Robert Pickering is a Microsoft MVP acknowledged as a community leader and a world class expert on F#. In his ten year career as a software engineer he has participated in a wide range of projects building large, scalable, and fault tolerant systems on the Microsoft .NET platform. He has experience in both consulting and working as an engineer for a software house. He has been invited to speak at prestigious events such as Microsoft TechEd, JAOO, and TechDays Paris. He has been involved in organizing conferences as track lead for events such as Functional Programming eXchange, London, and he organizes the ALT.NET community meetings in Paris. He has also appeared in podcast and screen casts such as `dnrtv.com` and `hanselminutes.com`.

Kit Eason is a software developer and educator with 30 years of experience in industries from automotive engineering through university supercomputing to energy trading. He currently works as a senior developer at Adbrain (`www.adbrain.com`), a leading data intelligence company that empowers marketers to regain control of their customer relationships in today's hyperconnected yet highly fragmented world. Kit also works as an author for Pluralsight: his courses there include "F# Jumpstart" and "F# Functional Data Structures." He has also appeared on the ".NET Rocks!" podcast.

About the Technical Reviewer

Fabio Claudio Ferracchiati is a senior consultant and a senior analyst/developer using Microsoft technologies. He works for Blu Arancio (www.bluarancio.com). He is a Microsoft Certified Solution Developer for .NET, a Microsoft Certified Application Developer for .NET, a Microsoft Certified Professional, and a prolific author and technical reviewer. Over the past ten years, he's written articles for Italian and international magazines and coauthored more than ten books on a variety of computer topics.

Acknowledgments

The first vote of thanks must go to the person who changed my professional life, and those of many others: Don Syme, the "father of F#." Don does invaluable work in mentoring and encouraging me and the whole F# community. He sets a positive tone that is reflected in everything the community does.

Huge thanks also to Fabio Claudio Ferracchiati, our technical reviewer, who has turned around reams of material in what seems like record time.

Thanks also to the various employers who have supported me (admittedly with varying degrees of scepticism!) during my journey into F#. Extra special thanks to Edoardo Turelli and all the team at Adbrain, who have somehow unlocked the secret of having the right people working on the right problems in the right way.

I'm deeply indebted to the many members of the F# community (both open source and within Microsoft) who have grown F# from an academic project to a thriving commercial and scientific language with a broad-based ecosystem. Every commit counts!

I'd like to thank everyone at Apress who contributed to this book, notably James DeWolf, Melissa Maldonado, and Douglas Pundick. Thank you for your patience, professionalism, and support.

Finally, huge appreciation to my wife, Val, for her unstinting support in everything I do; and to my children, Matt and Kate, both of them frankly more talented developers than I, for keeping me on my toes.

Foreword

That moment. That secret weapon moment. That I-got-lucky-and-came-across-something-that-helped-me-win-the-game moment. This might just be it.

This book teaches you all you need to know to get going with F#, an open source, cross-platform programming language suitable for just about every kind of programming and data processing task. And yes, learning F# is like learning a secret weapon. Wielding it, you will find yourself doing things previously beyond your limits: it will empower you, and with it you will do great things.

Why do I know this? Because I have seen the authors of this book, Kit Eason and Robert Pickering, learn F# and wield it powerfully and effectively in job after job. I've seen how it has changed the way they think about programming. In this book, they share what they have learned with you, and they want to help you learn and use this secret weapon too.

No programming language is a "silver bullet;" F# won't magically slay dragons in and of itself. However, F# does an excellent job of removing the incidental complexity that bedevils so much of programming. Take one example: the pervasive use of "null" values in languages such as C# and Java. In one real-world case study by Simon Cousins, using F# reduced the incidence of null checks by 200x in a transaction processing engine, with over 3,000 extra null checks in the fully object-oriented application in question. This is no minor thing: to continue the secret weapon analogy, a null check is like a potential weapon failure at a critical moment, a gun-jam. At any one of these 3,000 code points, that application was at risk of failing. It is simply better–and less risky–to fight your battles with a weapon that doesn't jam. F# reduces the number of potential failure points, and does so systematically. Some argue that incidental complexity is not important. This is wrong: removing incidental complexity is the first and most essential step you can take to becoming a more productive programmer, since it liberates you to address the real heart of programming. Learning F# will help you do this.

F# is known as a "functional-first" programming language, meaning you use simple functional programming as the first way to solve most problems. This is simple and easy, and lets you solve most programming problems with ease. F# programming is, however, pragmatic. Good software engineering really matters in F#: we care about code quality, naming, documentation, and good design. This book will show you all of these things. It will also teach you how to use F# for data access, web programming, parallel programming, and a myriad of other practical tasks. Finally, it will show you how to contribute back to the F# community through open source packages.

Take the moment and learn F#, and don't look back. Thousands of people are enjoying this language daily, and with more efficient, expressive coding, and higher productivity. And not just productivity, but delight and happiness in craftsmanship as well.

—Don Syme
F# Language Designer and F# Community Contributor

CHAPTER 1

Getting Started

This introductory chapter will address some of the major questions you may have about F# and functional programming.

What Is Functional Programming?

Functional programming (FP) is the oldest of the three major programming paradigms. The first FP language, IPL, was invented in 1955, about a year before FORTRAN. The second, Lisp, was invented in 1958, a year before COBOL. Both Fortran and Cobol are imperative (or procedural) languages, and their immediate success in scientific and business computing made imperative programming the dominant paradigm for more than 30 years. The rise of the object-oriented (OO) paradigm in the 1970s and the gradual maturing of OO languages ever since have made OO programming the most popular paradigm today.

Since the 1950s there has been vigorous and continual development of powerful FP languages—SML, Objective Caml (OCaml), APL, and Clean, among others—and FP-like languages—Erlang, Lisp, and Haskell being the most successful for real-world applications. However, FP remained a primarily academic pursuit until recently. The early commercial success of imperative languages made it the dominant paradigm for decades. Object-oriented languages gained broad acceptance only when enterprises recognized the need for more sophisticated computing solutions. Today, the promise of FP is finally being realized to solve even more complex problems—as well as the simpler ones.

Pure functional programming views all programs as collections of functions that accept arguments and return values. Unlike imperative and object-oriented programming, it allows no side effects and uses recursion instead of loops for iteration. The functions in a functional program are very much like mathematical functions because they do not change the state of the program. In the simplest terms, once a value is assigned to an identifier, it never changes; functions do not alter parameter values; and the results that functions return are completely new values. In typical underlying implementations, once a value is assigned to an area in memory, it does not change. To create results, functions copy values and then change the copies, leaving the original values free to be used by other functions and eventually to be thrown away when no longer needed. (This is where the idea of garbage collection originated.)

The mathematical basis for pure functional programming is elegant, and FP therefore provides beautiful, succinct solutions for many computing problems. That said, the stateless and recursive nature of FP can make it seem harder to apply for many common programming tasks. However, one of F#'s great strengths is that you can use multiple paradigms and mix them to solve problems in the way you find most convenient.

Electronic supplementary material The online version of this chapter (doi:10.1007/978-1-4842-1374-2_1) contains supplementary material, which is available to authorized users.

R. Pickering and K. Eason, *Beginning F# 4.0*, DOI 10.1007/978-1-4842-1374-2_1

Why Is Functional Programming Important?

When people think of functional programming, they often view its statelessness as a fatal flaw without considering its advantages. One could argue that since an imperative program is often 90 percent assignment and since a functional program has no assignment, a functional program could be 90 percent shorter. However, not many people are convinced by such arguments or attracted to the ascetic world of stateless recursive programming, as John Hughes pointed out in his classic paper "Why Functional Programming Matters."

> *The functional programmer sounds rather like a medieval monk, denying himself the pleasures of life in the hope that it will make him virtuous.*
>
> John Hughes, Chalmers University of Technology
> (www.cse.chalmers.se/~rjmh/Papers/whyfp.html)

To see the advantages of functional programming, you must look at what FP permits rather than what it prohibits. For example, functional programming allows you to treat functions themselves as values and pass them to other functions. This might not seem all that important at first glance, but its implications are extraordinary. Eliminating the distinction between data and functions means that many problems can be more naturally solved. Functional programs can be shorter and more modular than corresponding imperative and object-oriented programs.

In addition to treating functions as values, functional languages offer other features that borrow from mathematics and are not commonly found in imperative languages. For example, functional programming languages often offer curried functions, where arguments can be passed to a function one at a time and, if all arguments are not given, the result is a residual function waiting for the rest of its parameters. It's also common for functional languages to offer type systems with much better power-to-weight ratios, providing more performance and correctness for less effort.

Further, a function might return multiple values, and the calling function is free to consume them as it likes. We'll discuss these ideas, along with many more, in detail and with plenty of examples in Chapter 3.

What Is F#?

Functional programming is the best approach to solving many thorny computing problems, but pure FP isn't suitable for general-purpose programming. So FP languages have gradually embraced aspects of the imperative and OO paradigms, remaining true to the FP model but incorporating features needed to easily write any kind of program. F# is a natural successor on this path. It is also much more than just an FP language.

Some of the most popular functional languages, including OCaml, Haskell, Lisp, and Scheme, have traditionally been implemented using custom runtimes, which leads to problems such as lack of interoperability. F# is a general-purpose programming language for .NET (a general-purpose runtime) and for .NET's open source counterparts. It smoothly integrates all three major programming paradigms. With F#, you can choose whichever approach works best to solve problems in the most effective way. You can do pure FP if you're a purist, but you can easily combine functional, imperative, and object-oriented styles in the same program and exploit the strengths of each. Like other typed functional languages, F# is strongly typed but also uses inferred typing, so programmers don't need to spend time explicitly specifying types unless an ambiguity exists. Further, F# seamlessly integrates with the .NET Framework Base Class Library (BCL). Using the BCL in F# is as simple as using it in C# or Visual Basic (and maybe even simpler).

F# was modelled on OCaml, a successful object-oriented FP language, and then tweaked and extended to mesh well technically and philosophically with .NET. It fully embraces .NET and enables users to do everything that .NET allows. The F# compiler can compile for all implementations of the Common Language Infrastructure (CLI) and it supports .NET generics without changing any code. The F# compiler not only

produces executables for any CLI but can also run on any environment that has a CLI, which means F# is not limited to Windows but can run on Linux, Apple Mac OS X, Apple iOS, FreeBSD, and Android. (Chapter 2 covers some routes to using F# on these diverse platforms.)

The F# compiler is distributed with Visual Studio 2015: you simply need to select the F# option during installation. It is also available in Xamarin Studio (http://xamarin.com/studio). Although there are commercial versions of Visual Studio and Xamarin Studio, in both cases the free edition fully supports F#. Alternatively, Ionide (http://ionide.io) provides an F# IDE for the Atom and Visual Studio Code editors. You can also download and integrate F# with other editors and IDEs such as Emacs and Vim. F# supports IntelliSense expression completion and automatic expression checking. It also gives tool tips to show what types have been inferred for expressions. Programmers often comment that this really helps bring the language to life.

F# was first implemented by Dr. Don Syme at Microsoft Research (MSR) in Cambridge. There is an F# team within Microsoft, but the F# compiler itself and most of the tools in its ecosystem are open source, and there is a very vibrant and friendly open source community around the language. Generally, the term "Visual F#" is used when referring to the Microsoft implementation of F# and its integration with Microsoft Visual Studio. The wider term "F#" refers both to the Microsoft implementation and related activity, and to the wide variety of open implementations and tools.

Although other FP languages run on .NET, F# has established itself as the de facto .NET functional programming language because of the quality of its implementation and its superb integration with .NET and Visual Studio.

No other .NET language is as easy to use and as flexible as F#!

Who Is Using F#?

F# was initially marketed by Microsoft as a language for use in mathematics and the more math-oriented aspects of finance. Unfortunately, the legacy of this initial perception persists today. The reality is that developers are using F# in a remarkable variety of arenas, from quantum computing to scientific instrument control to music. Here, for example, is a testimonial from Tachyus, a startup that creates technology to optimize energy production for the oil and gas industry:

> *F# has allowed us to deliver enterprise-grade software on a rapid, start-up cadence. The F# type system makes it not just a great language for implementation, but also for design: once you've built a domain model with F# types, you've got the compiler's guarantee your model is consistent. Strong typing and functional-first programming have eliminated huge classes of runtime bugs, whereas most languages require voluminous test suites to catch them. The ability to write concise, expressive, and safe code has allowed us to break into a competitive enterprise software market faster than we'd ever expected.*
>
> Paul Orland, Tachyus
> (http://fsharp.org/testimonials/#kaggle-1)

Meanwhile Rachel Reese, at jet.com, a US-based online retail startup, explains their rationale for using F#:

> *We started building two solutions, a C# solution and an F# solution, to see where they would take us. In the end, we chose to stick with the F# path. The main reason: we were able to deliver the same functionality with far less code. This clearly eases maintainability and reduces bugs. If you've been part of the F# community for any length of time, you know that this is a very well known feature of the language and a commonly cited reason to switch to F#.*
>
> Rachel Reese, jet.com
> (http://techgroup.jet.com/blog/2015/03-22-on-how-jet-chose/)

In a very different application area, Anton Tcholakov uses F# to control scientific equipment:

> *Over the past few years, I have worked in a research lab at the University of Warwick where we often develop custom instrumentation for our experiments. Along the way, I've found that good experiment control software presents many interesting challenges: it requires a combination of concurrent control of several external devices and real-time data charting. Cancellation support is essential because experiments can be long-running and you may want to stop them part way through without losing your data. Good error handling and logging are needed to find the causes of failure when it occurs. Sometimes it's necessary to implement computationally demanding signal processing in software as well. Fortunately, I discovered F#, which is the perfect Swiss army knife for many of these problems.*
>
> Anton Tcholakov, University of Warwick
> (https://medium.com/@ant_pt/using-f-for-scientific-instrument-control-b1ef04d20da0#.nokyfi865)

The common thread for almost all F# adoptions is not that the applications are mathematical or functional: it's that the developers wanted to concentrate on the problem to be solved, not on the noise or patterns imposed by any particular language. F# has a great habit of "getting out of the way," as you'll soon discover.

Who Is This Book For?

This book is aimed primarily at IT professionals who want to get up to speed quickly on F#. A working knowledge of the .NET Framework and some knowledge of either C# or Visual Basic would be nice, but it's not necessary. To be comfortable learning F#, all you really need is some experience programming in any language.

Even complete beginners who've never programmed before and are learning F# as their first computer language should find this book very readable. Though it doesn't attempt to teach introductory programming per se, it does carefully present all the important details of F#.

What's Next?

Chapter 2 gives you just enough knowledge about setting up an F# development environment to get you going. Chapters 3, 4, 5, and 6 cover the core F# syntax. I deliberately keep the code simple, because this will give you a better introduction to how the syntax works. Chapter 7 looks at the core libraries distributed with F# to introduce you to their flavor and power, rather than to describe each function in detail.

Then you'll dive into how to use F# for the bread-and-butter problems of the working programmer. Chapter 8 covers data access, Chapter 9 covers concurrency and parallelism, and Chapter 10 covers how applications can take advantage of a network.

The final chapters take you through the topics you really need to know to master F#. Chapter 11 looks at support for creating little languages or domain-specific languages (DSLs), a powerful and very common programming pattern in F#. Chapter 12 explores advanced interoperation issues. Finally, Chapter 13 shows how to use F# type providers to access external data sources in a beautifully fluent way.

CHAPTER 2

How to Obtain, Install, and Use F#

This chapter is designed to get you up and running with F# as quickly as possible. You'll learn how to obtain F#, and how to install it on Windows, Apple OS X, and Linux. There are many ways to edit and build F# projects on the various supported platforms. Rather than cover every permutation, I'll concentrate on the most straightforward route to get you up and running with an IDE on each platform. If you have a favorite editor that I don't cover, such as Emacs, Vim, or Visual Studio Code, rest assured that these programs have bindings to let you use F# effectively. The F# website, `fsharp.org`, has up-to-date information on the various options.

Obtaining F# on Microsoft Windows

The most common IDE in use for Windows .NET projects is Visual Studio. Visual Studio comes in many editions depending on your requirements (and budget!), but all of them from the free Community edition upwards support F#. (F# is not supported by Visual Studio Express.) Search on `www.visualstudio.com` and download the edition that suits you. During the install you can select F# as one of the languages you want to work with. However, if you do not do this, or if you already had Visual Studio installed without having selected F#, don't worry. As soon as you create or open an F# project, an on-demand install will occur.

The next step is to install the Visual F# Power Tools. This is a suite of enhancements for Visual Studio that make it much easier to work with F# code. Although it's sometimes treated as an optional extra, you should definitely install Visual F# Power Tools. Without it you won't have basics such as "Go to definition" and "Refactor Rename." To install Power Tools, run Visual Studio and go to Tools ➤ Extensions and Updates ➤ Online, then search for FSharp Power Tools. You can also find the Power Tools in the Visual Studio Gallery. Once Power Tools is installed, exit from Visual Studio and rerun it as Administrator. You can do so by right-clicking the Visual Studio icon while holding down the Shift key, and selecting "Run as administrator" (see Figure 2-1).

R. Pickering and K. Eason, *Beginning F# 4.0*, DOI 10.1007/978-1-4842-1374-2_2

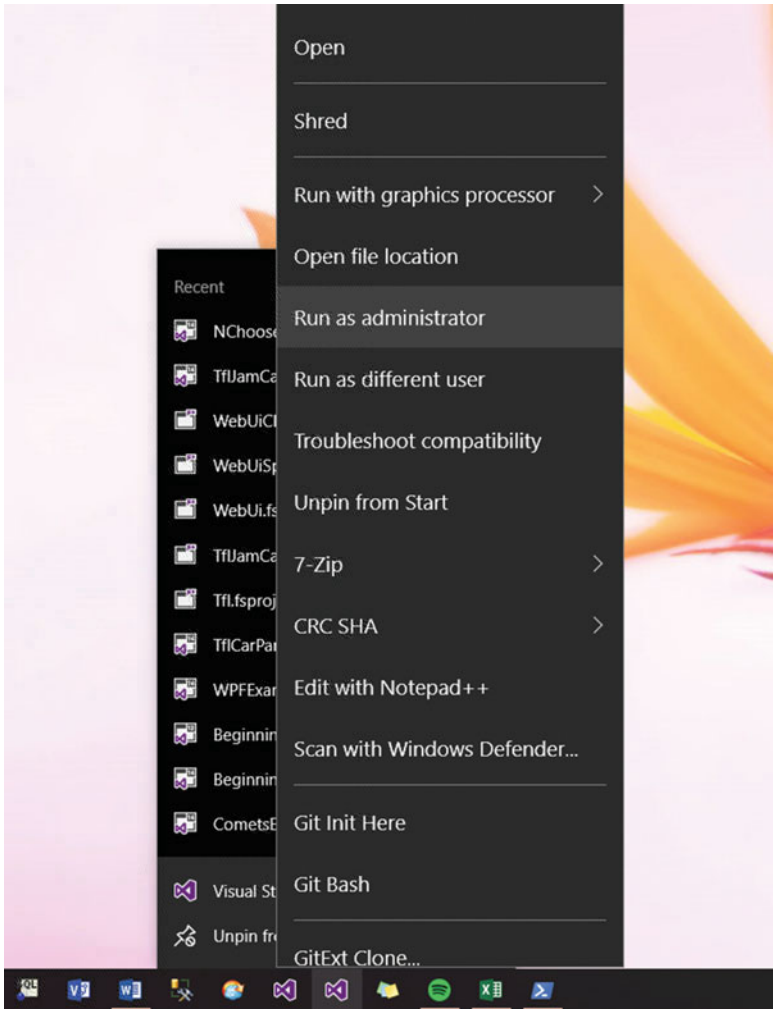

Figure 2-1. *Running Visual Studio as Administrator*

Once Visual Studio is running as Administrator, go to Tools ➤ Options ➤ F# Power Tools ➤ General and turn on all the options (see Figure 2-2). Restart Visual Studio so that the changes come into effect. If any of the Power Tools behaviors aren't compatible with your workflow, you can always come back to this dialogue and turn them off, but most of them are stable, useful, and unobtrusive.

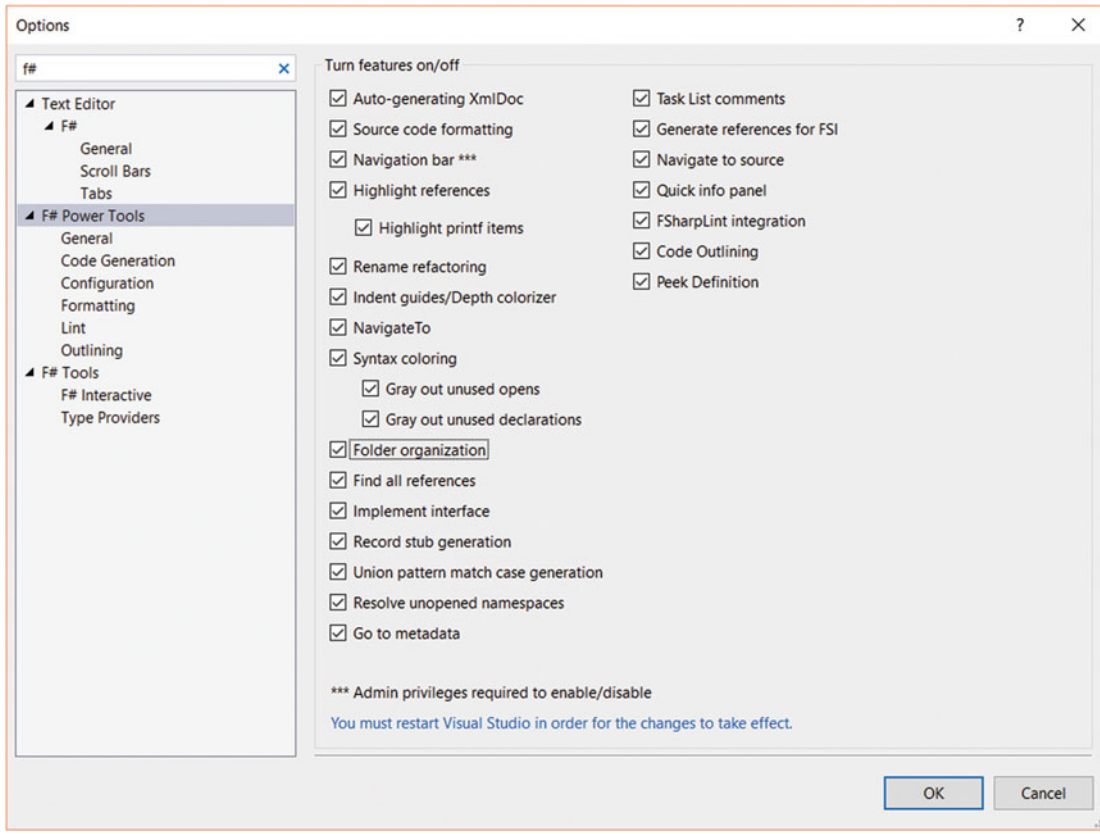

Figure 2-2. *Configuring Visual F# Power Tools*

Verify your setup by going into Visual Studio and selecting File ➤ New ➤ Project. You should be able to find a number of F# project templates under Installed ➤ Templates ➤ Visual F# or Installed ➤ Templates ➤ Other Languages ➤ F# (Figure 2-3).