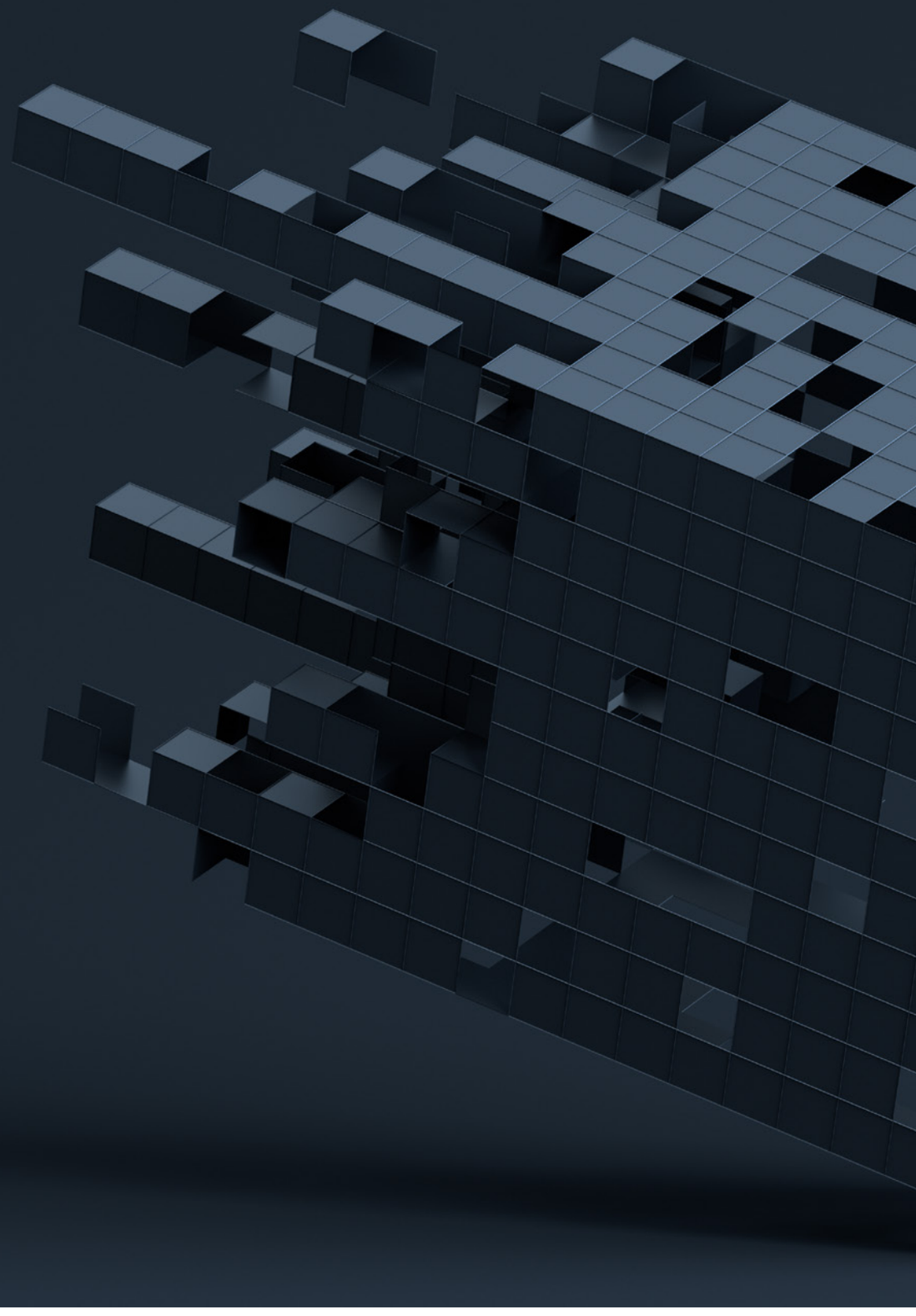
TechChannel

Using Modern IBM i Tools

→ 3 IBM i modernization success stories
→ 5 reasons Susan Gantner loves RDi
→ Jon Paris on why RPG is relevant



SPONSORED BY LANSA MIDRANGE DYNAMICS providing innovative IBM i solutions Profound Logic

contents

Modernizing From a Strong Foundation

"If you're buying an older house, buy one with good bones"—so went the advice I received while house hunting a few years back. I did, and it's amazing how modern an early 1940s build with a strong structure can look after some paint, updated fixtures and new hardware. Of course, this modernization was only possible because the house had a good foundation in place—and these days, I'd like to think you'd never guess how old it really is.

Similarly, RPG's strong foundation is what makes it such a modern tool. Today's RPG is barely recognizable as its earlier version—in format, syntax, capability and flexibility. It evolved along with programmer needs into a fully free-form language that modern programmers can easily pick up. And RDi is the only editor that supports all modern RPG syntax and features.

Similarly, one of the IBM i platform's greatest strengths is that you can run unchanged the same code you may have been running over 30 years ago.

Of course, effective modernization still calls for maintenance, planning and strategy—but the fact is, good bones make the process possible. There's a reason RPG and the IBM i platform have been around all these years; they continute to evolve to meet current business needs.

Keelia Estrada Moeller, Managing Editor

"

Today's RPG is barely recognizable as its earlier version—in format, syntax, capability and flexibility.

"

3 IBM i Modernization Success Stories

Learn how May Trucking Company, Concordance Healthcare Solutions and Holy Name Medical Center modernized their legacy systems to innovate with IBM i

IBM i clients recognize the need to modernize, and with the speed and reliability of Power Systems hardware and the versatility of IBM i, they're accomplishing some incredible things. What follows is a quick summary of three IBM i clients that have modified their legacy systems and/or core applications. There's still much work to be done, but rest assured, a lot of success stories have already been written.

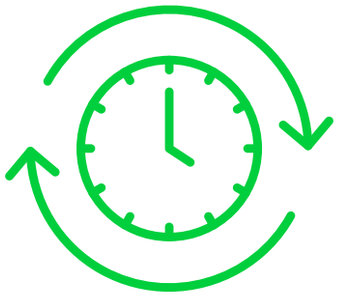
When it comes to application modernization, the only mistake is to do nothing.

Becki Wagoner, vice president of strategic innovation and technology, May Trucking

May Trucking

For many years, May Trucking Company felt its IT environment suited the company's needs. Their philosophy about their tech is one that many organizations hold. In the words of Becki Wagoner, May Trucking's vice president of strategic innovation and technology: "If it's not broken, why fix it?"

PHOTOGRAPHY BY EVAN KAUFMAN



Through the use of a hosted virtual tape library, May Trucking is saving an estimated 500 hours of downtime per year.

But after relying upon a green-screen based management system purchased in 1993 and a Power Systems 520 server that constantly ran near or at capacity, Wagoner concluded that the family-owned company needed to modernize its core application and operating environment.

Working with Profound Logic and iTech Solutions Group, May Trucking installed two Power Systems servers—an S814 and an S820—at its Salem, Oregon, headquarters. Profound Logic provided key migration tools and helped employees get up to speed with the new GUI software that featured enhanced back-end functionality. iTech handled the migration itself, smoothly moving the environment to the IBM i 7.2 OS from IBM i 5.4, which was released in 2008. “iTech spun up a virtual partition on a box they were hosting and I sent a tape of my full backup to load. We ran testing on everything running under 7.2 for almost two months. We had every department test to ensure there weren’t any major hiccups,” says Wagoner.

As a result of this transition, CPU capacity has dropped from more than 90% on the 520 to 20% or so on the S814. At that point the company began live replication of the S814 in near-real time to an iTech-hosted Power Systems S822. In addition, virtual backup tapes were backed up to an iTech-supported cloud environment. Through the use of a hosted virtual tape library, May Trucking is saving an estimated 500 hours of downtime per year. Profound Logic tools provide the interface for the new management systems along with other low-level improvements, such as the integration of Node.js.

Wagoner and May Trucking Company are delighted to operate in a more modern computing environment. Processes run faster, productivity has improved, and the availability of near-real time backup and disaster recovery provides peace of mind.

READ THE FULL STORY

Concordance Healthcare Solutions

When three previously independently owned and operated medical supply companies merged to create Concordance Healthcare Solutions in 2016, their IT platforms and solutions also had to be merged. To accomplish this, the newly formed company consolidated onto an IBM POWER9 server and a unified ERP environment that impacts every aspect of the business.

Concordance chose to upgrade to an IBM Power Systems S914 server and repurpose an existing Power Systems S814 server as an offsite high-availability box. Business partner UCG Technologies Inc. provided the sizing and configuration advice for both systems.

During this process, the company also opted to standardize on a warehousing solution to help workers throughout its office and distribution centers. Concordance's S914 is running 20-30% CPU utilization on an average day, whereas the organization was utilizing up to 50-60% on its previous system. The decision to migrate everyone to S2K Enterprise (which was selected as the operating platform of choice for Concordance), allowed the company to standardize across all departments.

Concordance now has a near-nationwide presence with 20 strategically located distribution centers across the U.S., enabling it to serve 67% of the nation's healthcare systems.

Keith Price, CIO and COO, Concordance Healthcare

READ THE FULL STORY

Meanwhile, the unification of systems proved particularly beneficial when COVID-19 struck and Concordance had to modify its business model. Rather than operating as usual, it shifted its focus to the distribution of personal protective equipment (PPE) needed by frontline medical workers, including N95 masks, gowns and gloves.

As Keith Price, Concordance CIO and COO, explains, during this grueling time, his supporting hardware and software infrastructure was the one thing he didn't worry about. It functioned as designed, as a unified solution that touched every part of the business.

PHOTOGRAPHY BY MICHAEL NEMETH

Price credits the employees who used those systems to make sure everything ran as smoothly as possible. “A lot of people put in many hours to do what needed to be done to take care of healthcare workers and their patients, to make sure that they had the proper supplies, and I’m very proud of them,” he says. “But we had it easy compared to the healthcare workers dealing with thousands of patients every day. Our days were long and hectic, but I just can’t imagine what those people went through.”

Holy Name Medical Center

Holy Name Medical Center is a 361-bed acute-care facility that staffs almost 1,000 physicians. An organization of this size requires a robust hardware and software infrastructure. In this case, that includes two active IBM Power Systems servers—a production 824 and a backup 750—with the 750 acting as an IBM PowerHA mirror of the 824.

CIO Mike Skvarenina, who's worked in midrange environments since the days of the IBM System/38, was initially hired to be a programming manager in 1992. Skvarenina and Holy Name continue to stick with the platform, seeing it as the linchpin in the overall operations.

In 1999, the organization began modernizing its key application. Screen-scraping technologies of that time could take a 5250 green screen and allow it to run in a web browser, but Skvarenina sought a high-end user experience. He wrote a few experimental programs and, once convinced of their viability, tasked his lead developers to begin writing a menu system and the beginnings of the hospital's electronic medical record. He coined this new interface/system WebHIS.

By developing its own system, Holy Name experienced a significant cost savings. Today, WebHIS comprises more than 500 RPG programs that facilitate patient care. Clinicians compliment Skvarenina on WebHIS frequently, saying it's much easier to use than the systems at other area hospitals.

However, Skvarenina had concerns about future RPG CGI development and application maintenance. He says RPG skills are declining, largely because experienced developers are retiring and few schools teach RPG. With this in mind, Holy Name is shifting to a bimodal application development model. This involves RPG application support and the development of new PHP apps, APIs and the eventual rewrite of all of its RPG code to PHP or another, more accessible language. With this approach, the organization can continue using its existing and proven medical information system while also setting itself up for the future.

Whatever the future brings, Skvarenina says the application will likely still run on the Power Systems platform. Why? Reliability and performance, for starters. "The platform is incredible. The OS is bulletproof. And the database is fantastic," he says. "We like that it supports hundreds of users with sub-second response times. We just like it."

READ THE FULL STORY

PHOTOGRAPHY BY BILL BERNSTEIN

My 5 Favorite RDi Productivity Features

Editing functionality, understanding and navigating code are among the reasons RDi tops the green screen for RPG development

BY SUSAN GANTNER

Throughout RPG's long history, the primary tool for editing and compiling code has been green screen and text based.

But today's modern RPG is barely recognizable as the same language as its earlier versions—not only in format and syntax, but also in capability and flexibility.

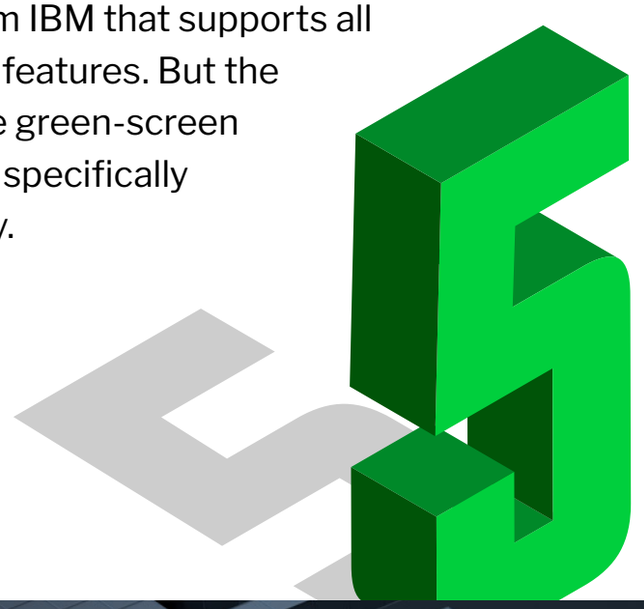
The modern toolset IBM introduced is the only one updated for the new syntax and features of RPG. Even more significantly, it optimizes code development to allow developers to get their jobs done faster and better. While still available, the deprecated green-screen tools have not been enhanced in over 10 years, and don't even recognize the current syntax and features of today's IBM i languages.

The development tool for modern RPG applications is Rational Developer for i (RDi). Just as today's modern syntax makes it easier for developers with no RPG experience to quickly jump into the language, RDi allows developers familiar with many other IDEs to more quickly learn, edit and compile IBM i applications.

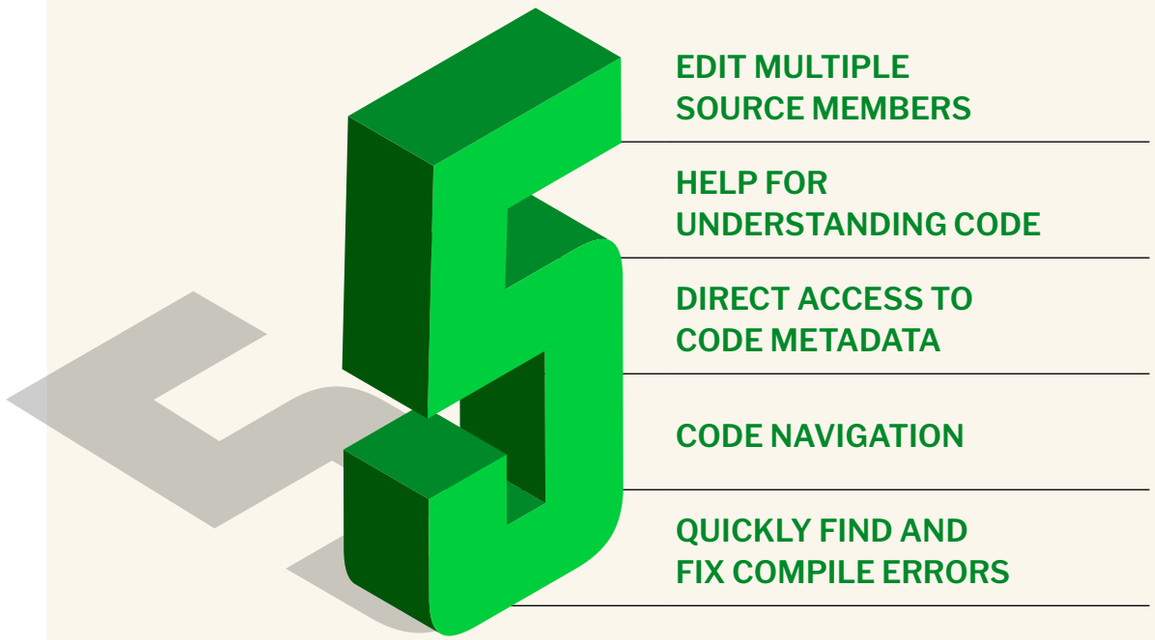
5 Favorite RDi Features

Lest you think RDi only appeals to new RPGers, I've been coding in RPG for decades and I'm addicted to RDi. Why?

Of course, as previously noted, one obvious reason is that RDi is the only editor from IBM that supports all modern RPG syntax and features. But the best reasons to leave the green-screen tools behind relate more specifically to improving productivity.



Here are five of my favorite RDi productivity features:



1. **Edit multiple source members**. I can (and nearly always do) have multiple source members open for edit at the same time using only one host job connection. Today's modular coding styles make this even more vital. I can switch between open members in full screen or I can split my edit screen—either vertically or horizontally with multiple splits if needed so that I can see and edit several members at the same time. When editing large programs, I can split my screen to see multiple views of different parts of the same program.

2. **Help for understanding code**. The RDi editor makes it faster to read and understand code by using color to highlight different parts of statements. This function is particularly helpful with free-form RPG as well as other free-form languages like CL. Different colors for RPG operations, CL commands, built-in functions, keywords, parameters, etc., make it faster to grasp the meaning of statements at a glance.

   For occasions when I am forced to work with older-style columnar RPG code where indenting nested logic is impossible, multiple tools are available to help me decipher deeply nested logic. I can choose to open an indented view of the source or I can ask for arrows to be drawn in the editor outlining the nesting levels in the code.

   When working with free-form RPG, I can establish my own preferences for indentation of nested logic. This controls indentation used for new logic that I add and allows me to request the editor to reformat blocks of code to ensure accurate indentation for nested logic based on shop standards.

   Of course, the mere fact that I can typically see from 2-4x the number of lines of code on the screen at once compared to the green screen also aids in understanding complex logic.

3. **Direct access to code metadata**. This is related to the previous point, because you can't fully understand logic until you know the details about the data that logic uses. In RDi, I can hover over any variable in my code—whether defined it's internally or externally—to see

details of its definition, including any comments related to it and any significant context of the definition (e.g., the data structure where it was defined, the external file and record format where its definition originated, etc.). For called subroutines or procedures, hovering over the names shows comments and/or details of the parameters and/or values returned by the called code.

If I need more information than the hover text offers, I have an integrated outline of my program. From there, I can see additional information, such as a cross-reference showing every line of code where a particular data item is referenced, including whether the data is updated there. The outline also contains lists of all files, variables and structures. And cross-reference data

shows lines of code from which a specific subroutine or procedure/function is called.

4. **Code navigation**. As helpful as seeing information about data definitions or called routines is, occasions always arise where I need to actually view and potentially edit the code or definition. The program outline previously mentioned can also be used as a navigation tool. Clicking on the name of a data item, subroutine or procedure will position the editor to that definition (although, I'll describe shortly an even faster way to get there). Likewise, clicking on one of the statements where the item is used in the outline's cross reference will navigate to that line of code.

My personal favorite navigation tool is one that not only takes me somewhere, but also gives me an equally quick way to return to where I had been before I made that detour. This is done via two of the many keyboard shortcuts available. When I position my cursor on the name of a subroutine or procedure in the logic where it's being called, I can press F3 to jump directly to that routine in the editor—no need to use the outline to get there. But the most exciting thing about this feature is that I can also use a shortcut to just as easily make the return trip back to where I was originally. On a Windows installation of RDi, that shortcut is Alt + Left_arrow; on a Mac (which I use) it's command + option + Left_arrow (or ⌘ + ⌥ + Left_arrow.) This shortcut also works to navigate to and from data definitions.

5. **Quickly finding/fixing compile errors**. The green-screen editor doesn't help me with my errors if my compile fails. I must leave the editor, look for the message about the compile, then use various techniques to locate and open the compile listing and position to the list of errors in the listing. After that, I'm on my own to navigate to the appropriate line of code.

With RDi, I request a compile with my source still open in the editor. Any compile errors automatically appear in the integrated error list. A simple double click on each error positions me directly where I need to be without leaving the editor or opening a compile listing.

RDi Addicts Unite

I could (and often do) go on and on about the reasons for my addiction to RDi. You can find more information in an article Jon Paris and I wrote in "Why RPG Developers are Adopting RDi." We're far from the only RDi addicts. There's a mailing list of RDi users on midrange.com who regularly share information and tips. A while back we published on our blog part of a post from a fellow RDi enthusiast, Buck Calabro. You can read his reasons in "SEU Was Holding Me Back."

↗	**WANT TO KNOW MORE ABOUT THE JOYS OF USING RDi?**
Just ask by emailing me at jonandsusan@partner400.com.	

Why RPG Remains Relevant and Vital

The programming language provides a great base for modernization efforts

BY JON PARIS

These days many IBM i shops are faced with a familiar problem—their RPG developers are retiring and they're finding it hard to hire replacements.

In my opinion, their problem is that they're looking for the wrong thing. They don't need RPG developers. They need developers. Specifically, developers who want to build business applications, not the next iteration of Facebook or Instagram.

At this point you may be thinking that modern programmers don't want to program in RPG. And, if I was talking about the type of RPG that some shops still run (i.e., fixed-form columnar code, not significantly different from what they ran on the System/36 or System/38 some 30+ years ago), then I would have to agree. Heck, I don't want to program in that old stuff either.

Luckily, I don't have to, because I'm talking about today's RPG, which is a fully free-form language that will look so familiar to today's programmers that few will have any difficulty picking it up.

How Did We Get Here?

One of the greatest *strengths* of the IBM i platform is that you can run unchanged the same code you were running 30-plus years ago.

One of the greatest weaknesses of the IBM i platform, is that you can run unchanged the same code you were running 30-plus years ago.

This has led to a situation where many shops adhere strictly to a mantra founded on the notion "if it ain't broke don't fix it." The result is that their code looks much the same as it did 30-plus years ago.

Even when developers want to embrace more modern standards, they sometimes find themselves thwarted by management that thinks they need to keep a senior developer—who's just too lazy to learn new techniques—happy.

Because underlying system requirements are constantly changing, developers on other platforms, Windows for example, rarely have the luxury of just letting things be. As a result, it's not uncommon for them to have to fix perfectly good programs simply to keep the code running on new releases. This does result in a continual evolution and ongoing modernization of their code base. It may not have happened for the right reasons but at least it happened. Of course, it also means that the number of developers required tends to be out of all proportion to the amount of work being performed, but I digress.

When our clients ask for our advice on modernizing their code base, we often suggest that they adopt a policy that calls for programs to be bought up to date whenever significant maintenance is to be performed on them. The definition of "significant" of course varies from shop to shop, but certainly anything that requires substantial testing is an immediate candidate. This results in a more modern and more maintainable code base that requires less knowledge of the more esoteric aspects of the old RPG language and is easier for new RPGers to come to grips with.

You can find more about some of our experiences in this area in this article on replacing RPG developers.

How Much Work Is Involved in 'Modernizing the Code Base?'

Five major flavors of RPG are out there in the wild:

1. RPG II (i.e., S/36 heritage)
2. RPG III code (i.e., RPG/400)
3. Fixed-form RPG IV (RPGLE)
4. RPG IV—free-form logic only
5. RPG IV—completely free-form

The amount of work needed to make the code current and comprehensible to new RPG programmers (i.e., to reach the level of #5), depends on which of these we are talking about.

RPG II code is often the hardest to deal with and I would normally suggest a rewrite rather than conversion as significant manual effort can be required even when tools are employed to do the conversion.

There's no excuse for any RPG III code to still exist. None. The conversion from RPG III to fixed-form RPG IV is straightforward and the minor incompatibility issues are well known and well understood. It has been, after all, over 25 years since RPG IV was first released! Sadly a few software vendors have allowed their products to languish in this state.

To go from fixed-form RPG IV to free-form is a job best left to a tool and a number of excellent ones are available ranging in cost from free to inexpensive. I have personally used three different tools: Linoma's RPG Toolbox, Arcad's Transformer and Craig Rudledge's free conversion tools. All do a good job. The better tools can also take care of any incompatibilities and ensure solid working code. One tool (Transformer) can even convert any stray GO TO operations that it encounters

To go from fixed-form declarations to free-form is a simple process and the same group of tools can be applied with 100% compatibility of operation.

Does this give us modern RPG? No, but it gives us code that is far easier to understand and maintain and that can form an excellent modernization base. New RPGers will have no problem understanding it and performing basic maintenance tasks after a very short time. My partner Susan Gantner and I, together with colleagues such as Paul Tuohy, have done many "new to IBM i and RPG" training sessions in recent years. We have trained Java, C#, C++, .Net, PHP and Python programmers in the joys of modern RPG. Most adopt it easily and happily.

Developers with an orientation toward business programming delight in the simplicity RPG offers when coding business calculations. RPG's native understanding of currency values (for example) makes life so easy when compared to one that requires the use of function libraries to produce accurate results even in simple quantity-times-price type calculations.

Evolving your code base in this fashion is increasingly being recognized as a viable approach to modernization. To get some idea of just how effective it can be, take a look at this recent report from Micro Focus and the Standish Group. You will see what we mean when we say modernizing your RPG code is a critical step in such an evolution.

Is RPG Really a Modern Language?

Today's RPG is certainly not the language that we older folks learned. It has come a long, long way from the stodgy old columnar language that its critics like to paint it as.

In terms of what one might typically expect from a modern programming language, about the only major feature that's missing is object orientation (OO). Frankly I don't think this is necessarily a bad thing. OO works well in situations such as building GUIs but has far less utility in business processes.

So, what are some of the modern features of today's RPG that its detractors like to remain blissfully unaware of? The list includes:

- Embedded SQL
- A comprehensive (and constantly growing) set of built-in-functions

RPG HAS MANY UNIQUE FEATURES THAT MAKE IT IDEAL FOR DEVELOPING DATA-CENTRIC BUSINESS APPLICATIONS.

- Ability to create and utilize custom function libraries
- Ability to directly tap into C and Java functions
- Built-in functionality for processing XML
- Files as parameters
- Procedure overloading
- Dynamic arrays
- Nested data structures
- Built-in support for processing and generating structured data such as JSON

And this does not include the growing base of open-source RPG offerings that simplify the provision and consumption of web services and other essentials of modern application development.

RPG: Not Just Another Language

RPG has many unique features that make it ideal for developing data-centric business applications. It goes against the grain for me to focus on RPG as just another language. But my mission here was to address two main points:

1. Today's RPG is similar to more ubiquitous languages such as PHP, Python, C#, etc., so much so that new developers pick it up easily. And, once they have learned modern RPG it is far easier for them to understand the older versions.

2. It’s both essential and relatively simple to modernize existing RPG code. This not only makes it easier for new RPGers to contribute but provides a stronger, more flexible base for further modernization of the applications as time goes on.

This e-book was published by

TechChannel

901 N. 3rd St., Suite 195, Minneapolis, MN 55401 // (612) 339-7571

Staff list

Publisher: Mari Adamson-Bray

Senior Content Director: Evelyn Hoover

Managing Editor: Keelia Estrada Moeller

Art Director: Jill Adler

Project Manager: Noelle Heaslip

Audience Development Director: Linda Holm

Account Executive: Kathy Ingulsrud
(612) 313-1785 // kingulsrud@techchannel.com

Account Executive: Nicole Johann
(612) 336-7675 // njohann@techchannel.com

Account Executive: Darryl Rowell
(612) 313-1781 // drowell@techchannel.com

TechChannel

TechChannel.com is home to a variety of content
to help you get started on your modernization journey.

Learn more now

SPONSORED BY

LANSA

MIDRANGE DYNAMICS
providing innovative IBM i solutions

Profound Logic