Introduction

If you still write RPG code as you did 20 years ago, or if you have ILE RPG on your resume but don't actually use or understand it, this book is for you. It will help you transition from the Original Programming Model (OPM) to a more modern, modular, and efficient ILE RPG.

With this book, each concept of ILE is made accessible. You will start by taking baby steps with small, easily understandable examples, and build to more complete and complex pieces of code. All the while, you will explore each component of modern RPG, learning how it fits with the other pieces to gain the full ILE RPG picture.

By its nature, this book is not an ILE quick-reference guide. Rather, it is a "slow-reference guide." It introduces new concepts with analogies to OPM whenever possible, explaining and expanding with realistic scenarios of increasing complexity (like inventory management programs, for instance).

A Brief Description of Our Journey

This book is divided into three parts:

- *Part One, "ILE Basics"*—Chapter 1 explains each type of object you can have in ILE, which goes a bit further than the programs-only model you are used to in OPM. In chapter 2, you'll learn how and when to create each type. The next stop is procedures, a fundamental concept discussed in chapter 3. Then things start to get really interesting, with examples that consolidate the information of the previous chapters. The code samples continue in chapter 4, which shows you how to build and, most importantly, use your own functions. Chapter 5 is all about parameters. It might sound a bit silly to dedicate an entire chapter to parameters, but you'll see that parameters play a key role in ILE. These five chapters provide a firm foundation upon which you can start coding in ILE.

- *Part Two, "Taking Advantage of ILE"*—Chapter 6 starts with a crucial part of ILE: built-in functions, or BIFs. This chapter covers the most relevant BIFs—get ready, it's a long chapter. Lots of examples are provided, many of which you can use or adapt to your coding environment's reality. After that long stretch, chapter 7 takes you on an easier, lighter path, with tips on how to write and maintain more efficient code, from naming conventions to code organization.
- At this point, you will be ready to take the next step toward modernization: it's time to `/FREE` your code! Chapter 8 is about transitioning to free-format RPG, discussing why you should make the transition, how to do it, and some typical problems and their solutions. Again, examples are provided, with a few surprises. Chapter 9 covers the "new" ILE debugger (`STRDBG`), which replaces the Interactive Source Debugger (`STRISDB`). Chapter 10 introduces the latest and greatest news for RPG, covering the free-format features introduced with V7R1 TR7.
- Chapter 11 is an extended introduction to SQL, covering the basics of both Data Manipulation Language (DML) and Data Definition Language (DDL). If you're not familiar with these names, don't worry; I explain all the necessary concepts, illustrated with simple examples. This chapter also introduces embedded SQL in RPG programs. You'll learn different ways to use embedded SQL, including a few tips on when to employ it and the possible shortcomings and pitfalls of embedding SQL in RPG.
- No chapter about SQL would be complete without discussing the unique possibilities that SQL offers to RPG programmers: you can easily make your RPG code (your fine-tuned business rules validation and enforcement code) available to the "outside world" by using SQL's stored procedures and user-defined functions. This certainly opens up exciting possibilities toward modernization. In a way, it frees RPG from the confines of IBM i, or at least, from the confines of green screens.
- *Part Three, "Beyond ILE—Start Modernizing Your Applications"*—Chapter 12 starts by explaining why you should consider modernizing your applications,

how you can do so, and where you should start. Chapter 13 discusses database modernization, taking advantage of your newly acquired knowledge of SQL, particularly DML, to help you reform your applications. You'll see that there is a considerable amount of RPG code related to data validations that can be replaced by DML constraints. Finally, chapter 14 is about user interface (UI) modernization and how to prepare your code for it. I'll start by introducing a multi-tier model and then explain the model-view-controller (MVC) concept, discussing how you can apply it to your code, thus taking an important step toward more open, flexible, and modern application-building! This chapter ends with a discussion of the RPG Open Access (RPG OA) licensed program, which IBM is now giving away for free, and a discussion of some UI modernization tools that make good use of RPG OA.

By the end of the journey, you'll be a better programmer. You'll have new tools, new approaches, and most importantly, new ideas, to solve those problems big and small that are the life of an RPG programmer.

From Old Problematic Monoliths to Innovative, Lightweight, Efficient Programs

The Original Program Model (OPM), or do-it-all-in-one-program model, has been around for a long, long time. It has served its purpose for many years, but is now rather limited and inefficient. It leads to problematic monoliths of code—huge programs that have to handle the screen interaction, database operations, and report generation. Even if the code is well-structured and commented, it can get very messy because the program is huge. The worst part is, if you have a similar situation in another program (the same business rule or database operation, for instance), you probably have the same code repeated in two (or more) programs.

ILE helps with that. It provides a lighter way to build programs by allowing the reuse of code, instead of its repetition. By using different "repositories" of code, ILE allows you to write code only once and reuse it in a simple way, as often as you want. The shared code between programs exists separately from the programs, and in only one place. The programs use that code as if it were their own. This allows the developer to construct the programs a bit like playing with Lego blocks: use a building block to write a record, use another to check a business rule, use yet another to print a report, and so on.

"Why ILE? OPM Has Served Me Fine So Far"

I like it when my readers reach out to me with questions about my writing; every writer does. It means people are actually reading and trying to use the stuff I write. However, I've noticed more and more that the questions are not about the topic *per se*. Instead, they're about the foundations that every RPG programmer—whether novice or expert with 20 years of experience—should know. You might say, "Why ILE? OPM has served me fine so far."

The problem is that OPM has, as we all know but don't like to admit, many limitations. It can create behemoths of code, with do-it-all-in-one programs that go on and on. This approach works ... until corrections and modifications are necessary. Here's where OPM has one of its biggest problems: it's not easy to maintain "old" code, especially when a change affects many different programs.

The modularity of ILE's smaller, "smarter" programs will save you a lot of time, not only when you are writing the code, but particularly when you are reading it later on. Did you know that on average, a piece of code is read eight times more often than it is modified? Think about it—if the code is simpler, more structured, and smaller, it takes a lot less time to read and understand! That's where I want to take you. This book will (hopefully) guide you on a quest for better programming, with better skills, better standards, and most of all, more efficient code. More efficient code is code that runs faster, uses the latest available built-in functions (BIFs), doesn't execute unnecessary operations, and is easier to read and maintain.

The Virtues of ILE, by IBM Itself (with a Little Help from Me)

Don't just take my word about the virtues of ILE; let IBM convince you! According to the IBM manual *ILE Concepts* (*www-01.ibm.com/support/knowledgecenter/ssw_ibm_i_71/ilec/sc415606.pdf?lang=en*), there are three main advantages of ILE:

- Modularity
- Reusable components
- Common runtime services

Let's consider each of these big concepts, one by one.

Modularity, or Playing with Legos

Earlier, I mentioned Lego blocks. As with Legos, the whole idea behind modular programming is to build with small, simple, reusable pieces of code. Smaller code blocks have shorter compile times and are easier to maintain.

In an OPM program, just to change two lines of code, you might have to read through 2,000 lines. When you finally find what you are looking for and change it, you still have 2,000 lines of code to compile! In ILE, it's simply a matter of identifying the right "Lego block" to change. After that, it's easy: it will be a small piece of code, it will be simple (if it's well-built), and it should compile in a breeze. These blocks are small, specific functions that are easier to understand and adapt, even if they were written in a style different from your own. Since there's a big community of ILE RPG programmers out there, you can do what the Java people have been doing for years: download the source code of a function that performs a specific task from the Internet, compile it, and easily use it in your program.

Modular programs should also be easier to test, although from personal experience, I can tell you that this is not always true. It depends a lot on whether the code was written in a debugger-friendly manner. Finally, with modular programming, the work can be divided. Each programmer can write a "building block" instead of the complete program.

Reusable Components—Don't Rewrite; Reuse!

You probably use something similar to "building blocks" in OPM, by having some subroutines that you copy from one program to another that requires similar functionality. The difference here is that you won't be copying the code; you'll reuse it. You'll write and compile it once, and then every program that needs that functionality will "connect" to the code. (I'll explain what this means and how to do it in chapter 2.)

What you might not know is that these pieces of code can be written in other programming languages, such as COBOL, C, C++, CL, or even Java! In the old days, an RPG shop had RPG programmers; today, the whole "RPG shop" concept doesn't make sense. Today's IT departments are composed of professionals trained in different programming languages. With ILE, you can take advantage of this heterogeneous environment, by using the best that each language has to offer and using it transparently in your programs, as if it were RPG code.

Common Runtime Services—Don't Reinvent the Wheel

IBM supplies a very nice set of off-the-shelf components that you can incorporate into your applications. These components provide message handling, date and time manipulation, math routines, dynamic storage allocations, and greater control over screen handling. Again, from my experience, this is extremely useful. Not only are the tools ready to use, they're also well documented in IBM manuals. Then you have the Internet: loads and loads of code from the RPG community, with varying levels of complexity and documentation, ready to be used.

Remember, these components don't even need to be written in RPG; they just have to "play nice" with ILE. For example, my previous book, *Flexible Input, Dazzling Output with IBM i* (also published by MC Press), features a few components that were originally built in Java and adapted to ILE to facilitate several interesting functions that RPG is not (easily) capable of, such as producing Microsoft Excel files or invoking Web services. So, before writing a generic function, check whether someone (IBM, a third party, or a fellow programmer) has already tackled that particular problem with a piece of code that you can use "as is" or adapt to your needs. You no longer need to reinvent the wheel!

Source Debugger—No Longer the ISDB Nightmare

Every OPM programmer's worst nightmare is debugging a huge program with the Interactive Source Debugger. ILE provides a brilliant and simple debugger that turns that nightmare into a pleasant dream. This debugger, combined with ILE's modular code structure (and the strict adherence to a few rules, explained in chapter 7) makes debugging much more efficient and less time-consuming.

Summary

This book explores the main advantages of ILE. I intentionally left out the more complex aspects of ILE, like activation groups and shared open data paths, which are beyond the scope of this book.

It should now be easy to understand why every programmer should embrace ILE as soon as possible. This book will help you do that. Keep reading, and find out about the basic module, service program, and program concepts in chapter 1.