

COBOL and Visual Basic on .NET: A Guide for the Reformed Mainframe Programmer

CHRIS RICHARDSON

Apress™

COBOL and Visual Basic on .NET: A Guide for the Reformed Mainframe Programmer
Copyright © 2003 by Chris Richardson

All rights reserved. No part of this work may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage or retrieval system, without the prior written permission of the copyright owner and the publisher.

ISBN (pbk): 1-59059-048-1

Printed and bound in the United States of America 12345678910

Trademarked names may appear in this book. Rather than use a trademark symbol with every occurrence of a trademarked name, we use the names only in an editorial fashion and to the benefit of the trademark owner, with no intention of infringement of the trademark.

Technical Reviewer: Hung Tran

Editorial Directors: Dan Appleman, Gary Cornell, Simon Hayes, Martin Streicher, Karen Watterson, John Zukowski

Assistant Publisher: Grace Wong

Project Manager: Sofia Marchant

Developmental Editor: Valerie Perry

Copy Editor: Nicole LeClerc

Compositor: Argosy Publishing

Artist: Faith Bradford

Indexer: Kevin Broccoli

Cover Designer: Kurt Krames

Production Manager: Kari Brooks

Manufacturing Manager: Tom Debolski

Distributed to the book trade in the United States by Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY, 10010 and outside the United States by Springer-Verlag GmbH & Co. KG, Tiergartenstr. 17, 69112 Heidelberg, Germany.

In the United States, phone 1-800-SPRINGER, email orders@springer-ny.com, or visit <http://www.springer-ny.com>. Outside the United States, fax +49 6221 345229, email orders@springer.de, or visit <http://www.springer.de>.

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, email info@apress.com, or visit <http://www.apress.com>.

The information in this book is distributed on an “as is” basis, without warranty. Although every precaution has been taken in the preparation of this work, neither the author(s) nor Apress shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the information contained in this work.

The source code for this book is available to readers at <http://www.apress.com> in the Downloads section.

About the Foreword Writer

Jerome Garfunkel is an international consultant and specialist in learning systems design. In the international commercial information technology (IT) community, he is recognized as one of the leading authorities in the field of programming languages and international computer standards. He has served as a senior technical advisor to the U.S. Department of Commerce. In addition, he sat on several American and international IT industry committees and has represented the United States in both the international and domestic IT standardization community in the ANSI Standards Planning and Requirements Committee, the ANSI Programming Language Study Group, the International Committee on Programming Language Guidelines, the American COBOL Committee, the International COBOL Committee, the CODASYL COBOL Committee, and the Object-Oriented COBOL Committee.

Jerome Garfunkel is a lecturer, an author, a consultant, an educator, an actor, a calligrapher, and a passionate motorcycle rider. As an educator and technologist, he lectures about leading-edge technologies such as the integration of legacy systems with Web-based services. Mr. Garfunkel was awarded the degree of Doctorate, Honoris Causa in Technology from De Montfort University in Leicester, England, for his lifetime contributions to the software engineering community. His collection of technical papers, memoranda, and notes is housed in the Charles Babbage Institute in Minnesota, for historical research. Included is the large body of his writings appearing in books, IT journals, magazines, and newspapers around the world.

Foreword

AS RODNEY DANGERFIELD says, “I don’t get no respect,” so too COBOL doesn’t “get no respect.” The COBOL language has been underappreciated, disrespected, criticized, and bashed for much of its existence. It is generally perceived as an inefficient, verbose application development tool, irrelevant to modern software development methodologies. Nothing could be further from the truth. The very existence of Christopher Richardson’s book, *COBOL and Visual Basic on .NET: A Guide for the Reformed Mainframe Programmer*, is a testament to COBOL’s continued relevance.

COBOL Bashing

The timing of Mr. Richardson’s book couldn’t be better. The fourth official publication of the COBOL standard (COBOL o2002¹) is pending as of this writing. It follows COBOL 68, COBOL 74, and COBOL 85. If you consider the Intrinsic Functions Addendum published in o1989, the new COBOL o2002 actually marks the fifth official release of standard COBOL.

The disrespect for the COBOL programming language, which is encouraged in many universities, is not new. The earliest gesture of “COBOL bashing” lies in the Computer Museum in Boston. It is a COBOL tombstone. Soon after the Conference on Data Systems Languages (CODASYL) committee was formed in o1959, a COBOL tombstone was given as a (gag) gift from some of the CODASYL COBOL committee members to the chairperson of CODASYL. It was meant to express their lack of confidence that this new “common business language” (CBL, later COBOL) would be used in the IT industry for very long. As it turned out, the COBOL language, defined by the CODASYL “short-range” committee and first published in o1960, has been an important part of the IT industry for 43 years—and still counting. Perhaps it is natural to presume that anything in the IT industry dating back to the o1950s/o1960s must be obsolete and irrelevant in the twenty-first century. The mistake in this logic is that the COBOL language (features and syntax) has evolved dramatically over its lifetime. I like to paraphrase an old

1. The use of five digits years (e.g., o2002) throughout this foreword is done so as not to be shortsighted in the year 9999 prior to the turn of the eleventh millennium. Sure, you might say, “Isn’t it a bit early to be worrying about the Y10K problem?” But as we now know in retrospect, this is the same kind of thinking over the past three to four decades that led us to the Y2K crisis. Well, maybe I’m being a bit too cautious. If most enterprise applications are developed in COBOL for the next 8,000 years, I suppose the Y10K crisis, as the Y2K crisis, will turn out to be a “noncrisis.” (Wink!)

General Motors commercial: “COBOL o2002 is not your father’s/mother’s COBOL.”

Although the COBOL tombstone may have been the earliest example of COBOL bashing, it certainly wasn’t the last. The annals of the IT industry are filled with other instances of public disrespect for COBOL. Dutch computer professor Edsger Dijkstra wrote in 1982, “The teaching of COBOL should be made a criminal offense!” For years, I have been speaking at universities around the world. Generally, COBOL has been on the defensive in these academic environments. Few people in my audiences (teachers included) were/are aware of what the modern COBOL language can do. COBOL has no real technical problems—it has “public relations” problems.

In the early o1980s, COBOL bashing took a turn for the worse. Serious efforts were underway to “kill” COBOL. In preparation for the publication of ISO/ANSI COBOL 85, there were serious legal and lobbying attempts (led by Travelers Insurance and joined by other respectable corporations) to block any new COBOL standardization effort. Some wanted to freeze the COBOL language as described in the ANSI COBOL 74 standard. Others wanted to roll back the COBOL standard to its “official” COBOL 68 version. The argument offered by these groups was that it involved a great corporate cost to update their enterprise applications in order for older COBOL programs to compile cleanly in newer COBOL compilers. No one told them that they didn’t need to recompile any programs unless the business application required updating.

This frustration is still felt by many IT departments today. It’s similar to the frustration experienced by many home computer users who object to frequent hardware changes, operating system upgrades, application updates, and the incompatibility issues surrounding all of this “progress.” This is a legitimate dilemma. The ISO/ANSI COBOL committees became very sensitive to any change (addition, modification, deletion) to the COBOL standard that might affect programs written earlier. Still to this day, the (INCITS/ANSI) X3J4 COBOL committee, which is responsible for the technical evolution of COBOL, maintains a special list of new and changed features incorporated in COBOL o2002 that could possibly produce incompatible results with older COBOL programs. Special voting rules were put into effect before “potentially incompatible” features can be added to the COBOL language. COBOL features designated as “obsolete” are required to remain in the new COBOL standard for at least one more iteration before they’re permanently removed from the official COBOL standard. This is done to give the COBOL community ample time (a decade or more) to update the affected programs and remove the obsolete features. COBOL is nonproprietary. The “official” COBOL standard is not owned by any one software manufacturer. It’s developed by a cross-section of IT professionals, both COBOL compiler/tools manufacturers and COBOL users. This requirement of balancing the COBOL committee representation was built into the membership rules of the COBOL development

committees so there would be a broad range of views regarding how COBOL could best serve the business application development community. It isn't accidental, therefore, that COBOL earned its reputation over the years as a solid, dependable application development language that protects its constituency—application developers—from whimsical changes. It is one of COBOL's many strengths.

Few things that were around in the IT industry in 1960 are still around today, except perhaps in museums and in basements. Yet the COBOL language, albeit highly evolved from its origins, remains relevant. The corporate assets represented by the billions (trillions?) of lines of COBOL code still running on commercial computers aren't about to be abandoned. Nor should they be. New tools that help integrate legacy (read: COBOL) systems with PC applications, Web services, new data formats, and protocols have been available since distributive systems emerged years ago. As today's application environment has evolved, .NET for instance, so has COBOL's capability to link to it, merge with it, and interact with it. Stretching the lifespan of an enterprise's legacy systems increases the value and productivity of its IT development staff and of the assets they produce.

Why Has COBOL Endured?

Given the rarity of anything in the IT industry surviving for over 40 years, it is appropriate to ask why has COBOL endured for so long. An underlying mission of COBOL language development during its entire lifespan has been to keep COBOL's syntax and new features relevant to modern application development methodologies and to do so with utmost respect for the large number of COBOL applications still running on computers around the world. The ISO COBOL 2002 language is an evolutionary product. Today's COBOL is the result of much determined work by the various COBOL committees that have contributed to its evolution, among them ISO, INCITS, ANSI, CODASYL, ECMA, and the national COBOL committees represented by individual countries (Netherlands, Canada, France, Japan, United Kingdom, Germany, United States, et al).

Change Is Natural

One thing is certain in the commercial IT world (and in life in general): Change is natural. Business computer systems are digital models of an enterprise and all of its operations. As an enterprise evolves so must its computer business systems (models) evolve. When designing any business application, we must anticipate changes, both corrective and perfective, in those applications. We must identify those parts of the system that will most likely need maintenance in the course of the system's lifetime.

When speaking to new systems designers, I often use an analogy of design techniques employed by the design engineers who build automobiles. It is absurd to think of a new line of cars being built without anticipating one of the basic maintenance chores required of all fuel-driven automobiles: refueling. Imagine for a moment that the car designers (system designers) placed the fuel cap underneath the car, hidden by the transmission. Would we tolerate dismantling our cars' transmissions each time we needed to refuel our cars? Of course not. This refueling (maintenance) chore was made easier by isolating those parts of the car that we need to reach to do this maintenance.

This is very similar to the design criteria that computer system designers must use. Thankfully, the original COBOL language designers knew this. When the COBOL language was first created, the CODASYL committee envisioned that the IT industry was fast changing and that most business systems would likely need to be modified during their lifetime. New hardware and new software development techniques were likely to follow. COBOL applications needed a way to adapt to ever-changing environments without causing chaos inside the enterprise systems development community. The solution was to make COBOL as adaptable as possible and to incorporate, inside the source program itself, whatever environmental documentation was required. This of course resulted in the Environment Division, one of four original Divisions still in the COBOL language. By isolating "all" the environmental dependencies of a COBOL application in one place, it was much easier to transport a COBOL application from one computer brand to another, from one operating system to another, from one database to another.

Today we take this concept for granted. Why should Microsoft Word behave differently on a Macintosh than on a Windows machine? In the late 1950s, however, this was a new concept just gaining popularity. Admiral Grace Hopper's contribution to the "birth" of COBOL is well documented. She was a member of the original CODASYL executive committee. A far greater contribution perhaps was her pioneering effort to develop and promote the concept of third-generation programming languages such as COBOL, Fortran, Algol, and so on. This allowed programmers to use a common, high-level set of instructions (higher than assembler languages). This high-level code was then translated (compiled) into the machine language of the particular hardware on which it would eventually execute. Further, to the point of adaptability, COBOL incorporated the CALL statement early in its development, as well as a COPY library source code facility. Later, the INVOKE statement was added as part of the object-oriented COBOL module.

These and other features in COBOL acknowledged the changing nature of computer business systems and anticipated the need for adaptability in COBOL language syntax. Rather than incorporate new COBOL syntax and data formats (borrowed from other programming languages) to map into every known programming language and database, COBOL simply chose to create flexible methods to interact with applications written in other languages and to pass data between modules effectively. You might say that the COBOL integrated development environment (IDE) is one of the earliest “open source” environments.

Another reason for COBOL's endurance lies in the large number of auxiliary tools available in a COBOL IDE: full-featured application development suites, code generators, software libraries, debuggers, conversion tools, compiler “add-ins,” and so on. The sheer momentum over 43 years from so many programmers producing so many COBOL applications resulted in cottage industries of supplemental software to aid in the task of COBOL application development. No other programming language has such a robust IDE.

The Y2K “Noncrisis”

I have saved for last the most important reason perhaps for COBOL's continuing excellence: the superior maintainability of COBOL applications.

Much of the criticism of the COBOL language within the application development community is aimed at its “wordiness.” It is true that COBOL is verbose. COBOL applications often require more lines of source code to be written than are necessary in applications written in other languages. This was by design. When the original members of the CODASYL COBOL committee set out to define the syntax for this Common Business Oriented Language (COBOL), they intentionally included many clauses, phrases, “noise words,” and so on not to make the job of the development programmer harder, but rather to make the job of the maintenance programmer easier and the results more accurate. COBOL instructions such as “ADD this-weeks-salary TO previous-year-to-date-salary GIVING new-year-to-date-salary” are indeed more verbose than, say, “LET $z = x + y$.” But no one can argue that the meaning/intent of the business logic coded in the former example is much clearer than in the latter example. This is by design. In application development, *clarity, not cleverness, is a virtue*.

COBOL contains syntax for coding its own shortcuts and clever programming techniques if a programmer is so inclined. The COBOL statement `COMPUTE z = x + y` is perfectly valid, but it is antithetical to good COBOL programming practices and is discouraged for all the reasons mentioned previously. Is this really important? You need only reflect on the Y2K “noncrisis” at the turn of the new millennium to determine how important this is. Many people blamed COBOL for the Y2K crisis, pointing to the huge number of legacy (read

again: COBOL) programs still running on computers in o1999 as we prepared for potential disasters when switching over to o2000. This argument is fallacious. As any good application developer knows, the problems created by storing dates using the last two digits of the year (e.g., 85) instead of four digits (e.g., 1985) resulted from shortsighted system design, not from a poor choice of the programming language used. All applications (business applications and others), whether written in COBOL, Fortran, C/C++, or Visual Basic, had to be checked and modified to deal with the change from o1999 to o2000.

It's my firm belief that the Y2K crisis, anticipated by so many, turned out to be a Y2K "noncrisis" specifically because most of those legacy systems were in fact developed with COBOL. Because the COBOL source code is written with so much more clarity than source code written in other languages, the huge task of reviewing all of that legacy source code and making changes when/where necessary was made much easier because of COBOL than nearly everyone had expected. True, the Y2K crisis was a tremendous problem for the IT industry. But the efforts involved to fix the problem were made much easier, not harder, because of COBOL, not in spite of COBOL. In a way, COBOL turned out to be "too good." The clarity associated with COBOL source code made those earlier COBOL applications much more maintainable than anyone had predicted. After all, why discard an application that's performing most of its business functions properly simply because that application needs updating, when modifying the original source code is easier, less expensive, and adds years of productive life to business systems? As a result, the lifespan of legacy systems was stretched much further than people (systems designers) had expected. Is this bad? No, I think not; it's shortsighted perhaps, but it's not bad.

What we learned from the Y2K crisis was that COBOL provides "health insurance" to corporate assets. That is, it's the best way for an enterprise to protect its investment in its IT assets. The same can't be said for applications written in other languages. In many cases, applications written in lower level languages (assembler) or other third-generation languages (C, Pascal, and so on) were simply not worth deciphering to fix Y2K (and other) problems. Instead, many were discarded and replaced by newly written programs (hopefully in COBOL, but probably not).

COBOL and Visual Basic on .NET: A Guide for the Reformed Mainframe Programmer

Mr. Richardson tells the reader in this book, "The world has changed. The .NET Framework and VS .NET is part of that change . . . and so are you." We in the application development community must deal with ever-changing technologies. We're constantly faced with new challenges to keep our programming skills current.

These challenges confront us whether we're integrating legacy systems with modern (integrated Web) technologies or whether we're developing brand-new applications on PCs and/or mainframes incorporating Web services. Can we continue to keep our legacy systems running our enterprises in the midst of these emerging technologies? Or must we abandon those systems and start from scratch to take advantage of these new technologies? Can we "have our cake and eat it too"?

Yes, we can develop modern applications using COBOL, taking advantage of its superior maintainability while incorporating Web-based services into these systems as described in this book. COBOL is alive and well in the twenty-first century and its future is bright. A professor friend of mine from Purdue University, when discussing COBOL's future, likes to tell his students, "You better wear your sunglasses." There's much life still left in our legacy systems due to the myriad of integration tools available to us. And thanks to *COBOL and Visual Basic on .NET: A Guide for the Reformed Mainframe Programmer*, we can understand this new technology from the mainframe programmer's perspective and learn to apply it in our real lives.

Jerome Garfunkel
Woodstock, New York
February, 2003

Introduction

THERE I WAS, AT MICROSOFT'S Professional Developers Conference (PDC) 2001, surrounded by thousands of fellow developers. We were all looking forward to the various events that would take place during that week. Microsoft, with Bill Gates himself in attendance, was soon to launch Windows XP, their newest Windows version. Although this was going to be great, it was really just icing on the cake. What we really came to the PDC for was to hear about .NET (and, of course, to go home with the latest versions of various software titles). I was captivated during the general sessions as several Microsoft knowledge holders spoke rather eloquently about how the development world was changing and how we as developers were in the driver's seat. This all sounded extremely comforting. One Microsoft speaker after another presented and demonstrated some of the newest features of the .NET Framework. Each feature mentioned received loud applause and unanimous cheers. This was a pep rally to end all pep rallies. It really felt good. Until. . .

One particular Microsoft speaker (who will remain nameless) proudly announced the .NET Framework feature the .NET common language runtime (CLR), which allows developers to potentially develop Windows and Web programs in *practically any* language.² He mentioned Visual Basic .NET and the crowd cheered. He mentioned the new language C# and again the crowd cheered. Then it happened. The speaker mentioned (with emphasis) that you would *even* be able to code in COBOL! Yes, the standing-room-only crowd reacted. However, there were no cheers, no applause, and no ovations. Rather, the crowd booed, heckled, and laughed—loudly and continuously. Ouch! I slouched in my seat and pretended to join in.



NOTE Giving Microsoft the benefit of the doubt, I believe the assumptions may have been that any developer attending the PDC event naturally was devoted to developing in the .NET environment. They may have deduced that such developers could not possibly mind such ridicule and humiliation. Perhaps they only erred on the latter. For the record: No ill feelings harbored. All is forgiven.

2. There is the requirement that a .NET version compiler be created to enable additional languages to be used in the .NET platform. Microsoft provides compilers for Visual Basic .NET and C#.

Having coded in this great language called COBOL for many years (both batch and CICS online applications), I concluded that most of the people joining the laugh-in had never actually carried the honorable title of COBOL/CICS/DB2 mainframe programmer. Simply put, they were either jealous or in denial.

Finally, Something for Us

There at the PDC 2001 event, I swore that I would do something for the group of developers interested in .NET that had a foundation similar to mine: mainframe COBOL programming. I wanted to create something that would speak positively to this group of developers. This group of mainframe programmers (those who are seeking *reformation*³) is now facing retraining challenges exacerbated by a void of mainframe-oriented guidance. This is a problem unique to mainframe programmers regardless of which new Common Business Oriented Language (COBOL) they choose to use on the .NET platform: Visual Basic .NET (VB .NET) or NetCOBOL for .NET (courtesy of Fujitsu Software).

This book is my attempt to address this need for mainframe-oriented guidance to tackling .NET. To the 90,000 COBOL programmers⁴ that exist in North America and the uncounted many abroad, hold on to this book. As you strive to join the ranks of .NET developers, you will be hard-pressed to find other books (or Web sites, for that matter) that attempt to provide this type of mainframe-oriented .NET guidance.

My Reasons for Going to the PDC Event

Why was I there at the conference? Was it worth it? Additionally, what is the connection between that event and the topic of this book? Naturally, I had my reasons for going to this mainframe-hostile event:⁵

- Learning about the .NET Framework
- Leveraging previous versions and PC technologies
- Confronting the possibility of starting over

3. Please pardon my attempt at humor—my own chance to poke a little fun at those like myself who either have gone or will go through a career “technology transition.” Yes, I too am a *reformed* mainframe programmer and proud of that fact.

4. According to published estimates of the analyst firm Gartner, there are approximately 90,000 COBOL programmers in North America.

5. OK, maybe using the phrase “mainframe-hostile” to describe the Microsoft PDC event may be an unfair exaggeration. I suggest that you take this lightly and view it as my weak attempt at humor (perhaps returning the favor at most).

Learning About the .NET Framework

I went to the PDC 2001 event to learn about the .NET Framework and all .NET-related technologies. Microsoft's technical team has included many features in the new .NET technology offering. As a result, many of my colleagues are referring to the new .NET Framework as a “revolution” (as opposed to an “evolution”). With the technological revolutions that I have subjected myself to in the past, this was right up my alley. Some people tend to object to the varying amount of marketing that you get at some of the Microsoft-sponsored events. Me, I welcome it. I want to be convinced and persuaded by the professionals.

As it turns out, the PDC 2001 event had a minimal amount of marketing material and a satisfying number of technical demonstrations and explanations. I recommend attending these types of events. Some of them are even free or nominally priced. Especially keep your eyes open for an annual Microsoft-sponsored event called Developer Days. Besides picking up information and free software, you can view this event as a good opportunity to network with fellow developers. Frankly, I had never attended an equivalent type of event while I was on the other side of the fence (in the mainframe world).



TIP There's an annual conference event that focuses on COBOL programming. If you haven't already heard about it, it's called the COBOL Expo. Be sure to check it out at <http://cobolexpo.com/homepage.html>.

Leveraging Previous Versions and PC Technologies

I wanted to hear from the horse's mouth (Microsoft, in this case) why the effort that I had invested into learning previous versions of Visual Basic and other PC technologies had to now be leveraged (or just forgotten, in some cases) in order to learn to develop on the new .NET Framework. I wanted to hear about the many other features (e.g., ASP.NET, ADO.NET, and XML Web services) and find out how I was to use these new technologies when developing tomorrow's applications. At the same time, I wanted to understand and have a healthy perspective on the fact that most of my bleeding-edge PC knowledge (gained over the last 4 years) is now slated to become *legacy* technology.

Allow me to remind you of the momentous transition that I had recently completed from the legacy mainframe technologies to the newer PC technologies. The explanations that were given, the cleanup that Microsoft has done to the “old” Visual Basic version 6.0 language, and the improvements made available by moving away from COM all combine to better enable developers to program

complex business-solutions. Everything that I heard left me feeling very comfortable to join the .NET crowd and retrain. In other words, it was time for me to make a new investment, intellectually speaking.

Confronting the Possibility of Starting Over

I wanted to see the look on the faces of thousands of former experts and gurus while they struggled with the concept of “starting over.” This last point deserves emphasis. The changes in the new .NET Framework and related .NET technologies are so encompassing that many have viewed them as a leveling of the playing field. Because practically everything is new, it is unreasonable for many to claim “expert” status—at least for now. That is right! If there were ever a great time for a mainframe programmer to consider crossing over, now is that time. This is a chance to get in on the ground floor:⁶ .NET version 1.0.



NOTE The absence of .NET experts and gurus is being addressed quickly. I am witnessing the rapid consumption of various Microsoft-sponsored events, countless .NET-related Web sites and, of course, some very well-written books (perhaps such as the one you are currently reading). Many developers have started to code already!

Yes, there were times that I felt inferior mentioning that I started with Visual Basic version 5.0 (many “old-timers” started with Visual Basic version 1.0). Now, I (and many others) will stand shoulder-to-shoulder and start with .NET version 1.0, VB .NET version 1.0, and C# version 1.0. This is a great opportunity! To all of my former mainframe peers, please give this one some serious thought.

What This Book Covers

Simply put, this book is about .NET. It is a guide to approaching the .NET world written from the perspective of a former mainframe programmer and written for other mainframe programmers (soon to become *reformed*⁷). Beyond that, this book is about the new .NET Framework and related .NET technologies

6. I apologize for the multilevel direct-marketing sales-pitch tone.

7. Again, my attempt at a little humor. Please note that the *reformation* in question is a light reference to the career “technology transition” that this book focuses on. Having gone through this “transition” myself, it is much easier to treat the topic lightly. Bear with me. By the end of this book, you (too) will be proud of your reformation (err . . . transition).

(e.g., VS .NET, VB .NET, NetCOBOL for .NET, and ADO.NET). Concepts and considerations for future .NET developers are included as well.

What is unique about this book? I wrote it to serve as a bridge from the mainframe world to the new .NET universe. This book is filled cover-to-cover with mainframe-to-.NET comparisons, analogies, and translations (from the old way to the new way). When applicable, in-depth conceptual discussions are offered. These discussions serve to smooth the edges during the inevitable paradigm shifts that await mainframe programmers headed down the .NET path. I intentionally included a healthy amount of mainframe terminology to create a comfortable learning environment for former mainframe developers. The book assumes the role of a .NET “introductory” text. In other words, this is an entry-level .NET text for advanced mainframe programmers. It will adequately prepare you for further, more detailed learning.

What This Book Does Not Cover

This book is *not* about bashing any technology or company (even when it is deserved). For relational database discussions, I have only included IBM’s DB2 database. Most of the SQL-related discussions are easily transferable to other mainframe DBMSs. The mainframe database system referred to as IMS is not discussed (I was never a big fan of this hierarchical type of data structure). The relational approach won me over long ago. In mainframe circles, I would be considered a DB2 bigot.

When I cover mainframe programming languages, I do not discuss the assembler programming language or any language other than COBOL, for that matter. You will also notice that I have omitted any discussion about available products/compiler that enable visual mainframe-type COBOL development to be done on the PC. With these types of PC-based products, a developer compiles the source code and uploads the binary load module from the PC to the mainframe. Although there are some great products out there, I do not discuss them in this book.

Supplemental References

I wrote this book to serve as your one-stop guide through the maze from the mainframe world to the Windows and Web world of .NET. To prevent this book from being over 1,000 pages in length, I chose to include more topics and at the same time cover less depth per topic. As a supplement, I include Web and text references for your continued learning and retraining effort. You will find these supplemental references located at the end of each chapter in the “To Learn More” section. This

section is divided into the subsections “Books,” “Magazines,” and “Web Sites.” Depending on your particular needs, some topics will deserve a more in-depth, step-by-step drill down. Additionally, for some topics, a list-oriented reference source will be helpful. These book, magazine, and Web references will go a long way toward the goal of filling those gaps. Please take advantage of them.

Technical Requirements

The following list presents the technical requirements for working through the examples and code listings in this book:

- Windows XP Professional operating system. I used Windows XP Professional during the sample application development and when capturing the screen shots. Although you can use Windows 2000 (Professional or Server) for .NET development, I recommend using Windows XP Professional (or newer) for your .NET development.
- Microsoft .NET Framework v.1.0 or v.1.1.
- Microsoft Visual Studio .NET v.1.0 or v.1.1.
- Fujitsu NetCOBOL for .NET v.1.1.
- Enterprise Services/COM+ v.1.5. This version of COM+ is bundled with Windows XP and is expected to be bundled with Windows .NET Server. If you use Windows 2000, and therefore COM+ v.1.0, you will notice some feature differences. This is one of several good reasons for doing your .NET development on Windows XP or newer operating systems.
- Internet Information Services (IIS) v.5 or greater. This is bundled with Windows 2000 and Windows XP Professional and greater. You will need to install IIS manually from your Windows installation source. It is not typically installed by default.
- Microsoft Message Queuing (MSMQ). This is bundled with Windows 2000 and Windows XP Professional and greater. You will need to install MSMQ manually from your Windows installation source. It is not typically installed by default.
- SQL Server 2000 (with a current Service Pack). The sample applications in this book that involve database access use Microsoft's database product, SQL Server 2000. With minimal changes, you can use other relational databases (e.g., Oracle).

- Internet Explorer v.6 or greater. This browser is installed and upgraded as part of your full .NET installation. You may choose to use other browsers. I used Internet Explorer exclusively throughout this book.
- Crystal Decisions's Crystal Reports v.8.5 or greater. A fully functional version of this software product is bundled with your full .NET installation. You will be required to complete a free online registration the first time you use it.

About the Source Code

You can download the code samples that I use throughout the chapters and appendixes from the Downloads section of the Apress Web site (<http://www.apress.com>). The source code is contained in two folders: Folder VS2002 and Folder VS2003.

Folder VS2002 has Visual Basic .NET and COBOL .NET code samples grouped by chapters that were developed using Microsoft's Visual Studio .NET version 1.0 and Fujitsu's NetCOBOL for .NET version 1.1, respectively.

Folder VS2003 has the same Visual Basic .NET samples as in folder VS2002, except that the samples were converted to run under Microsoft's Visual Studio .NET version 1.1 (also known as Microsoft's Visual Studio 2003).

You will notice that Folder VS2003 does not contain any COBOL .NET code samples. At the time of this writing, Fujitsu's current version (1.1) of NetCOBOL for .NET is fully compatible and integrates with Microsoft's Visual Studio .NET version 1.0, but not Microsoft's Visual Studio .NET version 1.1. Be sure to check the NetCOBOL Web site (<http://www.netcobol.com>) for the latest news of a compatible NetCOBOL for .NET release from Fujitsu.

Who This Book Is For

This book was written primarily for intermediate to advanced mainframe programmers who are seeking guidance toward converting/reforming⁸ to the "other side." Additionally, those seeking to remain on the mainframe and extend their technological reach across platforms will also find this book very valuable.

Although most (if not all) of the discussion in this book speaks as though your decision to leave the mainframe is imminent, I do realize that there are still production applications to be maintained. After all, you *do* want to leverage your assets, and you will not complete a successful transition overnight.

My quandary is that I have seen what it is like on the other side and the grass is greener. Once you are bitten by .NET, it will be difficult to approach your legacy

8. Converting, reforming, transitioning—it's all the same. You get the point.

mainframe development with the same zeal and enthusiasm as before. My guess is that once you immerse yourself into .NET, you will find it more and more difficult to mentally switch back and forth from one platform to the other. Eventually, you *will* choose one platform over the other, and this book that you are holding in your hands will serve as your guide to make this choice a lot less confusing.

The ideal reader is someone who is already proficient in any of the following:

- Batch COBOL programming language
- Interactive online CICS screen development
- Batch or online COBOL database programming using DB2

The only other requirement is that you are genuinely interested in learning about the .NET Framework and .NET development tools. You should be prepared to commit and prioritize to allow for dedicated study and practice time.



NOTE Reforming from the mainframe COBOL-oriented world is a significant accomplishment. A successful transition over to the .NET world requires that you embrace this new technology wholeheartedly. You will need to work hard, and the reward will reflect the amount of effort that you invest into this endeavor. Buying this book is one big step in the right direction.⁹

I've been there. I even straddled the fence (between the mainframe and the "PC") for a while. Eventually, I crossed over and proceeded to pursue the PC technologies with a relentless hunger and passion. When I first crossed over, I compared the new PC wilderness I entered to the "wild, wild West." I ended up wasting a lot of time and money trying to figure out *what* to learn first. I just needed some direction and a productive perspective. This book will be your time-saving guide. With guidance, your programming background will provide the foundation that will really make the big difference. I will go as far as saying that my mainframe foundation has better prepared me for this latest transition opportunity: the .NET transition. As a former mainframe programmer, that is your advantage as well.

Allow me to remind you (as I reminisce) that we mainframe programmers were groomed in an environment that has matured and continues to advance to this day—an environment that recognized the value of time-tested methodologies, standards, and disciplines. We took for granted that the old-timers were

9. I know, I know, a shameless plug.

always there to serve as our mentors and that we could always pick up an IBM tome/manual for the “last word” on resolving a best practice debate. We looked at security mostly as something that kept us from accidentally editing production files, not something that protected us from faceless viruses.

The world has changed. The .NET Framework and VS .NET is part of that change . . . and so are you.

Chris Richardson

Richardson@eClecticSoftwareSolutions.com

What Is .NET?

Defining and Connecting the Dots

In this chapter

- Covering the essentials of .NET programming
 - Exploring the various ways to access data using .NET
 - Reviewing the use of .NET to interface with the user
 - Introducing advanced .NET technologies
 - Understanding the roles of marketing and planning for .NET
-

THIS CHAPTER MARKS a significant milestone. You are now one quarter of the way into Part One of this book. Why is that a *significant* milestone? It just happens that Part One (“The Mainframe Paradigm Shift”) focuses on preparing you for your .NET retraining effort. Therefore, you have progressed a quarter of the way toward your retraining preparation. So, congratulations! Now, what will you need to do to complete your preparation? Well, that is where this chapter comes in.

This chapter will provide you with a full definition of .NET. While revolving around the question of “What is .NET?”, I will discuss several answers to that question. The .NET definitions will include everything from the basic programming aspects of .NET and the new .NET development tools to new .NET concerns centered on XML and Web services. To finish the chapter off, you will stretch your view of .NET to even include portions of the .NET platform that deal with enterprise servers, your career, and marketing concerns.

As you have deduced, this chapter will fulfill yet another quarter of your retraining preparation. The two chapters that follow this one (Chapters 3 and 4) will complete the remaining half of your retraining preparation. The four chapters in Part One of this book will have fully presented to you the mainframe paradigm shift.



NOTE This chapter is designed to introduce you to the scope of .NET, and it does not contain any code samples. Part Two of this book provides you with hands-on programming examples. There, you will find an ample supply of programming code samples.

Putting the .NET Question into Perspective

Instead of asking “What is .NET?” suppose for a moment that circumstances were reversed. Someone (aware of your impressive mainframe background) asked you, “What is mainframe programming?” In an attempt to give a full answer, would you start by telling the person about the collection of programming languages available from which to choose? Perhaps you would go into detail discussing the choices of user interface technologies. As you know, your answer would be terribly incomplete if you did not explain the various data access technologies, the Job Entry Subsystem (JES), and the development environment. Obviously, there is more—a lot more. Therein lies my point with this analogy: The world of .NET is also huge.

So, how do you get your arms around something this huge, this encompassing? Well, have you ever heard the saying “You eat an elephant one bite at a time”? To answer the question “What is .NET?” let’s take that same approach. While staying at a rather high level, I’ve divided the topic of .NET (the “elephant”) into the following “bites”:

- Programming Essentials with a .NET Language
- Accessing Data the .NET Way
- Interfacing with the User Using .NET
- Understanding Advanced .NET Technologies
- Marketing and Planning for .NET

In the following sections, where appropriate, I’ve included Cross-Reference notes to later chapters in the book. In these cases, you’ll find that specific chapters cover the .NET topics in much more depth. For now, in this chapter, I present a big-picture view while defining .NET. So, let’s get started.

Programming Essentials with a .NET Language

In this section, I discuss .NET programming languages, the .NET development environment, and the core underlying .NET technologies. Of course, this discussion will begin our efforts to answer the question of “What is .NET?” Some people might expect that coverage of programming essentials alone would be sufficient to help define .NET. As you will see, it is easy to understand why some have that expectation.



CROSS-REFERENCE The topics in this section are further discussed in Part Two of this book.

This section explains how .NET qualifies as all of the following entities:

- Programming language choice
- Programming language support
- Development environment
- Collection of class libraries
- Virtual machine and runtime
- Object-oriented technology

.NET Is a Programming Language Choice

That’s right, you have a chance to choose your .NET programming language. You make this language choice by selecting your specific .NET compiler. You can actually refer to a given language compiler as a .NET type of language compiler. However, to be technically correct, you would use other phrases such as “the language compiler supports the .NET Framework” and “the language is *managed*.” Later, in this same “Programming Essentials with a .NET Language” section, I discuss the .NET Framework and what it means to be managed.

According to Microsoft, there are about 20 .NET language compilers. This count includes those compilers provided by Microsoft and other compilers

developed by other software vendors (partners). A few of the .NET programming language compilers provided by Microsoft are as follows:

- Visual Basic .NET
- Visual C# (pronounced “C sharp”) .NET
- Visual J# (pronounced “J sharp”) .NET
- Visual C++ .NET

Among the list of partners, one in particular stands out (in my eyes¹). Let the history books record that Fujitsu Software is the first vendor (and the only vendor for now) to come forth with a .NET compiler for COBOL. As mentioned in Chapter 1, this particular compiler version product is called NetCOBOL for .NET. For all reformed COBOL developers, this decision by Fujitsu Software to jump onboard (teaming up with Microsoft) will stand to be a critically valuable one. Therefore, when you do .NET development, you actually have choices related to the language syntax and language compiler with which you develop.

An Opportunity to Choose

Recall that the mainframe offered several popular programming language choices for business programming: COBOL, assembler, Easytrieve, and Dyl280, to name a few.²

Each of these mainframe languages had specific advantages over the others. Some of those advantages related to design and compile-level characteristics. In the case of Easytrieve and Dyl280, compilation (or rather, the lack thereof) was even an issue.

Fortunately, in the world of .NET, a language that is said to be a .NET language is able to stand shoulder to shoulder with any other .NET language. In other words, as far as functionality and performance is concerned, one .NET language does not have a clear advantage over another.

-
1. No offense intended to those who share a passion for other languages such as Fortran, Pascal, and so forth. If it is any consolation, even those languages (Fortran, Pascal, and others) have .NET compilers built (or being built) to .NET-enable them.
 2. Using the words “Easytrieve” and “Dyl280” in the same sentence with the words “programming” and “language” may be a questionable approach. Later, in Chapter 4, you will see that the Web/Windows world has continued the tradition (as the mainframe world did) of loosely using the word “language” to describe various technologies.

Because the .NET development environment is designed such that languages are “neutralized,” performance is not a criterion; in theory, all the languages will perform well. That being said, you may tend to make your .NET language choice based on personal preference. However, you should take many other things into consideration. Suffice it to say that whether your preference is Visual Basic, COBOL, C++, or even Java, there is a comparable .NET language compiler waiting for you.



CROSS-REFERENCE Chapter 5 covers .NET language choices in further detail.

.NET Is Complete Programming Language Support

The previous section emphasized the availability of .NET as a language choice. Now, once you *do* select a .NET language (or languages), you will notice that your chosen .NET language has all of the normal language elements (i.e., conditional logic, loops, case statements, comparative operators, and so on) that you would expect of *any* programming language.³ In that way, programming on the .NET platform becomes comparable with the legacy mainframe COBOL programming that you have (presumably) done for some time.

During your legacy mainframe development experience, you have likely used COBOL (and other languages) to implement your business rules using these normal language elements. Well, in .NET you will look to these familiar elements and concepts (or language features) to implement your business logic and presentation routines. Granted, the syntax will vary from one .NET language to the next. Nevertheless, you will find that these basic programming elements are there still.



CROSS-REFERENCE Chapter 6 is devoted to the “nuts and bolts” of .NET programming.

3. This is at least true for each of the .NET languages that I discuss. Chapter 6 covers this topic in more depth.

.NET Is a Development Environment

In the mainframe world, you were accustomed to having an actual environment in which to develop your software. One mainframe tool comes to mind: Interactive System Programming Facility (ISPF). Some of you may have had the misfortune of working in mainframe environments where other products were in use—competing products that looked like and worked like ISPF (sort of). At any rate, you had a development environment. In this mainframe environment, you had an editor. Additionally, you had the ability to construct and compile in ISPF.

Whether you were doing batch programming or online programming, you typically did (almost) everything within this same environment. There were some exceptions—times when you left ISPF for certain tasks. Nevertheless, for the most part you leveraged the submenus on ISPF for your development needs. In that sense, you can say that ISPF (and products like it) represented an “integrated development environment.”

Well, you guessed it, there is an *integrated development environment* (IDE) in the world of .NET. Microsoft has a product called Visual Studio .NET (VS .NET). In short, it is an awesome tool. In this IDE, you design, edit, compile, and test your programs. As in the case of the mainframe, there are times when you will work outside of VS .NET. Over time, vendors will attempt to create competing development environments. Only time will tell which one will rise to become the preferred IDE. For now, Microsoft’s VS .NET is easily your .NET IDE of choice.



CROSS-REFERENCE You will explore the VS .NET tool in great depth in Chapter 5.

.NET Is a Collection of Class Libraries

Imagine that you’re developing a COBOL program on the mainframe. Perhaps you have a group of mainframe partitioned datasets (PDS) that make up an extensive library of copybooks, utilities, subprograms, and software routines. Let’s say that this reusable software library has been proven bulletproof, so much so that you’ve come to rely on this library and you often reuse selected PDS members.

Now, add on top of all of this that you chose to leverage the inherent COBOL Report Writer module and internal COBOL SORT features. Obviously, your intention would be to maximize any plumbing that the COBOL compiler provided

(that you may have a need for) and maximize any reuse opportunity that the reusable software library offered.

Well, with this mainframe analogy I have just about described the functionality of Microsoft's .NET Framework. However, to really be fair, I would have to take this imaginary reusable library and multiply it by a factor of 4,000. The .NET Framework contains several thousand reusable software *pieces*. Regardless of the .NET language that you happen to use, the full .NET Framework is available to you. The more that you leverage (reuse) the .NET Framework, the better off your .NET development experience and resulting .NET application will be.



CROSS-REFERENCE Although selected portions of the .NET Framework will appear in almost every chapter in this book, you will focus directly on the .NET Framework in Chapter 7.

.NET Is a Virtual Machine and Runtime

Suppose for a moment that I asked you to define the mainframe's Job Entry Subsystem (also known as JES, JES2, or JES3). You might explain JES in this way: You submit your batch programs in the form of Jobs to the operating system. The operating system in turn hands your Jobs to JES for execution. JES then manages the priority execution of your Jobs. Following execution, JES handles the purging of your Jobs from the operating system. So, what does this have to do with defining .NET and answering the question "What is .NET?"

Well, sitting on the bottom of the entire .NET world is a foundation, an engine referred to as the *common language runtime* (CLR). The CLR manages the fine details of what your program is doing, in much the same way that JES manages your Jobs. As JES purges your Jobs from the operating system, the CLR might purge your software objects from memory. You could say that the CLR is a micromanager of sorts. As JES is a major piece of system-level software, the CLR is an equally major piece of system-level software.



CROSS-REFERENCE Chapter 8 discusses the CLR in detail.

.NET Is an Object-Oriented Technology

The emphasis on the .NET Framework earlier in this chapter should also be applied to this topic of object orientation. Putting the two together will lead you to the creation of more maintainable and reusable components.

Good Design Practices Are Still Needed

For the reformed mainframe programmer, moving *toward* the object-oriented software development model means moving *away* from the following software development models:⁴

- Structured
- Procedural
- Top-down
- Spaghetti

Of course, well-seasoned developers (like us) have only developed applications using the structured development models. So, spaghetti code is a thing of the past, right?

Wrong. Although .NET is many things, it is not a panacea. As it turns out, even object orientation will not prevent developers from writing spaghetti code. However, fully leveraging all that .NET has to offer *will* encourage good program design. So, as you start writing great .NET applications using the new object-oriented development methodology, remember that design/code reviews and quality assurance processes continue to offer value.

In the world of .NET, you will be working with objects: Everything is an object. In your object-oriented program, you will create, reference, modify, and pass objects. After you understand the world of object orientation, you will prepare yourself to create maintainable, robust, and scalable applications. You will strive to create efficient code, and you will even learn not to create memory management problems in your application.

That's right, the concerns of memory management still exist. The concerns of managing memory will feel painfully familiar to the advanced mainframe Customer Information Control System (CICS) programmer who has worked with the GETMAIN and FREEMAIN storage commands. In addition, the phenomenon

4. This, of course, may not apply to those (few, relatively speaking) of you who have already begun object-oriented COBOL development on the mainframe.

referred to as “memory leak” is familiar to some advanced CICS mainframe programmers (like yourself). The good news is that .NET has implemented a solution to help developers in the area of memory management.

You will be introduced to this .NET-implemented feature, the garbage collector (GC), in Chapter 8. The GC will do some of what the mainframe FREEMAIN command did and more. As I mentioned earlier, in .NET everything is an object. The GC is just one aspect of what .NET offers to assist you in managing objects. Learn object orientation, and then prepare to work with .NET objects extensively.



CROSS-REFERENCE I discuss the topic of object orientation in Chapter 4 in detail. In Chapter 9, I discuss object orientation in even more detail as it applies directly to the coding samples.

Sure, there are objects and the .NET Framework, but what about data? You were just about to ask that question, right? Eventually, you *are* going come across the need to read and/or write data. As I discuss in the next section, the .NET platform has great support for accessing data.

Accessing Data the .NET Way

I must admit that this is probably one of my least favorite topics because it clearly flags the departure from using Job Control Language (JCL). That's right, after all of these years of using JCL on the mainframe for most of your data needs, along comes .NET. So, for the reformed mainframe programmer, .NET is a fond farewell to JCL.



CROSS-REFERENCE Part Three of this book further explores the topics presented in this section.

This section discusses using .NET to get data in the following ways:

- Without the help of JCL
- From a relational data source
- Described with XML

Getting Data Without the Help of JCL

As you know, on the mainframe JCL is used for many things, not just data access. JCL has been there for us through the years, providing a way to allocate the resources needed by our programs. Nevertheless, in the average mainframe JCL structure (Job), the data definition (DD) statements account for well over half of the JCL statements used. So, I repeat, it is time to say good-bye to JCL. Yes, as part of the package of your *reformation* away from the mainframe and into the .NET world, JCL is no longer a tool available in your arsenal. Period.

Now, here is the good news: .NET provides several ways to access data (and allocate other resources). Each of the available ways has a specific strength. As you learn about these approaches, you will discover that you will have a preference for one approach or another, depending on your needs. On the mainframe, you used one approach for regular sequential (Queued Sequential Access Method, or QSAM) files and a different approach for Virtual Storage Access Method (VSAM) files. By the way, for you, QSAM and VSAM files have joined JCL on the fond farewell list.

OK, before I upset a few people: Yes, I should have said QSAM and VSAM, *as we have known them*. First, the text files that you will work with on the Windows platform will remind you of mainframe QSAM files. Second, some vendors have created very useful tools that run on the Win32 platform to create indexed files. These indexed files behave similarly to mainframe VSAM files. For example, Fujitsu Software has a Win32 product called COBOL File Utility that will create an index for your Win32 text file. Granted, with this Win32 index file, you do not have all the power of a traditional mainframe VSAM file, but the Win32 file *is* indexed.



CROSS-REFERENCE Chapter 10 introduces a new perspective on data and covers several .NET tools for basic data access.

Obtaining Data from a Relational Data Source

Although a relational data source isn't your *only* choice for data access, it certainly is one of your choices. As it turns out, using relational databases in Windows and Web applications (and on .NET) is extremely popular. In Chapter 11, I discuss not only the actual coding concerns, but also the use of the SQL Server basic administrative tools.

On a similar note, Microsoft's SQL Server isn't your *only* choice of relational data source system. There are other good products on the market (e.g., Oracle,

Microsoft Access, and so forth). Nevertheless, in this book, you will use Microsoft's SQL Server (currently SQL Server 2000) for your database-related samples.

For those of you who have worked with IBM's DB2 RDBMS on the mainframe, you will feel right at home with Microsoft's SQL Server 2000. Well, sort of. The Structured Query Language, or SQL (now Transact-SQL, or T-SQL), is virtually the same as the SQL used to access DB2. You will realize the learning curve when you start looking around for some of the mainframe tools such as DB2 Interactive (DB2I), SQL Processor Using File Input (SPUFI), and Query Management Facility (QMF).

Working with SQL Server 2000, you will have a new set of tools to learn, namely Query Analyzer and Enterprise Manager. You can be certain that eventually you will need to write a database application. Therefore, time spent mastering these tools is time spent wisely.



CROSS-REFERENCE I further explore the topic of T-SQL in Chapter 3. Then, to smooth out the learning curve for relational data sources, I discuss SQL Server 2000, along with a new set of related tools, in Chapter 11.

Getting Data Described with XML

As mentioned in Chapter 1 during the discussion of the J4 committee members' activities, practically every software vendor is using XML in one way or another. Well, so is Microsoft. But to what extent? Let's just say that XML is to .NET as blood is to the human body. Therefore, it is appropriate to use this topic to help answer the question "What is .NET?"

With XML, you will be accessing data. Specifically, you will find yourself describing, reading, and writing data with XML. You will be amazed at the various ways in which XML is used and can be used throughout the .NET platform. Generally, you can look at XML (especially XML Schemas) as a way to describe your data, much as you would use a mainframe COBOL copybook. Additionally, when you are writing (and reading) data in your .NET application, .NET makes it very convenient to write your data in the format of XML. You will even find that all of your .NET configuration concerns will be addressed using XML. XML truly runs through .NET inside and out. As you dig deeper into .NET (and XML), you will become more comfortable working with XML. You will quickly grow to view it as just another tool/standard to help you with your data concerns.



CROSS-REFERENCE I expand on the topic of XML in Chapter 4. In Chapter 12, I further discuss XML in the context of describing and handling data.

Whether data is structured with the help of XML or a relational database, or accessed with ADO.NET (or any of .NET's innovative approaches), someone will want to see that data. In fact, a user will typically want to interact with your application and any data that is exposed. I discuss this aspect of .NET in the next section. As you continue through the chapter, perhaps you can appreciate (even more) that defining .NET is a significant undertaking.

Interfacing with the User Using .NET

In your previous career life, prior to your move toward *reformation*, you created user interfaces. Perhaps you used the Customer Information Control System (CICS) or the Interactive System Programming Facility (ISPF) to design and build useful screens. These screens provided the “face” in the word “interface.” Welcome to the .NET world. You now have a new set of .NET tools for creating your graphical user interfaces (GUIs). Also, with .NET you will be creating nongraphical “program interfaces.”

The User Is Still Always Right!

Regardless of the platform (mainframe or Windows/Web), all of the same “interface” design considerations apply. You will still want to clearly identify your target user group (those users who will interface with your application). Your interface should provide targeted users with the most pleasurable (hassle-free) experience. Although the term “user-friendly” may sound old-fashioned, to the user community this word is as fresh as ever. In some cases, you may find that the Windows/Web users are more demanding (remember, they do have a PC on their desk). In other cases, you may find just the opposite (with users being willing to do more on their own). The point being, you should learn the new .NET tools for developing application interfaces and remember to allow some time to understand the needs and expectations of your application's target audience.

Going forward with the “What is .NET?” question, this section discusses .NET in the following contexts:

- Windows, Web, and XML Web services development
- Use of a Toolbox during development
- State and event management
- Report creation and information delivery
- Deployment improvement



CROSS-REFERENCE Part Four of this book further discusses the topics in this section.

.NET Is Windows, Web, and XML Web Services Development

This section explores a few additional .NET definitions that I have come across. These are my favorite so-called .NET definitions. Why? Because early on I recall using each of them myself.

.NET Is Windows Development

Windows programming on the .NET platform is an exciting sandbox to play in. Though the Windows Form may not get as much attention as its cousin the Web Form, Windows programming is more alive than ever.

Although you were able to develop Windows applications with previous versions of Visual Basic and Visual Studio, your development experience just got (much) better. For example, your .NET desktop applications can now leverage the full power of the .NET Framework. On top of that, you can now easily deploy Windows and desktop applications over the Internet directly to the remotely located user. Make sure to spend some time in the area of creating great applications for the desktop.

Portable Devices Have Windows

You may find yourself developing applications for portable devices—not just portable computers, but also personal digital assistants (PDAs), mobile phones, and wristwatches. In other words, .NET, when combined with the .NET Compact Framework and Smart Device Extensions for Visual Studio .NET, is a development platform for portable devices.

Windows development with .NET is a broad and exciting area to be in. Some developers will even include Microsoft Office objects (Word, Excel, and so forth) in their application solutions. In addition, collaborative workflow-type applications are gaining popularity. Although these solutions were possible prior to the existence of .NET, the enhancements that .NET brings to the table will make these types of implementations that much more attractive. Use .NET to build Windows applications—it is all possible.

.NET Is Web Development

In the previous section, I mentioned that Web Forms tend to get a lot of attention. As the Internet is very popular, the technologies used to create Web sites should follow that popularity. Becoming proficient in this area will have you learning about ASP.NET and HTML, along with many other technologies. If you have built mainframe CICS applications, ASP.NET and HTML will reintroduce you to many familiar concepts. You will find that the learning curve is not that steep, especially with the help of the great book in your hands.⁵ Considering the popularity of Web sites and Web site development, it is easy to see why this particular answer for the question “What is .NET?” is a common one.



CROSS-REFERENCE Chapter 4 delves deeper into the topic of HTML.

5. Pardon me. Thanks.

.NET Is XML Web Services Development

What is .NET? Here is one very popular answer: XML Web services. Notice the three words: “XML,” “Web,” and “services.” When combined (as in “XML Web services”), they equate to what is becoming *the* answer. Certainly, XML Web services are something to get excited about. However, for our purposes here, if .NET *is* XML Web services, then what *are* XML Web services?

Again, let's refer back to mainframe CICS application development. In the past, you may have created a special CICS transaction that didn't have an actual screen associated with it. This special CICS transaction was usually referred to as a *started task*. You would use a CICS START command to execute the started task, possibly passing data to the started task using the FROM option.

If you are familiar with this type of advanced mainframe CICS programming, you are already acquainted with the general idea of XML Web services. Obviously, there is more to the CICS started task technology. Likewise, XML Web services are very powerful and require further explanation.



CROSS-REFERENCE Chapter 13 drills down further into Windows, Web, and XML Web services development.

.NET Is Using a Rich Toolbox During Development

As you dive deeply into VS .NET, you will explore the Toolbox. You will find that depending on the type of application that you are creating, the contents of the Toolbox will vary. For example, a Windows application may have certain types of controls (Toolbox contents) that are not applicable to a Web application and vice versa. The Toolbox can contain visual controls as well as other types of controls and components, such as data controls.

The Toolbox is designed to be exactly what its name implies: a container to hold tools that help you build applications. In Chapter 1, I mentioned rapid application development (RAD). Certainly, the Toolbox plays a large role in the RAD approach, especially when a developer fully leverages the contents offered in the Toolbox. The concept of having a Toolbox may be a bit strange at first for the reformed mainframe programmer. The good news is, you will quickly get used to using the Toolbox. To help get you started, I will present an analogy.

Imagine that you are developing a mainframe CICS application. At some point, you typically will create a screen for the user to interact with. This step requires that you create a BMS mapset and all of the corresponding map information. Now, suppose you have a PDS library available to you with several “pieces” of BMS mapsets that you could drag and drop to help you create your CICS screen. You can see how this type of reuse could lead to increased productivity and consistency (which also promotes maintainability).

Confused? Don’t worry, you’ll understand and master this feature later.



CROSS-REFERENCE Chapter 14 explores the Toolbox and its contents in more detail.

.NET Is Enhanced State and Event Management

Let’s go back to your mainframe CICS development to understand state and event management. Recall the CICS DFHCOMMAREA and the CICS RETURN command. Together, these two CICS technologies provided a way to pass data from one execution of a transaction/program to the subsequent re-execution of the same transaction/program. In mainframe terms, these CICS technologies supported the idea of a transaction/program being *pseudo-conversational*. Well, the idea of saving this data while conducting a “conversation” with the user is a very general hint at what state management involves.

On .NET, you could define *state management* as a methodical approach by which you first identify an established conversation (being held between your application and your users). Once you identify the conversation, you will need to keep track of (manage) selected data specific to the established conversation (interaction) for any given user. Fortunately, .NET has some really cool features to make state management rather simple. You will learn more about them in Chapter 15.

Let’s return to the mainframe CICS application analogy for a moment. To help explain what events are, I’ll use one term: events. That’s right. Well, in all fairness, when you do mainframe CICS development, you sometimes would use the phrase “event and response.” You would use the Attention Identifier (AID) keys to determine what events had taken place.

On the mainframe, do you recall that there was an event/response chart you used when designing CICS programs to help manage the expected events and

responses? Good—most programmers would have used this event/response chart right after the traditional flowchart was completed. This chart and the accompanying focus given to events and responses when designing a CICS program have prepared you for the type of event management awaiting you on the .NET platform. Even though the phrase “event management” is rarely used on the .NET platform, events are events, whether on the mainframe or off the mainframe. What’s more, you’ll need to manage your use of .NET’s events. It’s the same idea and concept, yet a very different implementation, as you’ll see.



CROSS-REFERENCE Chapter 15 further discusses state and event management.

.NET Is Report Creation and Information Delivery

I mentioned the mainframe COBOL Report Writer module earlier in this chapter. Recall how useful the COBOL Report Writer module was (when properly used). As mentioned before, the .NET Framework also has reusable classes to give you a big head start in creating reports.

For those of you who equate the phrase “report generation” with something that you get to do only when you are being punished, fear not. .NET has come to your rescue. Report generation has returned to reclaim its rightful distinction of being a glamorous endeavor.

With the .NET Framework objects and the built-in Crystal Reports objects, your interest in creating reports will be recharged as you create charts, spreadsheets, PDF documents, and other types of reports. You will see why the topic of report generation is now starting to blend in with the phrase “information delivery.” Once again, you can hold your head high and proudly say that you are implementing a report generation (er, rather, an *information delivery*) application. If it sounds like I am getting excited about .NET, I am—and so will you.



CROSS-REFERENCE Chapter 16 covers report generation and information delivery.

.NET Is Improved Deployment for .NET

You may have read about .NET's new XCOPY deployment feature. Next to .NET's improved packaging feature, the XCOPY feature is certainly highly regarded by many Web/Windows developers. Coming from the mainframe world, your reaction to this feature will certainly be different from the average pre-.NET Windows/Web developer's reaction. You see, the deployment that takes place on the mainframe already looks like XCOPY deployment. That is, as long as you are talking about offline batch program deployment. If you switch over to the mainframe CICS online type of program deployment, then, as you know, things get rather complicated.

So, when you see your new Windows/Web developer peers getting excited about .NET's improved deployment approach, try to get excited with them. You can feel fortunate that you are getting involved with the Windows/Web development world at a time when something like .NET exists. Hopefully, you will not have to maintain any legacy Web/Windows application. Then maybe you will never have to find out just *why* .NET deployment advances really *are* a big deal. If you had to live through what was called "DLL Hell," you too would gladly include this topic when answering the question "What is .NET?"



CROSS-REFERENCE Chapter 17 further explores .NET deployment.

Considering each subsection covered in this larger section, you may be surprised to find out that the next section deals with *advanced* .NET technologies. Yes, many of the previously discussed .NET explanations may have appeared advanced. Nevertheless, these previous sections were excluded from the advanced section. This is not to say that the topics covered up to this point are not important or even technologically superior—they are. This simply points out how enormous the .NET platform really is (and why it takes an entire chapter just to answer the question "What is .NET?"). In other words, .NET is not a toy. It is a real enterprise-caliber tool. So, please continue on to the next section to discover which topics are actually considered advanced.

Understanding Advanced .NET Technologies

The entire world of .NET *is* understandable. You may have to work hard, study hard, and practice hard. Nevertheless, it is doable. This section takes a quick look at the following advanced topics:

- Secure and configurable applications
- COM+ application creation
- Distributed and concurrent processing
- Interoperability



CROSS-REFERENCE Part Five of this book further discusses the topics presented in this section.

.NET Is Secure and Configurable Applications

Security is probably one of the most discussed topics in some circles. And yes, Microsoft has addressed this concern. So, when some people answer the question “What is .NET?” the security features will come to mind first. Portions of the .NET Framework you will learn about explore this topic completely. Additionally, the Global XML Web Services Architecture (GXA) WS-Security standard also increases the level of security available to .NET applications. You will want to learn about both.

Include Security As Part of Your Web Application Design

The area of security for a reformed mainframe programmer will seem strange. Not because the mainframe programmer does not appreciate the need and importance of the topic, but rather because the average mainframe programmer typically did not *need* to worry about security.

Even when you developed large mainframe CICS online applications, there was an entire team of security experts that secured the environment—mainly from internal employees. In the Web development world, your application is potentially exposed to the outside world. So now security is more of an application-level and enterprise-level problem, and thus your problem to share. You will need to take this paradigm shift very seriously.

Others will quickly refer to the enhanced, configurable features. .NET provides a host of configuration files. These configuration files are available at the user level, application level (both Windows application and Web application), machine level, and system level. .NET has certainly taken the idea of being configurable to a new level (pun intended).

For those programmers who have historically avoided hard-coding *anything* in their program code, these configuration files will be welcome. During your mainframe development, perhaps you created QSAM files and PDS members that contained configuration information and parameter values. Now you have this set of configuration files to take advantage of. Again, this is a similar idea and a similar concept—it's just the implementation that's different.



CROSS-REFERENCE Chapter 18 discusses configuration and security in depth.

.NET Is COM+ Application Creation

In the next chapter, I define COM+. For now, I will just mention that COM+ on the Windows platform is similar to CICS itself. In other words, in the same way that individual transactions and application programs are installed on top of the system-level program CICS, you can have application-level programs installed on top of the system-level program COM+.

There are many services that COM+ offers to an application. If you want to leverage those services, you create a COM+ application and install your program into the COM+ application. I will provide step-by-step instructions on how to accomplish this.



CROSS-REFERENCE I have extended this introduction to COM+ in Chapter 3. Later, in Chapter 19, I discuss COM+ further.

.NET Is Distributed and Concurrent Processing

If there were such a thing as a *really* advanced section in this book, this particular section would be it. That is the reason I chose to discuss these two topics in more detail near the end of the book. Allow me to point out that both distributed processing and concurrent processing are rather important topics. There is a possibility that one of your future applications will need to take advantage of either of these advanced features. That is certainly reason enough to not ignore these advanced topics. After all, this book is designed to be your one-stop guide to .NET.



CROSS-REFERENCE Chapter 20 explores distributed and concurrent processing.

Distributed Processing

On the mainframe, have you ever used CICS's Distributed Program Link (DPL) feature? Don't feel bad if you haven't. Even on the mainframe, this is considered to be an advanced topic. Nevertheless, .NET has a feature called .NET Remoting that is similar to the DPL feature.

On the mainframe, if you had a CICS program on one CICS region that you wanted to be able to "connect" to from a different CICS region, you would use the DPL feature. On .NET, if you have a program on one machine that you want be able to communicate with a program on a different machine, you would use .NET Remoting.

You might wonder, "If this .NET Remoting feature is similar to the mainframe CICS DPL feature, will this feature be equally obscure?" To that, I would have to respond, "It depends." It depends on the particular needs of your users and the types of applications you plan on developing. Even more, it will depend on the physical configuration of your production (hardware) environment and your organization's security/firewall policies. You will explore these variables and others in Chapter 20. At that point, you will gain an understanding of when it is appropriate to include .NET Remoting as part of your application design.

Concurrent Processing

The topic of concurrent processing leads to the area of (.NET) threading. When I discuss concurrent processing further in Chapter 20, I compare it with the CICS features multitasking and multithreading.

.NET threading (or multithreading) is certainly a sensitive topic. There are those who will quickly swear to its usefulness. At the same time, there are those who have been “burned” so badly from previous attempts to use it that they will not even enter into an open discussion of the topic. And then there are those who are simply excited that the current version of Visual Basic (VB .NET) finally supports true multithreading.

Regardless of which camp you belong to, you will still want to know how to use threading—properly, that is—in the event that you can justify using it. Generally speaking, an application that has significant processing overhead may (I repeat, *may*) be a candidate. If the timely completion of key portions of this same application becomes critical (e.g., an online application that interacts with a user), this would further support the notion of the application being a candidate for explicit threading management. Furthermore, if you have designed your application such that multiple threads can safely be dealt with explicitly, you just might (I repeat, *might*) have yourself a candidate for .NET’s threading.

Obviously, discussing *when* and *when not* to include explicit threading in your program design is important. Simply learning *how* to programmatically manage threads is equally important. In Chapter 20, when I further discuss this topic, I trust that you will be adequately informed, debriefed, and enlightened.

Yes, distributed processing and concurrent processing are rather advanced topics and should be treated as such. As you learn more about them, you will be armed with enough knowledge to use caution when traveling down the path of “alternative” processing models. Nevertheless, you *will* want to know that these features exist, just in case you ever need them.

.NET Is Interoperability

You can use multiple languages on .NET. However, what about the topic of interoperability? What about .NET’s capability to allow modules of multiple languages to interact and coexist in the same application? With .NET, software modules written in one language (e.g., VB .NET) can easily interoperate with software modules written in a different language (e.g., C#). .NET is ready for these types of situations.

In fact, interoperability is a key characteristic of .NET that seems to get a fair amount of attention. I believe this can be attributed to the fact that when you need interoperability, you *really* need interoperability. Inclusion of interoperability

as part of your application design usually means that there was not a more viable choice.

.NET Supports Component Object Model (COM) Interoperability

If you need to leverage existing Windows/Web legacy modules (i.e., COM modules), .NET's COM Interoperability (Interop) feature supports this need. For example, say that you have older COM objects that were written before .NET existed. If you need it to, your new .NET application can use COM Interop to directly integrate with the older, non-.NET module. Conversely, you can take new .NET (managed) objects and use them in your older COM (unmanaged) environment. Remember COM Interop just in case you inherit any legacy applications while you are becoming a .NET expert.

In your previous mainframe experience, you may have worked with applications written in COBOL that used subprograms written in assembler. You may have had Dyl280 programs that depended on subprograms written in COBOL or assembler. In other words, you are already familiar with the concept of language interoperability. What will be new for you here is how easy interoperability is with .NET. For example, with virtually no extra effort, you can interactively debug an application, stepping line by line, with execution transferring from a module written in one language to a module written in a different language. Now, that is a great feature.

Having multiple languages to choose among is great. It's also great that .NET fully supports multiple language integration and coexistence. While I'm on that topic, isn't it great that .NET fully supports each and every technological feature discussed so far? You've got to admit, there *is* a lot to .NET. So "What *is* .NET?"

Discussion from each of the previous sections has contributed to answering *that* question. Continuing on to the following section, you will take a slight turn away from this development/programming-centric focus. Although you will still ask the same question ("What is .NET?"), you will cast your view slightly outward toward the industry and the enterprise. You will find that these additional sides of .NET are in fact relevant and of importance to you.



CROSS-REFERENCE You can find a good example of language interoperability in Chapter 19.

Marketing and Planning for .NET

How are you doing so far? By now, I believe you are getting the point that .NET is quite huge. Remember, *you are eating this elephant one bite at a time* and you have just a few more bites.

In this section, I cover .NET in the following contexts:

- Multiple editions of VS .NET
- Group of Enterprise Servers
- Career choice

.NET Is Multiple Editions of VS .NET

That's right. Multiple editions of VS .NET are available. The good news is that regardless of the edition you happen to get, you will still have the core portions of .NET (VS .NET, the .NET Framework, and the CLR). What will differ are the tools, plug-ins, and features to which you avail yourself. The following list shows the available VS .NET editions (in the order of smallest feature set to largest):

- Professional
- Academic
- Enterprise Developer
- Enterprise Architect

For your initial training purposes, any edition should suffice—even the one with the fewest features. Obviously, as a developer, I could care less how Microsoft's marketing department decided to package the product. My concern is just to know what products are available and to clearly communicate that to you. Then, you can approach the subject with a bit of clarity.

If you happen to come across the following .NET software versions:

- VS .NET Beta 1
- VS .NET Beta 2
- VS .NET Release Candidate
- .NET Framework SDK

I recommend that you avoid installing them—at least they should not be your first choice. At best, use them for training purposes only. When it comes to deploying production code, you will want to have the retail version 1.0—any of the editions mentioned previously. Each of the prerelease versions in the preceding list had its time (the key word being “had”). Now that version 1.0 is out, make every effort to get it. In the worst-case scenario, visit Microsoft’s Web site and download the free 60-day trial version.



CROSS-REFERENCE Currently, a 60-day trial version (Visual Studio .NET Professional Edition) is available at <http://msdn.microsoft.com/vstudio/productinfo/trial.asp>.

Additionally, as version 1.0 of the .NET product moves through its maturity phases (as with any product), service packs are certain to be made available. Periodically visit Microsoft’s Windows Update Web site (<http://windowsupdate.microsoft.com/>) for any available product updates (at a minimum, make it a habit to apply the updates that are marked as critical).

The .NET Compact Framework

As I discussed earlier in this chapter in the section “.NET Is Windows Development,” you may choose to develop applications that target devices other than personal computers (i.e., mobile phones, PDAs, and so forth). For these devices, Microsoft has provided an edition of the .NET Framework called the .NET Compact Framework. The Compact Framework is a subset of the larger, fuller framework that you will use for Web and Windows development.

If you are interested in developing for portable devices, install the .NET Compact Framework (alongside the full .NET Framework) together with the edition of the VS .NET product you happen to have. Then, using the same VS .NET product, you have the option of developing applications for the Web, desktop, and/or portable devices.

Microsoft uses the term “smart devices” to generically refer to all types of devices other than traditional computers. Hence, the .NET Compact Framework software product is accompanied by the Smart Device Extensions for Visual Studio .NET software product.

.NET Is a Group of Enterprise Servers

As I mentioned earlier in this chapter in the section “.NET is Multiple Editions of VS .NET,” I generally care less (as a developer) about the apparent decisions of Microsoft’s marketing department. I am not here to cast judgment on them and the decisions probably influenced by them. In other words, if they want to slap the .NET label onto all of their software products, more power to them. So, rather than spending too much time on Microsoft’s reasoning, just accept it as fact: Microsoft decided to name a collection of their software packages “.NET software.”

What does all of this have to do with defining .NET? Well, to return to the original purpose here of defining .NET, consider the following list of Microsoft .NET Enterprise Servers:

- Microsoft Application Center 2000
- Microsoft BizTalk Server 2002
- Microsoft Commerce Server 2002
- Microsoft Content Management Server 2001
- Microsoft Exchange Server 2000
- Microsoft Host Integration Server 2000
- Microsoft Internet Security and Acceleration Server 2000
- Microsoft Operations Manager 2000
- Microsoft Mobile Information Server 2002
- Microsoft SharePoint Portal Server 2001
- Microsoft SQL Server 2000
- Windows 2000 Server

What *is* .NET? Well, to some, the answer would include some or all of these so-called .NET Enterprise Servers. For that reason, it is important for developers to know what the .NET distinction means (or will mean) when used in reference to one of Microsoft’s software products.

In my opinion, the general marketing direction seems to support the following: In the short term, some of the server software packages will carry the .NET

distinction for marketing purposes only. In the long term, each of the server software packages (eventually) will have native XML support, will run on top of the .NET CLR, will have its object model exposed via the .NET Framework, will be Web-enabled and Web service-enabled, and will then truly be a .NET server.



CROSS-REFERENCE The Flash animation file at the following URL demonstrates the interoperation that is possible among the .NET Enterprise Servers: <http://www.microsoft.com/servers/evaluation/interop.asp>.

Extended Learning Objective

You will want to familiarize yourself with (at least a few to start, and eventually the majority of) the .NET Enterprise Servers. In order for you to reach (or retain) the level of a senior developer, consider it an expectation. Start by learning each server by name. Later, build a working knowledge of each server (particularly the ones in use at your place of employment).

Do you think this is going overboard? Let me remind you that on the mainframe, a senior developer was familiar with most (if not all) of the system-level software packages that were installed, especially the ones that affected production application development and processing. These software packages commonly came from software vendors such as IBM, Candle, and Computer Associates (among others). Software products such as CICS, DB2, OMEGAMON, and CA-7/11 all fall into this system-level software category.

.NET Is a Career Choice

Perhaps you are wondering, “If .NET *is* so many things, how will I be able to become a .NET developer? How will I be able to learn so many things *and* be so many things—all at once?” Well, my reader friend, with very few exceptions, most of the sections that you have read *can* represent areas broad enough for a specialization.

The idea of specialization is not foreign to the mainframe development community. I recall working with a gentleman (years ago) who was respected for his in-depth programming ability. Yet, he always programmed using Dyl280. That I knew of, he did not touch COBOL or assembler—only Dyl280. I repeat, he was

respected and carried quite a bit of responsibility with his Dyl280 specialty. Granted, he probably *knew* of other technologies, but he was a master of the Dyl280 product.

So, in the mainframe environment, you had some who were known as great offline batch programmers or great CICS programmers. I am sure that you have worked with developers that spent their entire day creating powerful REXX and ISPF applications. You may have specialized in something yourself. Yes, occasionally you came across that exceptional person, the one who was a “Master of All Technology.” Or, more often, you came across someone who just *thought* he or she was a “Master of All Technology.”

Is there anything wrong with trying to eat the entire elephant? No, be my guest. Just do it *one bite at a time*. Perhaps you can start by specializing, and then either move on to other parts of the .NET landscape as time allows or find a good fit and stay put. What makes a good fit? Well, everyone is motivated by different things. Some will look at what skill sets demand the highest salary. Others will look at other factors for motivation. To each his own.

For our immediate concern, .NET is a career choice (after all, that is the bottom line). Microsoft has a great career roadmap on their MSDN site that I strongly recommend you examine. Whether you take the “learn everything” approach or the “specialization” approach, this roadmap will help you on your chosen path: http://www.microsoft.com/traincert/training/roadmap/chart_tabloid.pdf.

.NET Is a New Microsoft Certification

Twenty years ago, I worked with a team of mainframe CICS experts that happened to consist of independent contractors. On one occasion, I received a business card from one of the contractors and noticed his professional certification noted as “CDP” and “CCP.” I remember asking the expert about his certification (CDP stands for *Certified Data Processor* and CCP stands for *Certified Computer Programmer*).

His reply was as follows:

“Chris, this certification means nothing if you cannot do the job. First, learn to do your job. Later, if you want a good challenge to keep you skills sharp, go for the certification. Otherwise, the certification would help you if you needed to market yourself for a promotion or in the case of being a contractor—for your next work assignment.”

Therefore, I spent the next 20 years learning how to do my job.

In keeping with the “What is .NET?” theme, .NET is a new distinction for Microsoft certification along with a new set of qualifying examinations. For application developers, there is the new Microsoft Certified Application Developer (MCAD) credential. For solution developers, there is the updated Microsoft Certified Solution Developer (MCSd) credential. Microsoft offers other types of certifications as well, including the one that I have obtained (so far): the Microsoft Certified Professional (MCP) credential.

Summary

This chapter’s goals were as follows:

- To cover the essentials of .NET programming
- To explore the various ways of accessing data using .NET
- To review the use of .NET to interface with the user
- To introduce advanced .NET technologies
- To understand the roles of marketing and planning for .NET

So, what *is* .NET? Throughout this chapter, I have provided several answers to this question. I am certain that now you can appreciate the various ways that this question can be answered. Try it. Ask several people *the* question. Depending on whom you ask, you will get different answers. The great thing is that each answer is likely to be correct. .NET really is a lot of things, including this one thing: Practically the future of Microsoft Windows and Web (also portable/mobile) targeted development depends on it.

The Microsoft executives have gone on record as stating that Microsoft’s .NET commitment amounts to a “betting of the farm” for Microsoft. In other words, .NET is not *just* the next big thing, it *is* the next big thing (especially for Microsoft, and thus for a developer community several hundred thousand strong).

In the next chapter, you will look at your retraining effort from a slightly different angle. You will explore (and remove) some of the common training obstacles that recently reformed mainframe programmers are likely to come across. My intention is to make sure that you have the proper foundation before you remove the brakes and dive even deeper into .NET.

To Learn More

The following are some suggested supplemental references to further your retraining effort.

Magazines

.NET Magazine:

<http://www.fawcette.com/dotnetmag/>

Visual Studio Magazine:

<http://www.fawcette.com/vsm/>

XML & Web Services Magazine:

<http://www.fawcette.com/xmlmag/>

Web Sites

The .NET Compact Framework—Overview:

<http://msdn.microsoft.com/vstudio/device/compactfx.asp>

.NET Enterprise Servers Overview:

<http://www.microsoft.com/servers/evaluation/overview/>

.NET Training Roadmap:

http://www.microsoft.com/traincert/training/roadmap/chart_tabloid.pdf

Microsoft .NET Basics: What Is .NET?:

<http://www.microsoft.com/net/defined/>

Microsoft .NET Language Partners:

<http://msdn.microsoft.com/vstudio/partners/language/default.asp>

Microsoft Certifications:

<http://www.microsoft.com/traincert/mcp/default.asp>

Microsoft Developer Network (MSDN):

<http://msdn.microsoft.com/>

Microsoft Windows Updates:

<http://windowsupdate.microsoft.com/>

Visual Studio .NET Professional 60-Day Trial Edition:

<http://msdn.microsoft.com/vstudio/productinfo/trial.asp>



<http://www.springer.com/978-1-59059-048-5>

COBOL and Visual Basic on .NET
A Guide for the Reformed Mainframe Programmer
Richardson, C.L.
2003, XXXIX, 1032 p., Softcover
ISBN: 978-1-59059-048-5
A product of Apress