Chapter 1

A High-Level Introduction to ILE

Before we start, I want to make a few brief remarks about what to expect from this book.

First, it is about three things and pretty much three things only: free-form RPG, Integrated Language Environment (ILE), and Model-View-Controller (MVC). If anything else is covered, it was more or less an accident, and I apologize.

Second, I am going to try to make this as practical as possible, and I will expect you to participate.

I have lots of code samples and many of them actually work.

I will be expecting you to not just follow along but do coding yourself. Nothing says lovin' like something you code up yourself, so be prepared to do more than read. The **Now It's Your Turn** sections are specifically designed to make you do things that you have just read about. Read it, do it, live it.

Nor are any of these topics what I would call hard. But they do require some practice to learn and get comfortable with. Make sure you give yourself a healthy dose of playing with them.

Finally, this book is really meant for those of you who may feel that you are being left behind.

Naturally enough, many books, maybe most, are aimed at the people who are near the cutting edge. Unfortunately, when we are talking about the IBM i technical base, I believe that for every IBM i user who is on 7.whatever, there are three who are still on 5.4 or maybe even below. They do not go to COMMON (no money), they do not get regular upgrades to their system (no commitment from management), they drink mostly Miller or Bud (can't explain that when Sam Adams and Fat Tire are readily available almost everywhere), but they are still there, working on the i every day, using it to support their business, toting that barge and liftin' that bale.

> Or maybe they are on 7.2. But for whatever reason, they are not being allowed to take full advantage of that. They are still using positional RPG, writing big single-module programs that will be almost impossible to modify in a timely manner for today's fast-paced IT environments.
>
> As a nod to this group, I am going to assume no existing knowledge of /Free or ILE or MVC. And almost all the code examples are written from a 5.4 point of view. Because the truth is, you don't need to be on the latest release to be writing modern, effective RPG code.
>
> Oh, I do cover the more up-to-date topics, like the new free-form control statements that can replace F- or D-specs, which came out in 7.1, or some of the stuff that shows up in 7.3. But I treat that as icing, not cake.

So if you feel that IBM is leaving you behind, and you have no choice but to code positional RPG in big ol' programs (BOPs), this one's for you. You don't have to do things the old way. You may not have all the bells and whistles available, but you can modernize your RPG quite effectively even on 5.4. Come on, man. Give it a whirl.

The Organization of the Book

I have given a lot of thought to where we will start and have changed my mind several times. I finally decided that the center of this is ILE; everything we are going to do depends on it, so it's a great place to start. Here is what we are going to do.

> We will kick things off with two "more theory than practice" chapters on ILE, to give you a solid understanding of what it is based on and how it sort of works.
>
> Then a couple of chapters on /Free, just in case you are not familiar with it, because all of the code we use here will be /Free (it doesn't make sense to use anything else).
>
> This will be followed by three chapters where we spec out actual ILE programs (including a service program) using the /Free we have learned.
>
> Then back to /Free for some oddball, semi-advanced topics that will be especially interesting to those of you who are on newer releases.

Next will be a return to ILE for really advanced topics, mostly related to your new best friend: service programs.

And then, at last, like the final glorious culmination of an epic adventure, MVC.

But enough preliminaries. Are you ready? Great. Let’s get started.

What ~~Is~~ Was OPM?

Did you ever wonder how a program runs? That is, how it starts, how it stops, what rules (other than the code syntax) it follows during its brief and sometimes tragic life.

Me neither. I was always happy enough just to have it finish without my having to look in QEZDEBUG. But the truth is that every program has a "program model" that it follows and that governs just how it does what it does.

For many years, for RPG, this model was called the Original Program Model (OPM), and it did a number of things behind the scenes.

> First, it figured out how to start a program. You know, find a spot in the system where it could run, give it the resources it needed, all that jazz.
>
> Second, for OPM, it controlled the RPG cycle. That's that weird thing that I have never used that will do things (like read files) without you specifically asking it to. Freaky.
>
> Third, it controlled how you called one program from another using the CALL or CALLB opcode.
>
> And fourth, it told the program when to end by using the LR, and it released all resources and cleaned up the party debris.

Problems with OPM

Unfortunately, OPM had a number of built-in problems that got in the way of how we really want to program today. In a nutshell, it was designed to handle one large program at a time rather than a small, multi-module approach. As a result, OPM comes with the seeds of its own destruction sown inside.

Inefficient Calls

> For one thing, even though you could call a second program from a first in OPM, something we really need to be able to do if we are going to switch from a BOP (Big Ol' Program) to smaller, more nimble modules, it didn't do it very well.

The calls were easy enough to make, but every time you called another program, it required a full startup of that new program (opening files, finding space for it, introducing it to the other program. letting them smell each other so they wouldn't fight, etc.) with no sharing of resources.

This takes a lot of time (relatively speaking), and if you end up calling the same program repeatedly from the OPM model, it can add up to some real time and therefore not provide the quick response that you need. Besides, just thinking about the lost time bothers some people immensely.

No Local Variable Support

This is the big one for me: every data element in an OPM program is a global element and can be used everywhere in the program. Lack of local variables is a major difference between RPG and Web languages and makes it very hard for you to build truly modular programs.

What's wrong with global elements? Well, by letting a given data element run wild throughout the entire program, you tend to increase the level of data coupling and so increase the chances of making it possible to change a data element in contradictory ways in several far-flung parts of the program. At the very least, it makes it much harder to track what is happening with that element if you are debugging. It is pretty much agreed by everyone that data coupling is bad—very, very bad.

What you want to do is use data elements that are local rather than global. That is, they are defined and can be used only in a small section of code. But the ability to do this is not available in OPM.

Generally, we refer to this kind of thing as *scope*—the extent or hunting ground of a particular variable. Items that have global scope can be used anywhere in the program, something that might seem convenient but which greatly increases the complexity of a program. Local scope variables can be used in one specific area, and so it is much easier to understand what is happening to that element and how it affects the program.

Scope is something that we in the RPG world don't think much about because in OPM we did not have a program model that allowed local variables. So everything naturally ended up everywhere, and therefore there was only one "scope." But when you get into Web languages, and especially objectoriented (OO) versions, scope becomes a critical concept and one that is carefully monitored.

As we shall see later, in ILE the question of whether a variable is global or local will be set by where in the program it is defined, and once that definition is made, hard and fast rules will control its use. And sometimes rules are good.

Limited Incentive to Set Up Modular Programs

For me, in the end, it's all about the fact that OPM pushes you toward a BOP. OPM just has very few tools for encouraging you to build multi-module applications.

The inefficient calls are one thing.

The lack of local variables is another.

The inability of OPM to handle true procedure-oriented programming is a third.

And the final nail in the coffin is the inability of OPM to support service programs. As we shall see, the main advantage of service programs is not so much just their ability to separately encapsulate source but rather the ability of the service program to insulate the logic in the service program and the programs that use them from changes in either of those entities.

To be honest, I like OPM. It is simple and direct. It is also an evolutionary dead end. And the future is ILE.

It may seem cruel, but in the end IBM decided that a new program model was needed, and they decided to call this new program model ILE.

Don't worry, they didn't kill OPM, and you can still run your programs in the old way if you want to. But there is a time for everything under the sun, and it seems like now is the time to use ILE and let OPM get some much-needed rest.

What Makes ILE Different (and Special)

ILE stands for Integrated Language Environment. It was designed to be just that: a single environment where you could develop and link together programs using a variety of programming languages, sort of a “one environment to find them all, one environment to bind them all” kind of thing but with no dwarves or elves or oliphants (sorry, Sam).

The new model was introduced slowly over a number of operating system releases, starting with C in V2R3 (that was actually before I was born ☺), and adding RPG, COBOL, and CL in V3R1. All of this happened back in the 90s, so you can see that ILE has been around for a while.

In order to list what ILE gives you, I can’t do better than to refer to some of the points outlined by Barbara Morris in her excellent presentation, “ILE Concepts for RPG.” Unfortunately, the Web link to it is no longer active, so you only have me to believe as to its contents. Very convenient, eh?

Modularity

ILE is all about little things: little programs, little pieces of code.

Little sections of code are easier to understand, they are easier to test, and they are less likely to go in the dumper for you. Plus, if you start doing anything in the Web arena, small pieces of code are what they use. Might as well start thinking that way, because most of us are going to end up doing some kind of Web coding.

Where you really see the beauty of small modules is during testing and down the road in maintenance. With less data coupling and fewer lines of code per unit to be tested, testing is faster and more straightforward. And small modules almost guarantee that you will no longer have to spend two days to get into a 20,000-line program before you even start your modifications.

I was struck by a sentence in James Martin’s book, *Free-Format RPG IV, Third Edition* (MC Press, 2015) that talked about how the decision to not bring the `GOTO` into /Free would prevent spaghetti code. And that is certainly true, but there is also spaghetti code that is not created with a `GOTO`, but rather with a plethora of subroutines, all of which use the same data elements. Data spaghetti. Small modules with local variables and low data coupling are the only way to really prevent data spaghetti code. I guess ILE would be considered "ravioli code."

Control

OPM was like model railroading. You put the train on the track and you could have intricate track, but it pretty much just went around the setup. You don't have a lot of control over it, which is why you put your time into building a mountain pass and having a small plastic Bigfoot wave at the train.

But with ILE you can go off-roading. It gives you control over when things start, who hangs with whom (that is, what programs run with what other programs in a common environment), and how things end. It's totally cool. For once in your life, you call the signals. Yeah, baby, you're the man.

Reduced Data Coupling

This point is mentioned above, but it is worth stating on its own.

Because ILE supports local variables (we will see that when we look at sub-procedures), and because things tend to be done in smaller, more focused modules, the data coupling is generally not as extensive as it is in a BOP.

This makes debugging simpler and makes it easier to follow the data flow in the event you need to make mods to this app down the road.

When talking about ILE, the focus seems to get put on parameters and binding and service programs and activation groups. But it all begins with procedures, small bits of code that do stuff. If I could encourage you to do anything, it would be to use local variables as much as you can. I firmly believe that data coupling is at the root of many problems with program design. Reduce that with local variables. Seriously.

Error Handling

I don’t bother too much with error handling myself. I see that as someone else’s problem. Besides, my apps don’t have errors. You know how it is. But it is important to a lot of programmers, so we should consider it.

With OPM, your error-handling options were limited. If an exception occurred that was not specifically handled, all the system could do was send an inquiry message to the system operator asking what to do. What a polite way to say “blow up, end of weekend.”

In ILE, you can actually boil that error up to a higher-level program and have it handled. That is, with ILE if Program 1 has error handling, and it calls Program 2, which does not have error handling, if there is a problem in Program 2, it can be transferred back up to Program 1 to be handled.

Handling errors with ILE can be a whole area on its own and is one that we are not going to deal with. It’s a little more detailed than I want to get into, so I suggest you look at the IBM Redbook (actually, it’s a Red Paper, but a pretty long one) “RPG: Exception and Error Handling,” by Jon Paris, Susan Gantner, Gary Mullen-Schultz, and Paul Tuohy. Or you might try chapter 16 of *Programming in ILE RPG* (now that’s a textbook and a very good one), by Bryan Meyers and Jim Buck, from MC Press. Finally, there is the old standby from IBM, *Who Knew You Could Do That in RPG IV?* by a whole slew of very well-known people, some of whom the FBI has not caught up with yet.

API/Language Access

An added plus is the fact that ILE gives you access to many APIs, including the entire C library.

Now I know what you are thinking. The commonly held belief is that using C causes calluses on your feet and underarms. And frankly, that’s probably true. But there are times when only a C API will do. I can’t think of any offhand, but I also don’t have any calluses on my feet or underarms. It all makes sense if you think about it.

In general, ILE makes accessing a number of other languages easier and more efficient. As we move into the future, we are going to see fewer applications that are just RPG (or CL) and more that involve other languages (including Web ones) as well. But more about that much later.

Better Performance

I am sort of obligated to say this. It is one of Barbara's bullet points, but to be honest, have you ever seen an IBM announcement that touted "slightly poorer performance than in previous releases because of the extra stuff we are doing"? So I find it hard to really get excited about "performance enhancements."

But in this case, I believe it. Mostly because a lot of it is based on vast and technologically unexplainable improvements in the speed with which calls are made. And since ILE relies heavily on calling procedures, it is right up ILE's alley.

Why Bother with ILE?

I am not talking here about why you should bother with ILE from the point of view of your company. If you are interested in that, just Google "advantages of ILE," and you will get links to dozens of articles.

No, what I am talking about here is why *you* should bother with it. You personally, that is.

After all, I am asking you to embrace a new set of tools, to give up the comfortable paths of the past and spend some time and energy learning something new. You've been doing RPG for quite some time now, probably without using any or much ILE, and things aren't going too bad. Why should you change? It's a reasonable question.

For you, of course, I can't answer. I know for myself that I just can't understand a professional outlook that says, "I am OK using a tool that is 30 or 40 years old rather than a newer one that actually works better." And I can't understand the concept of a professional who does not put learning new stuff near the top of his or her work "to do" list.

I guess I feel it is my obligation, my responsibility as a professional, to keep moving forward. To stop and just let my profession go on without me is, in my opinion, the very definition of unprofessional. But, that's just me.

ILE provides you with the tools you need to write modular code that can be efficiently accessed from other modular pieces of code. Modularity gives you a number of advantages: easier to code, easier to understand, easier to test, and easier to modify down the road in the unlikely event that becomes necessary. And efficiency is always a good thing, especially when the apps you are writing may be used not just in the green-screen world but on the Web as well. Yes, it will take a bit of time and some work to get proficient with these techniques. But that's what life is for—to learn new things. So why not ILE?

Just Keep This in Mind

But, at the same time, there is one thing I want you to keep in mind.

Anytime we adopt a new technique, we sort of automatically expect it to give us a big productivity boost right from the start. And I think that's pretty much true of /Free. But it won't happen so quickly with ILE.

Part of the problem here is how we define "productivity." For the most part, we do it from a very short time frame point of view. It is all about how many lines of code we can produce in a day or an hour. But that is not what productivity is all about.

Take this book, for example. Am I more productive if I write a chapter in two hours and then revise it 10 times before I am finished, or if I write it in six hours and only revise it twice?

Now you might say that it depends on how much time I spend on each revision, but in reality that doesn't matter. I am more productive if I have to pick a chapter up fewer times because the longer I work on it at one time, the more connected are my thoughts. If I revise it 10 times, that is 10 times I have to drop what I am doing and switch my thoughts back to that particular chapter.

And programming is similar, at least in terms of keeping your focus on one thing rather than switching it in and out all the time. ILE does not give instantaneous productivity increases, but it does so eventually. More important than productivity, however, which is always hard to measure, is the quality of the code you produce. With smaller modules that have less complicated code and do one particular thing, there are fewer bugs, and those that do crop up are easier to fix.

If you are new to ILE, it is quite possible that your first reaction is going to be "this is more work; it is not as efficient." But remember, all new things seem less efficient at first. If you return to OPM at that point, you are missing a great opportunity to grow professionally not only in terms of the tools you are familiar with but also with the quality of the code you produce. Hang in there, and it will get easier. Hang in there, and it will become more natural.

What Makes a Program ILE?

OK, finally, something practical. Here's the deal with ILE. It all starts with the compile command and the activation groups.

Both ILE and OPM use the same RPG compiler, but it has been enhanced, and some additional commands have been added to the operating system to modify how the compiler works. That is, OPM and ILE share the same compiler but *not* the same compiler commands. We are going to look at three separate compiler commands: `CRTRPGPGM` (Create RPG Program), `CRTBNDRPG` (Create Bound RPG), and the `CRTRPGMOD`/`CRTPGM` (Create RPG Module/Create Program) combo. Each is a little different in terms of how it handles OPM and ILE.

CRTRPGPGM

If you use the `CRTRPGPGM` command, the object that will be created is OPM. It has no ILE capability and will not work for a program that is stored in `QRPGLESRC`. Needless to say, we will not be using this command in this book, and you shouldn't use it either. 'Nough said.

CRTBNDRPG

If you use `CRTBNDRPG`, then the result may be an ILE program. Why "may be?" Because the ILE question with this compile command depends on the `DFTACTGRP` parameter that is present with this command but is missing from the `CRTRPGPGM` command mentioned previously.

If you specify `DFTACTGRP` to be `*YES`, then the system will compile that program using the default activation group, which means it will be an OPM program. Keep this firmly in mind. The default activation group belongs to OPM, and vice versa. It is not a real activation group.

But if you say *NO, then you will create an ILE program. Another field (ACTGRP) will then magically appear where you can specify the name of the activation group you want to use. The default is QILE, but we will see in the chapter on activation groups (chapter 17) how that is probably not the best choice. And you will want to pay attention to the activation group that you assign (lots more about that in chapter 17). It's very important.

CRTRPGMOD/CRTPGM

Finally, if you use the CRTRPGMOD and CRTPGM command combo, then the result will always be ILE. There is no DFTACTGRP parm in either of these commands; it is assumed that value is *NO; the command does not give you the option to change it. Then in CRTPGM you can set the specific activation group you want to use via the ACTGRP parm.

So the question is: based on the version of the software that you are on, what command comes up when you do an option 14 in Programming Development Manager (PDM) when you are in QRPGLESRC? Is it CRTRPGPGM? Well, it won't be because since this command only produces OPM objects, it is not available from QRPGLESRC.

And if you get CRTBNDRPG with an option 14, then the question is, what is the value for the DFTACTGRP command? If it is *NO, then you are all set for ILE. If not, then you will need to manually override at compile time or else use an H-spec in your program. Of course, you could always change the default value in the CRTBNDRPG command. That's what I would do.

QRPGSRC/QRPGLESRC

What about QRPGSRC and QRPGLESRC? They are nice and all, but the record length difference is not magic. Nor is the difference between program type RPG and RPGLE. You can create an RPGLE program in QRPGSRC, although I don't recommend it. And, by using source that is in QRPGLESRC and the CRTBNDRPG command with a DFTACTGRP of *YES, you can create what is essentially an OPM version of an RPGLE program.

The system does key in on the source type (RPG or RPGLE) to determine what compile command should be the default command applied (CRTRPGPGM or CRTBNDRPG). And the source type is set based on whether you are in QRPGSRC (RPG) or QRPGLESRC (RPGLE). So, that's something. Plus, we already saw that CRTRPGPGM would not run on a program stored in QRPGLESRC. The point is, however, QRPGLESRC is not magic, and there are other things you have to do, as we noted earlier. Just being in QRPGLESRC does not mean that you are doing ILE. It means you are ILE eligible. The rest is up to you. At the same time, you should put your ILE programs in QRPGLESRC, if for no other reason than the fact that it helps prevent confusion and prevent you from using CRTRPGPGM.

In the end, the *only* thing that determines whether or not your program is ILE is the value of the DFTACTGRP flag. *YES means OPM, *NO means ILE.

Also please remember that compiling your program as ILE does nothing per se, either. It does not rewrite your program in any way to take advantage of what ILE offers. All it does is open the gate. It makes using ILE functionality possible. But it will still be up to you to write your code so that you take advantage of this availability.

Now It's Your Turn

What? Do you need an invitation? Fine. Please, at this point, go out and do a "14" with a prompt and see what command and parms are attached to it. Will you do ILE by default or OPM?

While you are at it, you might as well do 15, too, and verify that it brings up CRTRPGMOD.

Of course, there is no number for CRTPGM. You have to key in all six letters of that yourself. You betcha. Your Dad was right. Life is rough.

What Ya Shoulda Learned

Not much practical so far, but we are just starting to get a feel for what ILE is. At this point, you should know a couple of things at least.

First, how the OPM model controlled how a program ran pre-ILE.

Second, what the problems were with that model and how they were at variance with the way people want to program today.

Third, how ILE uses a different program model to allow people to code in a more modern way.

Fourth, hopefully as a result of the previous three items you will feel you can explain to others the advantages of ILE and how it can help them develop even better, cleaner, more maintainable code.

And finally, we looked at the switch that turns OPM off and ILE on (the `DFTACTGRP` parameter on the `CRTBNDRPG` command) and took a look at what the defaults are for your system.

There's more that I want to say about ILE from a conceptual point of view, and by golly, I think I will. Get set for another exciting chapter.