.NET Test Automation Recipes

A Problem-Solution Approach

James D. McCaffrey

Apress®

.NET Test Automation Recipes: A Problem-Solution Approach

ISBN-13: 978-1-59059-663-0

ISBN-10: 1-59059-663-3

Printed and bound in the United States of America 9 8 7 6 5 4 3 2 1

Lead Editor: Jonathan Hassell
Technical Reviewer: Josh Kelling
Editorial Board: Steve Anglin, Ewan Buckingham, Gary Cornell, Jason Gilmore, Jonathan Gennick, Jonathan Hassell, James Huddleston, Chris Mills, Matthew Moodie, Dominic Shakeshaft, Jim Sumser, Keir Thomas, Matt Wade
Project Manager: Elizabeth Seymour
Copy Edit Manager: Nicole LeClerc
Copy Editor: Julie McNamee
Assistant Production Director: Kari Brooks-Copony
Production Editor: Katie Stence
Compositor: Lynn L'Heureux
Proofreader: Elizabeth Berry
Indexer: Becky Hornak
Cover Designer: Kurt Krames
Manufacturing Director: Tom Debolski

Distributed to the book trade worldwide by Springer-Verlag New York, Inc., 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 `http://www.springeronline.com`.

For information on translations, please contact Apress directly at 2560 Ninth Street, Suite 219, Berkeley, CA 94710. Phone 510-549-5930, fax 510-549-5939, e-mail `info@apress.com`, or visit `http://www.apress.com`.

The source code for this book is available to readers at `http://www.apress.com` in the Source Code section.

Contents at a Glance

Contents

PART 2 Web Application Testing

About the Author

DR. JAMES MCCAFFREY works for Volt Information Sciences, Inc. He holds a doctorate from the University of Southern California, a master's in information systems from Hawaii Pacific University, a bachelor's in mathematics from California State University at Fullerton, and a bachelor's in psychology from the University of California at Irvine. He was a professor at Hawaii Pacific University, and worked as a lead software engineer at Microsoft on key products such as Internet Explorer and MSN Search.

About the Technical Reviewer

JOSH KELLING is a private consultant working in the business software industry. He is formally educated in physics and self-taught as a software developer with nearly 10 years of experience developing business and commercial software using Microsoft technologies. His focus has been primarily on .NET development since it was a beta product. He also enjoys teaching, skiing, hiking, hunting for wild mushrooms, and pool.

Acknowledgments

Many people made this book possible. First and foremost, Jonathan Hassell and Elizabeth Seymour of Apress, Inc. drove the concept, writing, editing, and publication of the entire project. My corporate vice presidents at Volt Information Sciences, Inc., Patrick Walker and Christina Harris, suggested the idea of this book in the first place and supported its development. The lead technical reviewer, Josh Kelling (Kelling Consulting) did a terrific job at finding and correcting my coding mistakes. I'm also grateful to Doug Walter (Microsoft), who contributed significantly to the technical accuracy of this book. Many of the sections of this book are based on a monthly column I write for Microsoft's MSDN Magazine. My editors at MSDN, Joshua Trupin and Stephen Toub, provided me with a lot of advice about writing, without which this book would never have gotten off the ground. And finally, my staff at Volt—Shirley Lin, Lisa Vo Carlson, and Grace Son—supplied indispensable administrative help.

Many Volt software engineers working at Microsoft acted as auxiliary technical and editorial reviewers for this book. Primary technical reviewers include: Evan Kaplan, Steven Fusco, Bruce Ritter, Peter Yan, Ron Starr, Gordon Lippa, Kirk Slota, Joanna Tao, Walter Wittel, Jay Gray, Robert Hopkins, Sam Abolrous, Rich Bixby, Max Guernsey, Larry Briones, Kristin Jaeger, Joe Davis, Andrew Lee, Clint Kreider, Craig Green, Daniel Bedassa, Paul Kwiatkowski, Mark Wilcox, David Blais, Mustafa Al-Hasnawi, David Grossberg, Vladimir Abashyn, Mitchell Harter, Michael Svob, Brandon Lake, David Reynolds, Rob Gilmore, Cyrus Jamula, Ravichandhiran Kolandaiswamy, and Rajkumar Ramasamy.

Secondary technical reviewers include Jerry Frost, Michael Wansley, Vanarasi Antony Swamy, Ted Keith, Chad Fairbanks, Chris Trevino, David Moy, Fuhan Tian, C.J. Eichholz, Stuart Martin, Justice Chang, Funmi Bolonduro, Alemeshet Alemu, Lori Shih, Eric Mattoon, Luke Burtis, Aaron Rodriguez, Ajay Bhat, Carol Snyder, Qiusheng Gao, Haik Babaian, Jonathan Collins, Dinesh Ravva, Josh Silveria, Brian Miller, Gary Roehl, Kender Talylor, Ahlee Ly, Conan Callen, Kathy Davis, and Florentin Ionescu.

Editorial reviewers include Christina Zubelli, Joey Gonzales, Tony Chu, Alan Vandarwarka, Matt Carson, Tim Garner, Michael Klevitsky, Mark Soth, Michael Roshak, Robert Hawkins, Mark McGee, Grace Lou, Reza Sorasi, Abhijeet Shah, April McCready, Creede Lambard, Sean McCallum, Dawn Zhao, Mike Agranov, Victor Araya Cantuarias, Jason Olsan, Igor Bodi, Aldon Schwimmer, Andrea Borning, Norm Warren, Dale Dey, Chad Long, Thom Hokama, Ying Guo, Yong Wang, David Shockley, Allan Lockridge, Prashant Patil, Sunitha Mutnuri, Ping Du, Mark Camp, Abdul Khan, Moss Willow, Madhavi Kandibanda, John Mooney, Filiz Kurban, Jesse Larsen, Jeni Jordan, Chris Rosson, Dean Thomas, Brandon Barela, and Scott Lanphear.

Introduction

What This Book Is About

This book presents practical techniques for writing lightweight software test automation in a .NET environment. If you develop, test, or manage .NET software, you should find this book useful. Before .NET, writing test automation was often as difficult as writing the code for the application under test itself. With .NET, you can write lightweight, custom test automation in a fraction of the time it used to take. By *lightweight automation*, I mean small, dedicated test harness programs that are typically two pages of source code or less in length and take less than two hours to write. The emphasis of this book is on practical techniques that you can use immediately.

Who This Book Is For

This book is intended for software developers, testers, and managers who work with .NET technology. This book assumes you have a basic familiarity with .NET programming but does not make any particular assumptions about your skill level. The examples in this book have been successfully used in seminars where the audience background has ranged from beginning application programmers to advanced systems programmers. The content in this book has also been used in teaching environments where it has proven highly effective as a platform for students who are learning intermediate level .NET programming.

Advantages of Lightweight Test Automation

The automation techniques in this book are intended to complement, not replace, other testing paradigms, such as manual testing, test-driven development, model-based testing, open source test frameworks, commercial test frameworks, and so on. Software test automation, including the techniques in this book, has five advantages over manual testing. We sometimes refer to these automation advantages with the acronym SAPES: test automation has better Speed, Accuracy, Precision, Efficiency, and Skill-Building than manual testing. Additionally, when compared with both open source test frameworks and commercial frameworks, lightweight test automation has the advantage of not requiring you to travel up a rather steep learning curve and perhaps even learning a proprietary scripting language. Compared with commercial test automation frameworks, lightweight test automation is much less expensive and is fully customizable. And compared with open source test frameworks, lightweight automation is more stable in the sense that you have fewer recurring version updates and bug fixes to deal with. But the single most important advantage of lightweight, custom test automation harnesses over commercial and open source test frameworks is subjective—lightweight automation actively encourages and promotes creative testing, whereas commercial and open source frameworks often tend to direct the types of automation you create to the types of tests that are best supported by the framework. The single biggest disadvantage of lightweight test automation is manageability. Because lightweight test harnesses are so easy to write, if you

aren't careful, your testing effort can become overwhelmed by the sheer number of test harnesses, test case data, and test case result files you create. Test process management is outside the scope of this book, but it is a challenging topic you should not underestimate when writing lightweight test automation.

Coding Issues

All the code in this book is written in the C# language. Because of the unifying influence of the underlying .NET Framework, you can refactor the code in this book to Visual Basic .NET without too much trouble if necessary. All the code in this book was tested and ran successfully on both Windows XP Professional (SP2) and Windows Server 2003, and with Visual Studio .NET 2003 (with Framework 1.1) and SQL Server 2000. The code was also tested on Visual Studio 2005 (with Framework 2.0) and SQL Server 2005; however, if you are developing in that environment, you'll have to make a few minor changes. I've coded the examples so that any changes you have to make for VS 2005 and SQL Server 2005 are flagged quickly. I decided that presenting just code for VS 2003 and SQL Server 2000 was a better approach than to sprinkle the book text with many short notes describing the minor development platform differences for VS 2005 and SQL Server 2005. The code in this book is intended strictly for 32-bit systems and has not been tested against 64-bit systems.

If you are new to software test automation, you'll quickly find that coding as a tester is significantly different from coding as a developer. Most of the techniques in this book are coded using a traditional, scripting style, rather than in an object-oriented style. I've found that automation code is easier to understand when written in a scripting style but this is a matter of opinion. Also, most of the code examples are not parameterized or packaged as methods. Again, this is for clarity. Most of the normal error-checking code, such as checking the values of input parameters to methods, is omitted. Error-traps are absolutely essential in production test automation code (after all, you are expecting to find errors) but error-checking code is often three or four times the size of the core code being checked. The code in this book is specifically designed for you to modify, which includes wrapping into methods, adding error-checks, incorporating into other test frameworks, and encapsulating into utility classes and libraries.

Most of the chapters in this book present dummy applications to test against. By design, these dummy applications are not examples of good coding style, and these applications under test often contain deliberate errors. This keeps the size of the dummy applications small and also simulates the unrefined nature of an application's state during the development process. For example, I generally use default control names such as `textBox1` rather than use descriptive names, I keep local variable names short (such as `s` for a string variable), I sometimes place multiple statements on the same line, and so forth. I've actually left a few minor "severity 4" bugs (typographical errors) in the screenshots in this book; you might enjoy looking for them.

In most cases, I've tried to be as accurate as possible with my terminology. For example, I use the term *method* when dealing with a subroutine that is a field/member in a C# class, and I use the term *function* when referring to a C++ subroutine in a Win32 API library. However, I make exceptions when I feel that a slightly incorrect term is more understandable or readable. For example, I sometimes use the term *string variable* instead of the more accurate *string object* when referring to a C# string type item.

This book uses a problem-solution structure. This approach has the advantage of organizing various test automation tasks in a convenient way. But to keep the size of the book reasonable, most of the solutions are not complete, standalone blocks of code. This means

that I often do not declare variables, explicitly discuss the namespaces and project references used in the solution, and so on. Many of the solutions in a chapter refer to other solutions within the same chapter, so you'll have to make reasonable assumptions about dependencies and how to turn the solution code into complete test harnesses. To assist you in understanding how the sections of a chapter work together, the last section of every chapter presents a complete, standalone program.

Contents of This Book

In most computer science books, the contents of the book are summarized in the introduction. I will forego that practice and say instead that the best way to get a feel for what is contained in this book is to scan the table of contents; I know that's what I always do. That said however, let me mention four specific topics in this book that have generated particular interest among my colleagues. Chapter 1, "API Testing," is in many ways the most fundamental type of all software testing. If you are new to software testing, you will not only learn useful testing techniques, but you'll also learn many of the basic principles of software testing. Chapter 3, "Windows-Based UI Testing," presents powerful techniques to manipulate an application through its user interface. Even software testers with many years of experience are surprised at how easy UI test automation is using .NET and the techniques in that chapter. Chapter 5, "Request-Response Testing," demonstrates the basic techniques to test any Web-based application. Web developers and testers are frequently surprised at how powerful these techniques are in a .NET environment. Chapter 10, "Combinations and Permutations," gives you the tools you need to programmatically generate test cases that take into account all combinations and rearrangements of input values. Both new and experienced testers have commented that combinatorics with .NET makes test case generation significantly more efficient than previously.

Using the Code in This Book

This book is intended to provide practical help for you in developing and testing software. This means that, within reason, you may use the code in this book in your systems and documentation. Obvious exceptions include situations where you are reproducing a significant portion of the code in this book on a Web site or magazine article, or using examples in a conference talk, and so on. Most authors, including me, appreciate citations if you use examples from their book in a paper or article. All code is provided without warranty of any kind.