Better Tools: iSeries Projects

By George Farr and Phil Coulthard, IBM Toronto Lab

In this our third article, we continue our description of the first step in the J2EE five-step roadmap. This first step is "Better Tools", and in it we use modern tools for RPG and COBOL development. In the last issue, we described the Remote System Explorer (RSE) for PDM-style drill-down access to remote objects and members. While we believe the RSE is where most will and should start their modernization journey, we think the next logical step for many will be into a more structured development environment and process offered by iSeries projects.

An iSeries project is a typical Eclipse project. Ultimately, it is nothing more than a folder on your local Windows disk, containing other sub-folders and files. However, in this case these sub-folders represent iSeries source files, and the files are iSeries source members. Each iSeries Project is associated with a single library within an RSE connection, and this association is important in a number of actions you can perform, as we will see.

There are two ways to create iSeries projects: using the New iSeries Project wizard available from the File->New->Other window, or by using the "Make Available Offline" popup action in the Remote System Explorer. The former

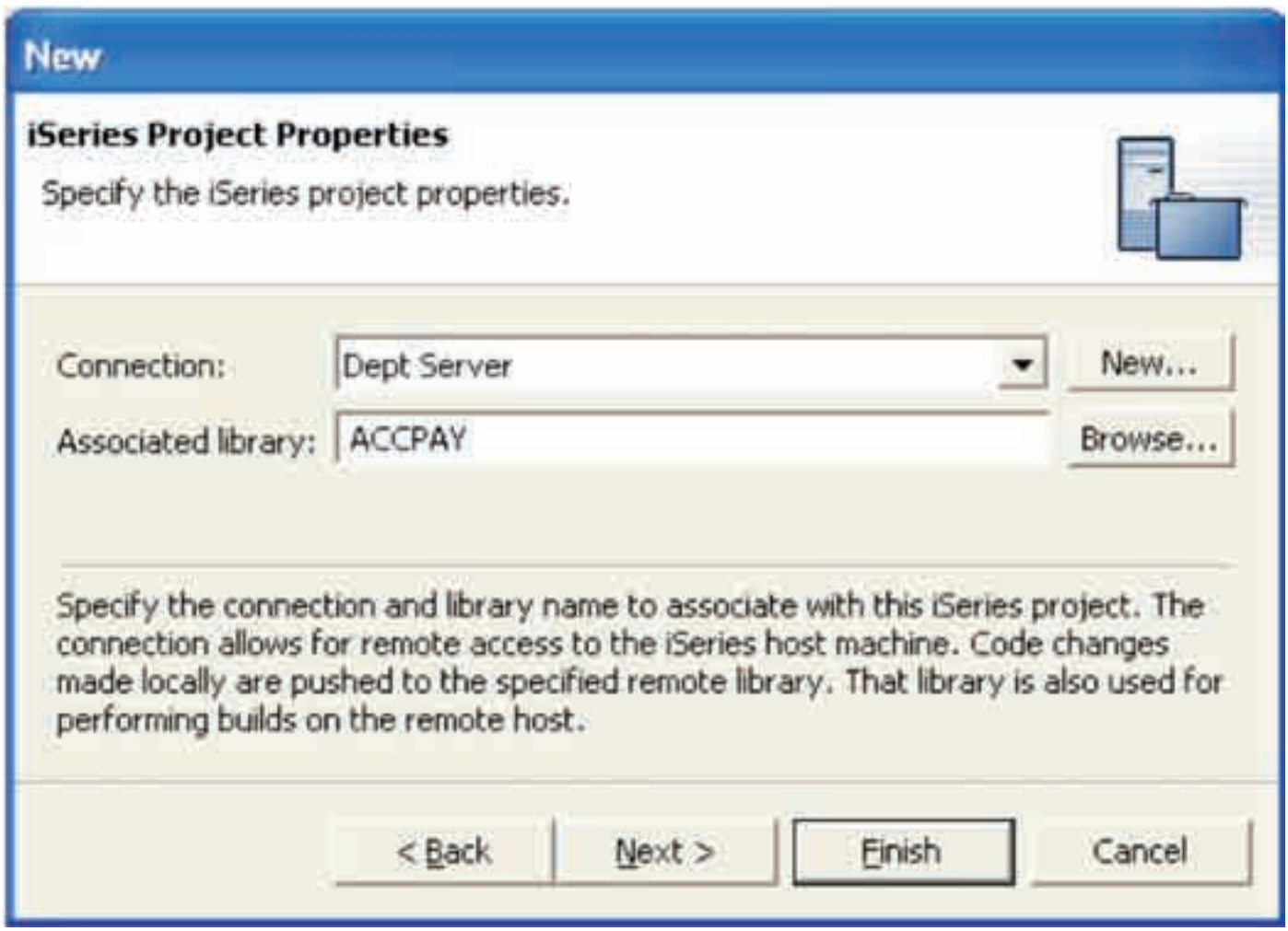

Figure 1. New iSeries Project wizard

prompts for the project name, and the RSE connection and associated library as shown in **Figure 1**. The latter implicitly determines the connection and library of the selected files or members, and creates a project with a derived name and the associated connection and library.

Once you have a populated project, you are now in a position to work with the source in it. For example, you can edit a member with the same rich editor that is used in the Remote System Explorer. The difference though is that you are editing a local copy of the member, not the remote member directly. This means you can work on the member in disconnected mode, while sitting in a train for example. This is most powerful when combined with the caching support in the RPG, COBOL and DDS program verifiers, which allows you to find and fix all compiler errors while disconnected. There are other popup actions you can perform on the local members, such as rename, delete, copy and paste. The default view in the iSeries Project Navigator (the tree view on the left in **Figure 2**) is to show only members you have locally in your project. However, sometimes it is nice to see all the members, and objects, in your associated library as well, for context. To this end, in the right-click action for the project you can enter into Show Remote Objects mode, which shows the combination of all remote members and objects and all local members. If a member is both local and remote then it is not shown twice, but rather it is only shown once and the double-arrowed yellow icon and text indicate that is both local and remote.

If the member has been changed in the associated library since importing it into the project, the icon adornment will be red instead of yellow, indicating a collision. Similarly, different icons and text indicate members that are only remote or only local.

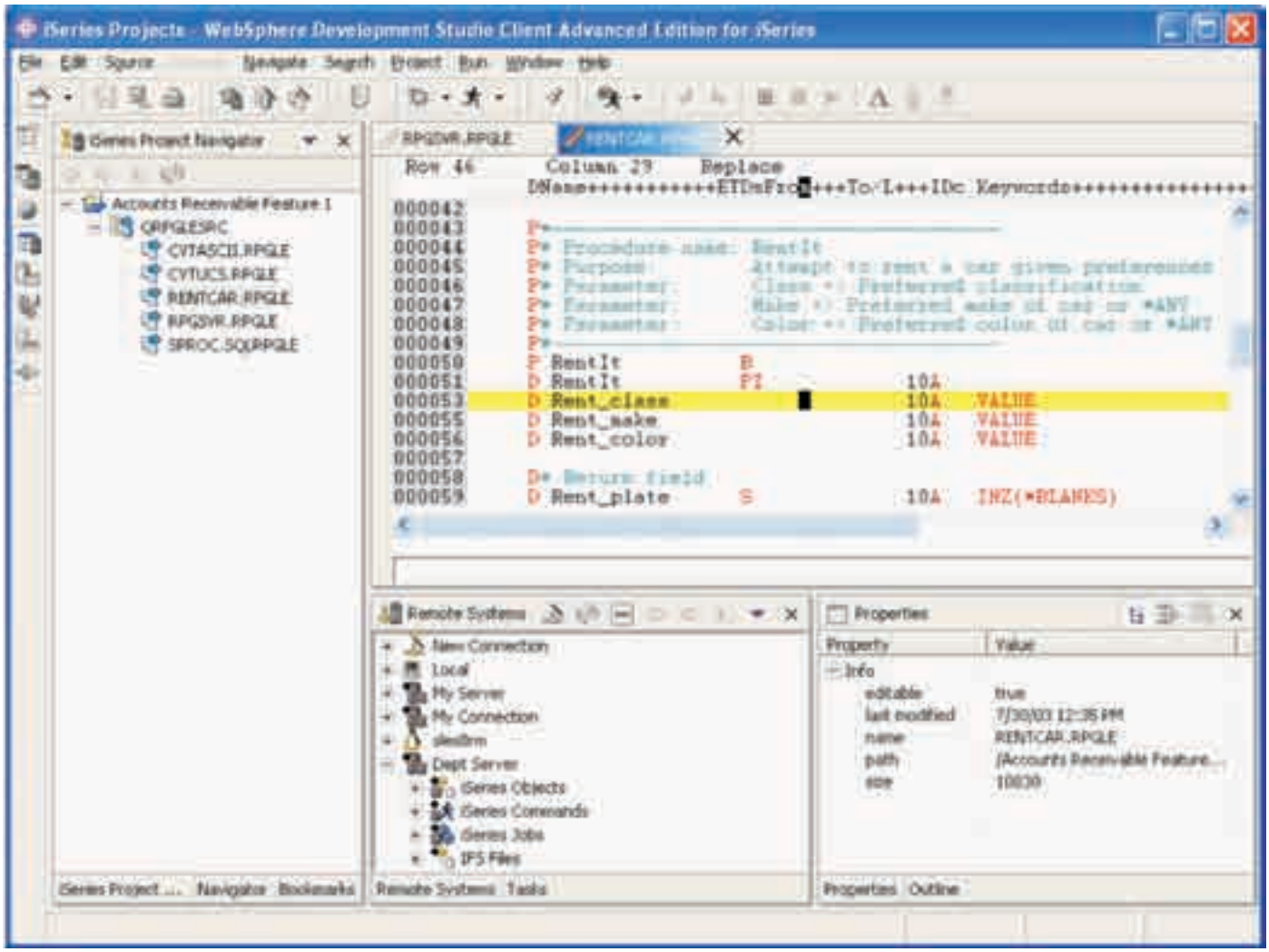

Figure 2. iSeries Projects perspective

Once you have an iSeries project, you work in the iSeries Projects perspective, where you see and manipulate your projects. In this perspective you can use wizards to create new source files and source members in the project, or you can use the Import Remote Objects popup action on the project to populate it with existing source members from any library on any system. **Figure 2** shows the iSeries Projects perspective in its default configuration.

In **Figure 3** we see the iSeries Project Navigator in show-remote-objects mode, and the popup menu for a remote-only member. Notice how we can easily import remote members in this mode by selecting the Add to Project action.

For any remote object, if you want to perform actions directly on the object, the Show in Remote Systems View action will expand the embedded Remote Systems view to show and select the object, so the RSE's full complement of actions are easily accessible there, for the selected object.

Finally, when you are done editing and working with your local source, you use the Push action to push the changed members back to the project's associated library. If you created any new source files or members locally, on a push the appropriate CL command is run to create them. If someone else has changed a same member in the meantime, you will be told of this and given the option to continue or cancel the push. However, in most cases we recommend using a unique scratch library per developer, for the purpose of compiling and testing their own changes prior to integrating with their colleague's changes.

Speaking of compiling, this is usually the next interesting thing you will want to do after pushing your changes. This is most easily done by using the Build action, which by default will generate a CL member named COMPILE.CLLE for compiling each changed member, using the last-used compile command from the Remote System Explorer. Of course, you can edit that member as needed, and by configuring your build style from the project's properties you can stop the default behavior of re-generating that CL member each time the Remote Actions->Build action is selected.

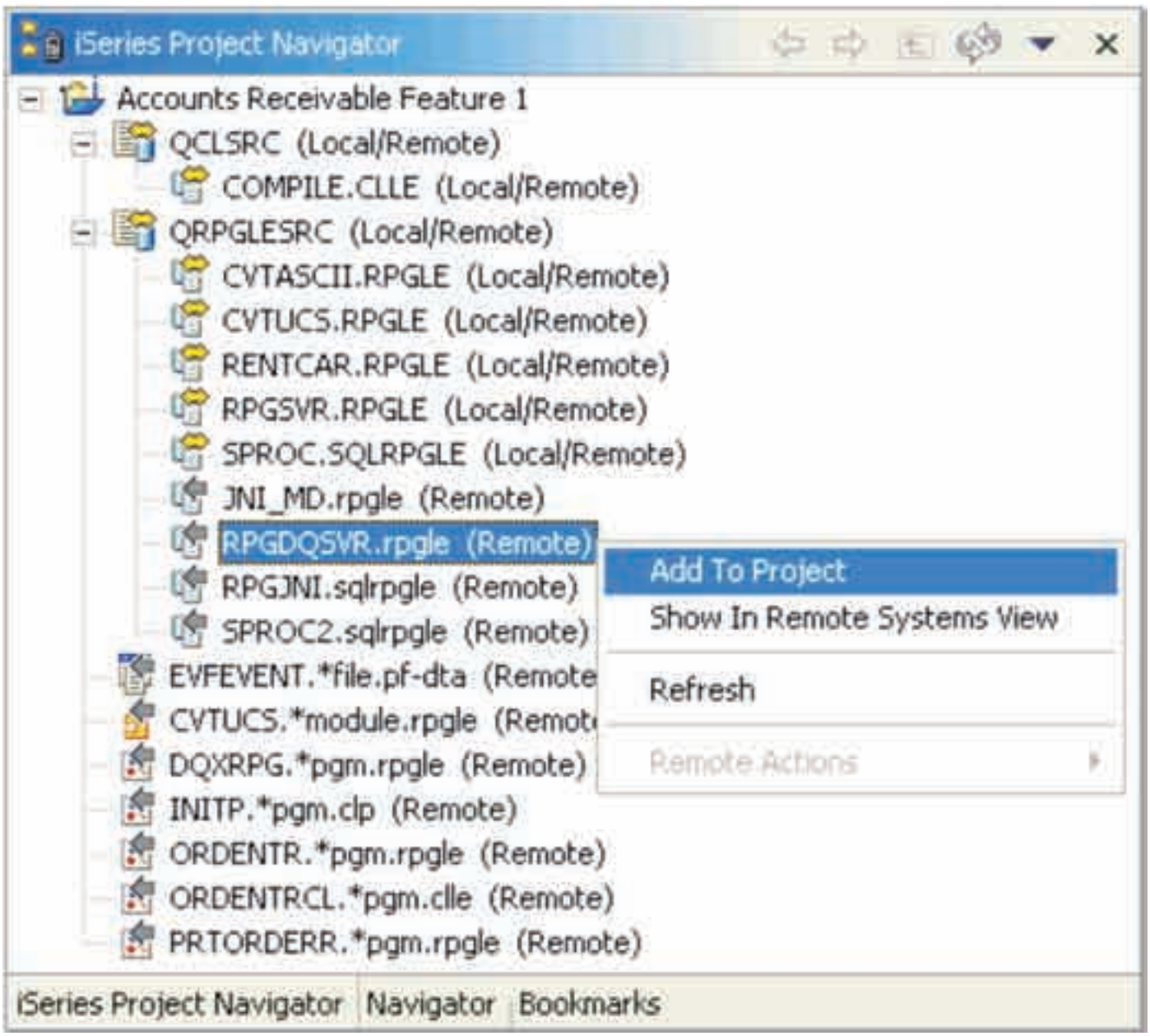

Figure 3. iSeries Project Navigator in Show Remote Objects mode

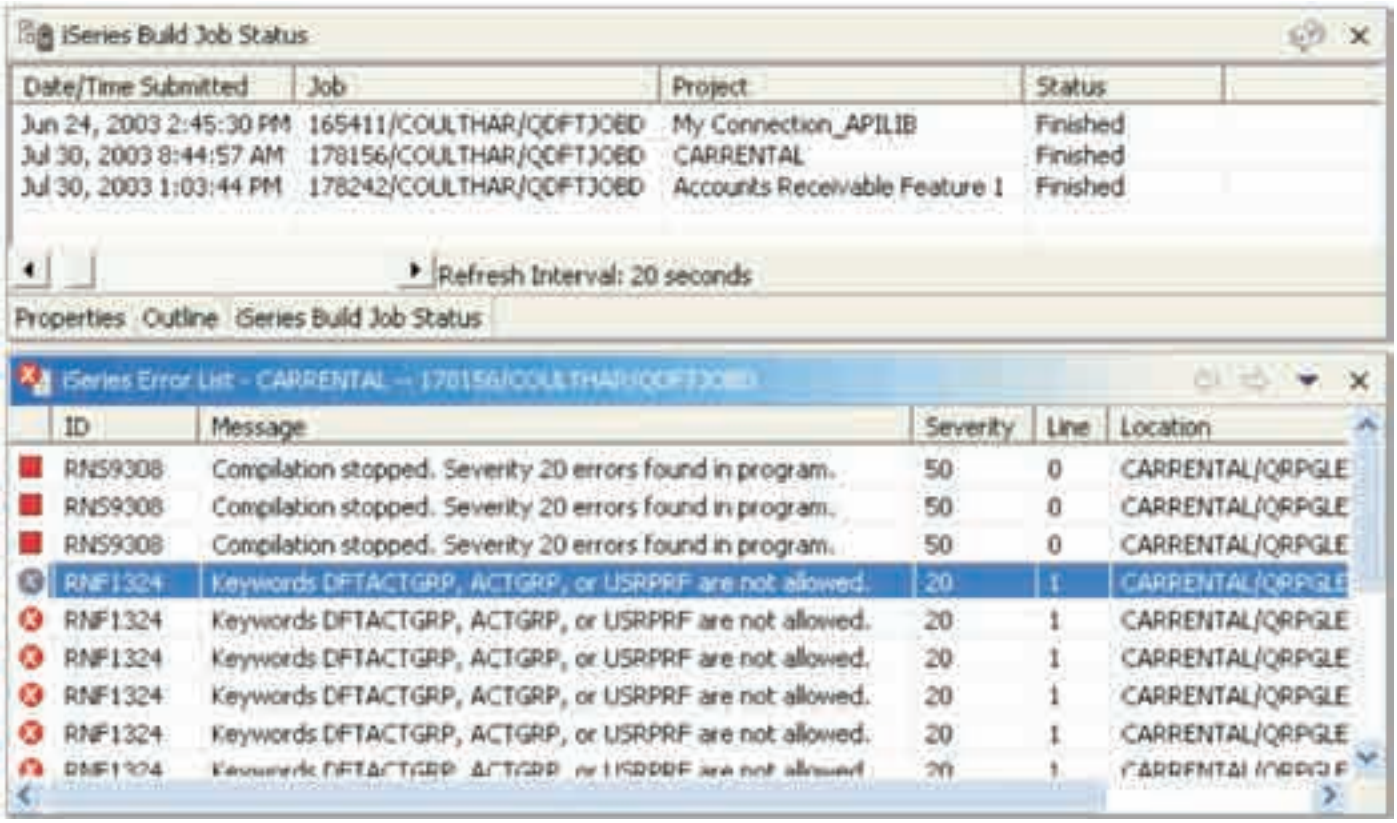

Figure 4. Build Job Status and Error List views

When this action eventually is selected, the generated or edited CL member is pushed to the associated library of the project and compiled, and run, there. Further, if you manually create a BIND.CLLE member, it too will be pushed, compiled and run on a build, but only if the COMPILE member compiles and runs successfully. This is where you would put post-compile actions such as binding your ILE modules into programs and service programs. All submitted builds are displayed in an "iSeries Build Job Status", where you can easily monitor their status in the job queue. When complete, all errors from all compiles are combined and can be seen in a single Error List view by a right click Retrieve Errors action in the build status view. As in the RSE, when you double click on one of these errors, the member is opened in the editor, and you are positioned at the error. Of course, if you used the verifiers you will not have compile errors in your RPG, COBOL or DDS members ever again. **Figure 4** shows the build status view, and error list view.

By the way, we are often asked how to permanently affect the library list of a project so that the builds pick up the necessary libraries. The answer to this is the same as it is in the Remote System Explorer: via the Properties action on the associated connection, in the Remote Systems view, in the Subsystems->iSeries Commands page.

There are other options for doing builds as well, and these options are known as "build styles". For example, there is a build style to simply run an iSeries command that you specify in the configuration for the build style, which is the right choice if you already have a process for building your application. If you use any of the popular change management products on iSeries, then chances are that they supply their own build style as well, which snap into the iSeries Project perspective … the build action is specifically designed to be extendible by vendors and indeed it has been.

When your changes all compile cleanly you can use the Debug action to run the application and step through the code to ensure it works as expected. This is the same debugger used in the RSE, and by Eclipse for Java. To launch the debugger, use the debug icon in the toolbar, and in its dropdown select Debug As to specify the CL command to launch the application. You can just from four flavors of iSeries debugging: batch, interactive, job (existing) or multi-threaded. Once you create the debug configuration once, you can subsequently just select it from the dropdown list. Also as in the RSE, you can set breakpoints within the editor prior to launching the debugger … by simply double clicking in the left-most margin area for the line you wish to stop at.

So these are the basic steps you iterate through when working with iSeries projects … create the project; populate it via importing or creating members; edit and verify as you would with the RSE; then push your changes; build the changed members; and finally debug your changes.

There are two primary reasons for using iSeries projects. The first is for disconnected mode, as we have mentioned, with all the power of the editor, program verifiers and reference manuals at your disposal. The "Make Available Offline" actions in the Remote System Explorer make this scenario very straightforward. The other option is a more structured development method. Because these projects are typical eclipse projects, they support the team actions that all eclipse projects support. This means you could associate the project with a CVS or Rational ClearCase repository on the LAN, for example, and team members working on the same feature or enhancement could all share the same project. Each would first import, edit, push, build and debug their own changes to the members. When happy with their own changes, they then would synchronize with the repository, picking up their colleagues' changes. Then they could push, build and debug the integration of their changes with their colleagues'. They would iterate this process until complete, and then use their existing process for pushing their collective changes up the stack to test and eventually production, hopefully using one of the iSeries change management products.

The use of iSeries Projects exposes programmers to eclipse project-based terminology and functionality, and so it is a good stepping stone to further steps in the roadmap. If you know how to work with iSeries Projects, you will have less culture shock when you move into working with WebFacing, Web and Java projects, for example.

We hope this introduction to iSeries Projects have convinced you they are an interesting new tool for RPG and COBOL developers, and they play an important role in the modernization of your team as it moves through the roadmap, or simply as it works on that next critical release. We'll talk to you again next month!

***George Farr** and **Phil Coulthard** are co-authors of the Midrange Computing books Java for RPG Programmers and Java for S/390 and AS/400 COBOL Programmers. George can be reached at farr@ca.ibm.com, and Phil can be reached at coulthar@ca.ibm.com.*


Phil Coulthard


George Farr

* This article is a reproduction of an article first published in iSeries News/400 Magazine. (Reprinted with permission.)