

HEAD: Microsoft and IBM Play Nice

DECK: An introduction to .NET technology for the System i developer

BYLINE: By Richard J. Schoen

Welcome to the world of Microsoft* .NET technology. You're probably not used to seeing the word Microsoft in an IBM publication. However, as you continue reading, I think you'll find that Microsoft and IBM can play well together in the IBM* System i* environment. This article introduces .NET technology and provides an overview of how you can utilize it in your current and future System i application development. After reading this, you may find yourself compelled to learn more about the .NET application development environment and how you can use the Microsoft toolset to quickly develop high-impact desktop and Web-based applications for the System i platform.

Setting the Stage

As I've attended various System i events over the last few years, I've noticed a disturbing trend. It seemed that .NET technology had been ignored or poorly covered at most of the seminars and educational sessions. RPG, Java* and PHP languages are great development tools, but if you don't see the entire picture of what's available for System i development environments, you can't make an informed decision about the best development methodology to use for developing new System i applications for the desktop or Web. Because of the lack of .NET education tracks at COMMON, I asked the user group to add a series of introductory sessions and a one-day seminar to give System i developers an introduction to .NET. They obliged me and we did the first set of sessions at the Spring COMMON 2007 event in Anaheim. There's a one-day lab class planned for this month's COMMON Focus in Columbus, and next spring we'll repeat the sessions in Nashville. You'll also be seeing some .NET coverage in this magazine.

The Catalyst for Change

I want to provide a little background that should illustrate why System i developers might want to consider updating their skill sets and learn a new technology such as .NET. Over the last 15 years, we've seen the Internet revolution come to pass. We've seen personal computers go from being almost non-existent to a daily staple. The newer generation of young CFOs and CIOs are more computer literate and expect a GUI to front-end every application they use. Because of this change, many companies have begun to replace the System i applications that have run their businesses successfully for years with Windows* technology-based ERP and other applications. I've also personally seen companies struggle with the decision to continue using their System i platform or completely migrate to a new one. We in the System i world need to provide these companies with reasons to keep their legacy systems and the tools to modernize them for future needs.

With a little education on how to use a technology like .NET to develop applications, a System i programmer can quickly write a graphical application on top of an existing System i solution, thus adding a new user interface to important parts of an existing system or simply adding ease of access to existing ERP data from a Windows or Web-based inquiry. My experience with this dates back to the early 1990s when I was working as a developer for a company whose executive team demanded a daily report that could be quickly accessed from a desktop icon. They wanted to see a chart showing how things were going, not a greenbar report. We used a query tool to build the data into a pie chart and showed how System i information could be quickly published in a graphical format. This is how I formed the phrase: “A little GUI can go a long way” when trying to build PR value for the System i platform.

Why .NET?

With the variety of development methodologies available today, such as Java, PHP, RPG CGI and .NET, why would anyone choose the .NET development route for developing new System i applications? As a long-time System i developer, I’ve had the opportunity to work with every one of these development environments and more. After recently going through a two-year learning curve to become proficient in Java, I concluded that the average System i developer might not have the time to effectively learn and implement this new development technology, especially if they wanted to quickly build Windows and Web applications to enhance the value of their System i architecture.

At the same time I was learning Java, I was evaluating .NET technology as a next-step migration path for several of our existing Windows-based products. What I learned after working with the .NET technology is that I could quickly write a subfile-like Windows inquiry application in fewer than 100 lines of actual code. I also learned that I could access my existing RPG programs on the System i platform, take advantage of the database and achieve sub-second response time from my Windows and Web applications—just like I was getting from my System i applications. Really, the choice for me boiled down to the fact that I found the .NET environment and, in particular, the VB.NET language was natural and easy to learn, just like RPG. A lot of the concepts such as sub-procedures and D-specs have direct correlations between RPG and the VB.NET language.

What Is .NET?

As the year 2000 approached, we developers spent a lot of time worrying about whether our applications would work after the clock struck midnight. The moment passed and I remember thinking, “OK, now that the new millennium has proven that there really wasn’t much to worry about, it’s time to start thinking about what exciting things will happen during the 21st century.” The year 2000 was also the year that Java really hit the scene and was promoted by IBM. Meanwhile, the Microsoft development team was working on a new technology that would soon become known as the .NET framework.

The way I like to explain it is that Microsoft sat back and observed what Sun was doing with the Java language and environment. Then they chose the best pieces of the Java environment and improved on those to bring us the .NET framework. At the same time, they wanted to appease the Visual Basic and C/C++ developers on the PC side of the fence with a single, unified programming environment that supported both language sets. The Internet also needed to be an integral part of this new development environment because of its growing importance.

The .NET programming environment was released in 2002 and received with great fanfare and fear from the PC programming world. VB6 had become entrenched as the PC programming language of choice and VB.Net had some changes to the language that made VB programmers a little intimidated, much like the initial fear a System i developer can feel when choosing to learn a new language. Recent improvements to the VB.Net language, PC-to-System i connectivity speeds and System i database drivers as well as .NET 2.0 and the Visual Studio 2005 programming toolsets have made writing desktop applications, Web applications and Web services with .NET a lot easier. Now, a free version of the Visual Studio 2005 programming environment is available to programmers who want to try their hand at developing desktop and Web applications (<http://msdn.microsoft.com/vstudio/express>). With all the recent improvements and FREE tools available, there is virtually no barrier in place, other than a slow Internet connection, that would prevent developers from using .NET technology to start developing applications for the System i platform.

System i Connectivity with .NET

One of the things I like best about programming applications in the .NET environment is the ability to write an interactive subfile-style of inquiry with little programming. Direct SQL calls to the database let me easily filter records and display them in a data grid that is essentially the equivalent of a subfile. This type of program can be easily expanded to create table maintenance, order entry and other programs.

Remote-command calls provide a basic functionality that allows .Net programmers the capability to run any CL command or program that doesn't require the user to receive any parameter feedback. A remote command call is pretty much limited to a success or failure notification that the job was run. A CPF message (System i error message) can be returned to the user application upon program completion or error allowing for simple error notification. Although limited in functionality, remote-command calls can be a great way to build a graphical front end to a report call that you then want to submit as a batch job with the submit job (SBMJOB) command. You may also have a CL command that sends back a simple message to provide user feedback or you may simply want to trigger a System i process from your desktop or web application.

Remote-program calls are even more powerful. A remote-program call is an important extension to the remote-command call that lets you call any existing System i RPG, COBOL or CL program, pass parameters to the program and receive back the parameter results after the call. On the System i platform, we've always used the term "call/parm" to describe this functionality; however the same functionality can be used from within a .NET application to call an System i program and pass parms. This lets developers reuse existing System i program logic in a Windows or Web application without rewriting existing System i RPG or COBOL code. An example of this in the real world might be passing parameters from a Web page for a customer who wants to purchase an item. A call to an RPG pricing program could be made to return custom pricing based on an existing pricing routine instead of rewriting the code in VB.NET.

Stored procedures are another topic that could warrant several articles. Think of a stored procedure as a nice way to package up an SQL query or an RPG program call so that parameters or arrays of record data can be easily passed to the program and returned to a Windows or Web application. Stored procedures are a useful way to write a chunk of logic on the System i platform and use it from within RPG and COBOL programs or a .NET windows or Web application.

What's Next?

In an upcoming article, I'll delve deeper into various System i connectivity issues and further illustrate what you can accomplish with .NET and the System i platform. I hope that I've stimulated your curiosity enough to want to try the .NET toolsets and that you'll start thinking of ways you can add value to your System i platform by building Windows and Web applications that take advantage of its best features.

Richard Schoen is the president and chief technology officer of RJS Software Systems, an information-management and data-integration solutions developer for the System i platform. Richard founded RJS in 1990 and today guides the direction and development of all of the company's core products. Richard can be reached at: richard@rjssoftware.com