Chapter 10

A Summary of Your ILE Options

All right now, let's pick ourselves up off the floor. That's right, dust yourself off a bit. Make sure your shirt buttons are lined up straight. OK, smooth down that hair, assuming you have some. Come on, look sharp.

Before we go on and return to /Free for what might be considered "advanced" topics, let's take a minute and go back over what we have learned in the last three chapters. And let's start with that question. What have we learned here, and what do we want to be sure we remember?

Review of the Three Options

Let's start by going back to the three program examples we looked at in chapters 7, 8, and 9.

Sub-Procedures within a Program

This was what we did in chapter 7, and, as you well know, I heartily applaud the use of sub-procedures instead of subroutines (local variables, yada, yada, yada).

To be frank, though, ILE is about a lot more than just replacing subroutines with sub-procedures. True, we are using sub-procedures, but we still end up with one big program (large execution size and probably high complexity). And just putting sub-procedures into a large program is not what MVC is all about. I like what sub-procedures brings to the table, but just using them in place of subroutines is probably the least real ILE way to do things.

That said, it does work out just fine for places where you have a simple program that did have some complex and/or repetitive logic in it, but not for a bigger app.

Obviously, with this approach you want to use `CRTBNDRPG` to create the program.

Program to Program Call

This was the crux of chapter 8, and it gave us a good look at using the prototyped call and the associated `PR` and `PI` D-specs to go from one program to another.

This is still not hard-core ILE, but is probably a better way to go (than just doing embedded sub-procedures in a big program, that is). First, you are surely reducing the size of your program by spinning logic off into separate modules. Granted you now have more pieces to keep an eye on, but each piece should be simple and easy to understand or modify. This also sets you up to do MVC, which we will see a little later on.

Of course, much of the efficiency that is generated here will be dependent on how the modules are related. Are you going to just call them? Are you going to bind them together? Are you doing this in one step or two? What about the activation group scheme you will use? Lots of open questions, and we will grapple with some of those as we go forward.

In chapter 8 we treated each program as a separate entity, compiling both with separate CRTBNDRPG commands. But you don't have to do it that way.

For more efficiency, you could compile both as modules with CRTRPGMOD and then use CRTPGM to tie them together. Just remember that if you use CRTBNDRPG and CRTPGM to tie these separate programs together, you will get faster response, but since the bind happens when the programs are compiled, any change to one means a recreate of the whole thing. This can be an issue if the one being changed is the called program and it is used in lots of places.

Either way you compile, the decision is not a slam dunk, and you need to think about the characteristics of the relationship.

Service Programs

The final option (chapter 9) is to put our logic into sub-procedures (not standalone programs) and then organize them into one or more service programs (rather than CRTRPGMOD/CRTPGM standalone programs).

As we saw before, it is up to you how you organize your sub-procedures in the service programs. Sometimes you may want to do it around a master file. Or you might want to do it around a business issue or critical task (like credit checking and approval). It's up to you. Point is, I don't really care, but you should.

Again, you have small modules, the sub-procedures, and fast access because there is a binding between the calling program and the service program. You also get a fair degree of module independence because if you set up your service program right, you can change the sub-procedures in the service program without having to recompile the programs that access that sub-procedure. We will talk more about this later (binding language).

And the Winner Is?

So, what's the difference? How do you decide which one you would want to use?

And the answer is, it depends.

One thing I don't want you to do is look at the three previously mentioned options as some sort of hierarchy, where you start with a program that has subprocedures but the best way to do it is with service programs. That is not the case. Although, maybe it is ....

What is best depends on what the application is like. And it also depends on when you want the binding to occur. And how likely it is that there will be changes in the modules once they are set. And that is the strength of ILE. You never had those options in OPM. Now you can decide what strategy is best for your particular situation.

So what do you think? You know your situation. Your environment. Your development needs. What is the best way for you to proceed? Not just for the three program examples but for using ILE in general.

It's easy to just give this some cursory thought and then turn the page and go on, but I want you to take a tooth-brushing "minute" (sing "Happy Birthday" three times) to give it some serious thought. Are you ready to start getting into /Free ILE?

If not, why not? What are you waiting for? What are the roadblocks in your way, and how can they be removed or mitigated?

If yes, then where shall we start? I assume you wrote the example programs that we went over, so you are ready to move forward on some real work. What do you want to start with? Service programs? Changing a subroutine over to a subprocedure? What makes sense to you?

You have several weapons to choose from, so take a minute or two now to start an ongoing dialog with yourself and your teammates over how you want this whole thing to go down. And then get going on it. Yes, there is more to learn, but you know enough now to be more than dangerous. Don't let what you've learned remain theoretical. Get practical and hands-on as soon as possible.

A Few Thoughts

Of course, I wouldn't be surprised if right now you are a little confused. After all, with OPM you put code in a source member, compiled it, and you were done. There is a certain charm to the simplicity.

But with ILE we have procedures, modules, programs, and service programs. Plus goodness knows what I haven't told you about yet.

We already know there are three different ways in which programs can be built: embedded sub-procedures, program-to-program calls, and service programs.

Finally, on top of that there are two ways to compile: the one-step and the two-step method. (Of course, when you are dealing with service programs, it is a four-step process: two steps to create the service program and then two steps to create the calling program and bind the service program to it.)

Some people look at this and say "ILE is too complex." And that is too bad, because like the glass half empty or half full, you can look at ILE as either being too complex or providing extraordinary flexibility.

But that still leaves you with the question: how do I get my hands around this? What strategy is best for me? Please allow me to offer you some additional things to think about.

It's Not Just About Efficiency

First, too often ILE gets caught up in efficiency concerns: that it is fast, and everything has to be decided based on what the fastest connection is between two modules.

But for me, efficiency is usually a secondary concern. Most of the time, you are not calling one module 10,000 times a minute from another module. And even if you do call something a fair number of times, today's processors have power to spare. Your app is unlikely to have slow response time as its number-one problem.

Write Modular Code

I believe the essence of ILE is modularity. The essence of ILE is about creating small, easily understandable modules, and then combining them in whatever way seems to make the most sense given how often the modules might change and how fast you realistically need the connection to be. Know what I mean, Vern?

Bottom line: if you really want to participate in the ILE revolution, start writing modular code. We will emphasize this when we talk about MVC later on, but that's not the only game in town, and there are many patterns you can follow in writing modular code.

Embrace Binding

Too often in the IBM i world, we think of programs as being solitary things that live alone, like rogue bull elephants.

But togetherness is a hallmark of ILE, and so you want to always consider binding modules together when you are building an app.

There are a couple of ways to do this: using `CRTRPGMOD`/`CRTPGM` at compile time and via a service program at run time—and the one that works best for you is the one that most closely resembles your app.

Use Common Sense in Putting Things Together

Once you do that, then you can use common sense to put the modules together.

Is one module going to change frequently? If so, you might want to avoid a binding that happens at compile time.

Is the sub-procedure you are building a one-of-a-kind thing that will never be used anywhere else (I know never say never, but sometimes you can be pretty sure, and you have to draw some lines somewhere)? If so, then do you really want to bother with a service program? You want to use a sub-procedure because of the modularity it provides, but there are worse things than just including those sub-procedures in the program where they are used.

Is ILE Too Complicated?

It’s a fair question. Especially after all the options that we have seen so far, ILE looks kind of complex, doesn’t it?

And to that I say, uh, yeah, maybe. What’s your point?

If you look closely at ILE, there are some things that probably could have been designed to be less surprising by IBM. But they didn’t. And that is all there is to it. Time to move on.

Seriously though, when you think about it, and compare it to some of the things that go on in other languages (like PHP or Ruby or Java), it’s seriously not that bad. I mean anyone who looks at ILE and says, “Oh, it’s just too complex for me” is probably not giving themselves enough credit.

Yes, there is some complexity there, but most of it is born out of the fact that it’s not the one-size-fits-all world of OPM. Instead, you have the flexibility to do things a variety of ways, and it is up to you to choose the way that works best for you and your situation.

What Ya Shoulda Learned

Granted, this chapter has mostly been review, but there is still plenty to think about.

- Describe, to yourself or to anyone who will listen, perhaps a loved one who has no choice, the three ILE structures that we outlined in the chapters previous to this.
- Think about the requirements you currently have facing you. How many would be good candidates for using ILE? Be honest now. Don't just say that nothing fits because your needs are special. ILE techniques can fit into a wide variety of situations. Frankly, everything is a candidate for ILE.
- For each situation you can think of, what is the best ILE approach to use?
- Lay out a plan, at least in your mind, for how you will get from where you are to where you are using those ILE options.

Seriously, think about it. The next move is up to you.