The Minimum You Need to Know

About Mono and Qt

Roland Hughes

Logikal Solutions

Printed and bound in the United States of America

ISBN-13 978-0-9823580-8-5

This book was published by Logikal Solutions for the author. Neither Logikal Solutions nor the author shall be held responsible for any damage, claim, or expense incurred by a user of this book and the contents presented within or provided for download at http://www.theminimumyouneedtoknow.com.

Trademark	Owner
Zinc Application Framework	Professional Software Associates, Inc.

Additional Books by Roland Hughes

You can always find the latest information about this book series by visiting http://www.theminimumyouneedtoknow.com. Information regarding upcoming and out-of-print books may be found by visiting http://www.logikalsolutions.com and clicking the "upcoming and out of print books" link. At the time of this writing, Logikal Solutions and Roland Hughes offer the following books either in print or as eBooks.

The Minimum You Need to Know About Logic to Work in IT
ISBN-13 978-0-9770866-2-7
Pages: 154

Covers logic, flowcharting, and pseudocode. If you only learned OOP, you really need to read this book first.

The Minimum You Need to Know To Be an OpenVMS Application Developer
ISBN-13 978-0-9770866-0-3
Pages: 795
Includes CD-ROM

Covers DCL, logicals, symbols, command procedures, BASIC, COBOL, FORTRAN, C/C++, Mysql, RDB, MMS, FMS, RMS indexed files, CMS, VMSPhone, VMSMAIL, LSE, TPU, EDT, SORT, and many other topics. This book was handed out by HP at a technical boot camp because the OpenVMS engineering team thought so highly of it.

The Minimum You Need to Know About Java on OpenVMS, Volume 1
ISBN-13 978-0-9770866-1-0
Pages: 352
Includes CD-ROM

Covers using Java with FMS and RMS indexed files. There is a lot of JNI coding. We also cover calling OpenVMS library routines, building with MMS and storing source in CMS.

The Minimum You Need to Know About Service Oriented Architecture
ISBN-13 978-0-9770866-6-5
Pages: 370
The National Best Books 2008 Award Winner – Business: Technology/Computer

Covers accessing your MySQL, RDB, and RMS indexed file data silos via Java and port services from a Linux or other PC front end. Also covers design and development of ACMS back end systems for guaranteed execution applications.

The Minimum You Need to Know About Java and xBaseJ
ISBN-13 978-0-9823580-3-0
Pages: 186

This book is available only as a free PDF. It's source files are available on SourceForge and on the book's Web page. This book is meant to provide a much needed tutorial for the Open Source xBaseJ library. If you have some fundamental Java skills this book can have you developing your own xBaseJ applications in a matter of hours. The xBaseJ library was used by the author to create the FuelSurcharge project on SourceForge.

The Minimum You Need to Know About Qt and Databases
ISBN-13 978-0-9823580-5-4
Pages: 474

This book is meant to provide a much needed tutorial on using Qt with various IDEs and database tools. Most of the books on the market do a great job showing you most of the GUI features of Qt, but are sadly lacking when it comes to explaining how to use Qt with databases. It is not uncommon to find at most one chapter in your favorite Qt book and to learn it is woefully inadequate for the task at hand. This book is meant to solve those problems. While Qt attempts to shield you from many underlying database differences, this book will prove that it is only partially successful and you still have to design your application around the limitations of your chosen database.

Infinite Exposure
ISBN-13 978-0-9770866-9-6
Pages: 471

A novel about how the offshoring of IT jobs and data centers will lead to the largest terrorist attack the free world has ever seen and ultimately to nuclear war.

There are a number of reviews of this book available on-line. The first 18 chapters are also being given away for free at BookHabit, ShortCovers, Sony's eBook store, and many other places. If you can't decide you like it after the first 18 chapters, Roland really doesn't want to do business with you.

John Smith: Last Known Survivor of the Microsoft Wars
ISBN-13 978-0-9823580-6-1

A post apocalypse novel which some might consider the sequel to Infinite Exposure. If I do my job correctly, it should piss just about everyone off.

Source Code License

This book is being offered to the public freely, as is the source code. Please leave comments about the source of origin in place when incorporating any portion of the code into your own projects or products.

Users of the source code contained within this book agree to hold harmless both the author and the publisher for any errors, omissions, losses, or other financial consequences which result from the use of said code. This software is provided "as is" with no warranty of any kind expressed or implied.

Visit http://www.theminimumyouneedtoknow.com to find a download link if you don't want to retype or cut and paste code from this book into your own text editor.

Table of contents

Introduction

Why This Book?

I had to ask myself this very question when starting to write this book. Actually, an author should really ask that question about every book they write but it was particularly important for this book. There are oceans of books on C#. There are ponds of books covering Mono. There are various Web sites devoted to the promotion and support of Mono and its various Open Source off-shoots. Why should *I* write a book on it?

Believe it or not, I came up with a pretty good answer. Most of the books out there fall into one of the following categories:

- Rah-Rah Microsoft, don't bother teaching anything
- Language syntax bible which might be great for reference, but not for teaching.
- Me-Too regurgitation of the stuff available for free on-line by professional authors who have to crank out at least six books per year to earn a living.

Those of you who have followed this series know that my books never fall into those three categories. I've spent over 20 years in the world of IT with most of my years as a real consultant. The difference between someone who calls themselves a consultant and a real consultant can easily be seen on their resumes. Scan back over the past 20+ years and you will see the people who call themselves consultants have worked at no more than three companies. Once in the door they latch on like a parasite and try to become indispensable so they can milk the site for all it's worth. A real consultant will rarely spend more than 18 months at any single client site, but they will be called back to sites multiple times over the years. In short, a real consultant travels from client site to client site picking up information about additional ways of doing things rather than doing things the same way for 20+ years.

Why is the definition of a consultant important? That's simple. Most of the books out there on Mono/C# weren't written by real consultants. They were written by people who call themselves consultants. While many will claim to have over 20 years of experience in the field, most of them will have one year of experience repeated 20 times rather than have traveled all over and done many things. You don't have to dig deep in their backgrounds to find out if the book you are about to buy is a waste of cash or not, simply flip to the section on connecting to a database. (That's right, there will be only one and it might not even consume an entire chapter.) Flip through the beginning and very end of that section. *At any point do they point out directly connecting an application across the Internet or from outside of a secured internal network is a really bad idea?* If the answer to that question is "yes", the book you are looking at was written by a veteran; "no" means your book was written by a commercial author or someone with less than one year of real experience no matter how many years it was repeated.

Yes, we will cover that topic yet again because I seem to be the only one writing about it and the rest of the IT world seems to be appearing on "60 Minutes" trying to explain why you shouldn't go to prison for writing the system which lead to this month's largest identity theft.

Why this book? Because I lived when this happened and Microsoft sure as hell didn't *invent* C#. I'm tired of seeing books anoint Microsoft for *inventing* something they just renamed, bought, or in this case, mostly stole.

Why this book? Because Microsoft is quickly disappearing from desktop and back office environments. Companies which made the mistake of betting on them need someone to show them alternatives *now*, before it is too late.

Why this book? Because most developers cannot learn an OOP tool from either a syntax reference manual or a marketing brochure. I'm going to continue doing what this series does and has been acclaimed for, redeveloping applications with different tools so developers can straddle technologies.

Why OpenSuSE?

It is true that I've used Ubuntu and KUbuntu for other books in this series. While that set of Linux distros may be the most famous, they've taken some shortcuts to target the now disappearing NetBook market which has established a sunset date for Ubuntu as well. There is a real problem with their distribution methodology which is forcing some incredibly bad design decisions into that distro. If you mandate that your initial "live" version fit on a single CD instead of a DVD, you are going to make an awful lot of bad decisions due to that initial bad decision.

OpenSuSe comes on a DVD, or a thumb drive, and in other methods which allow for much more than just a single CD's worth of storage. Because of this you don't see a different release of OpenSuSE for each potential Linux desktop. Not only are these desktops all available on your initial installation media, *they are actually tested together*. I cannot tell you the number of times I was using one flavor of Ubuntu but needed something written for one of the other flavors to make my life easier, then had lots of things stop working correctly once the other 700MEG was downloaded to support the other desktop. When you posted a support issue, you were quite honestly told "don't do that" and the issue was closed.

Another reason we are using OpenSuSE instead of Ubuntu is that the support people actually care. I have wasted hundreds of hours in email arguments with the supposed powers that be in Ubuntu land pointing out massive technical flaws due primarily to their genetic inability to grasp technical topics. Quite recently one of these cerebral giants told me they were going to simply close without comment bug reports I was filing about their forcing all things Qt to use database plug-ins instead of compiled in connections "because plug-ins were more secure." The English language simply does not have the ability to accurately describe the depth of that stupidity, and this was the person in charge! Definitely promoted to their level of inability! I didn't ask them, but they sure sounded like an MBA from Keller! That level of incompetence is the norm for graduates of that school in my experience.

For those of you who do not comprehend the security hole Open Source plug-ins create, imagine if some Russian mafia family, or Chinese Triad, or some other organized crime entity takes it upon themselves to write a custom version of each plug-in for each release. The custom version adds the “feature” of logging all table layouts and I/O to the “temp” directory, then queuing a task at program exit to transmit this information to a server of theirs somewhere on the Web.

The crime family then “donate” a lot of time to various Open Source projects loading these plug-ins into the distributions. Nobody notices any difference because they have only added a logging and delayed transmit feature. The crime family ends up with every record written to every database using those plug-ins on every system containing them. Yes, they end up with a lot of chicken recipies and music catalogs, but, eventually they start getting the records for various Open Source competitors to Quicken, QuickBooks, Seible, etc. Now they have current balance and account information and possibly even identies. All that is necessary to stop this from happening is to statically compile in the database connectivity. Organized crime must then infiltrate each Open Source package they want data from instead of any package that could be installed on your computer. Points of failure now reduced from billions to one.

In short, we are using OpenSuSE because it has matured and, regrettably, it has close ties to Microsoft. You might remember that Novell cut a deal which let Microsoft sell SuSE servers and gave Novell a big chunk of cash. Part of that cash went to Novell producing a bunch of classes providing Linux services to Mono Core libraries. I'm currently using a 64-bit version of OpenSuSE 11.4 both to do this development and to write this book. It is by far the stablest and best thought out version of Linux I have ever used. It doesn't currently have the “fan base” of Ubuntu, but the “shine” of Ubuntu's initial “just works” marketing campaign is starting to wear off now that many people are encountering things which don't work.

Speaking of things which don't work, I tried to make it an entire month with Fedora 14 before moving to the latest OpenSuSE. I realize Fedora has its fans, but, nothing worked that I really needed and Fedora's desire to force my disks into some LVM

configuration really irked me.

If you want a more believable explanation, how about this? Ubuntu has begun ignoring its user base in order to cling to some early marketing decisions. Fedora 14 is a train wreck. OpenSuSE was the last end user desktop on this list: http://www.ultraedit.com/products/uex.html I have seen that exact same list for other commercial products as well. Yes, there will always be hundreds of Linux distros, but there will only be a handful companies spending actual dollars for commercial products. At this point in time it looks like OpenSuSE will be the survivor three years down the road.

How C# Really Came About

A long time ago there was a company named Sun Microsystems (now owned by Oracle Corporation.) Like many computer companies of the day it had proprietary computer hardware and operating systems. Unlike most computer companies of its day, it had direct involvement with embedded systems and to some extent video games. The embedded systems market was becoming an expensive pauper's child and some high minded individuals at Sun looked to solve this problem. Indeed, they could not achieve their vision of Internet connected appliances unless they could create a prepackaged and ready to program embedded computer for only a couple of dollars.

Embedded systems are so prevalent in today's world that most of you cannot imagine life without them. That coffee maker which you program to start brewing at some point in the morning has an embedded system in it. That DVD player and the remote control for it? Yep, both of them. If you happen to drive a car which was built-in the past ten years it has somewhere between dozens and hundreds of self contained embedded systems on board.

Let's consider for a moment the DVD player that retails for $40 in the U.S. That thing most likely has a production cost of less than $8, including all packaging. How do I figure that? Most retailers purchase inventory via a distributor, not direct from the manufacturer and both of them will expect at least a $10 cut of that $40 list which leaves

about $12 per unit profit for the manufacturer, less shipping costs, warranty claims, and returns.

Just how much do you think they actually spent on the embedded systems in both the player and the remote control? It was pennies. Some little geek had to spend months of their lives burning ROMs, continually shrinking the footprint of an "application" which continually was getting new features added. He/she wasn't coding with a high level language, but working in the assembler language for the chosen CPU trying to squeeze everything to the cheapest CPU and smallest chip count. If someone in purchasing got a "better deal" on a different CPU, the little geek had to start over.

The original concept was an embedded Java processor. Almost everything on a single chip on a little card with connector pins for serial, parallel, network, and possibly even some display type device (normally LED connections.) The unit would come in several different flavors and sizes. It would contain both flash ROM and actual memory so that it could be updated on the fly. *The production cost would be driven down by selling billions of units.*

Yes, the current embedded system model would still be cheaper, but this model would be both cheap and fast. Code would be transportable between devices so instead of taking 6-10 months to develop the next version programmers could have it to the testing group in 6-10 days. The presence of Internet capabilities, not to mention all of that RAM meant features which were once science fiction could easily be implemented. Rather than simply displaying some cryptic flashing lights or a fault code, your washing machine could signal the manufacturer it needed repair and what parts it needed. It would send along its serial number and the message would actually get to the service department of the vendor who sold it and the service department would call *you* to schedule an appointment.

Laugh all you want about my washing machine description. Some high end luxury cars already have this feature. There have even been television commercial pitching this feature to consumers. Just how do you think OnStar works? You've all seen those TV commercials where emergency personnel are dispatched automatically when the vehicle determines it was involved in a collision with airbags deployed. There is a series of embedded systems all communicating together making that happen.

While it is true we have yet to get such a washing machine, we have gotten other science fiction products. With the embedded system in a DVR you can now pause live TV and start it back up again. Your DVR can even connect to a video on demand service and download movies for you to watch. Do I even need to mention all of the things your cell phone can do now, besides make a phone call?

Of course, the first cut of Java, which was to be the language processed by the virtual machine in the embedded device, got written on Unix, then Linux, then a lot of other platforms. Sun wanted to test it and get it adopted, so they made it free for downloading. Specifications were published for JVMs so anyone could develop a JVM for their favorite platform, but Sun also developed the JVM for many platforms. Of course, it was the built-in Internet classes and the ability to work within Web browsers that caused Java development to go off on a major tangent adding feature after feature.

Eventually, bloat became so bad that multiple editions of Java had to be released: