


[print](#) | [close](#)

## Tips for Programming and Development

*System iNEWS Magazine*

[Aaron Bartell](#) [Bryan Meyers](#) [Dan Riehl](#) [Jef Sutherland](#) [Lynne Noll](#) [Charles Caplan](#) [Julian Monypenny](#) [Scott Klement](#)  
[Wally Day](#) [Dan Riehl](#) [CARMEN NULAND](#) [Lynne Noll](#) [Scott Klement](#) [Alan Seiden](#) [Carsten Flensburg](#) [Alan Seiden](#)  
[Bryan Meyers](#) [Jef Sutherland](#) [Carmen Nuland](#) [Alan Seiden](#) [Carsten Flensburg](#)

Aaron Bartell

Fri, 12/01/2006 (All day)

### Debugging

#### Store Program Debug Information Automatically

[Click here](#) to download the code bundle.  
 To report code errors, email  
[SystemiNetwork.com](mailto:info@SystemiNetwork.com)

Including debug information in program objects is mandatory for many shops because it enables you to immediately start a source debug session if necessary. It also provides an option to recover lost sources or

verify the current version of a program.

Before V4R5, the preferred method to ensure a specific parameter setting on compile commands was either to use the Change Command Default (CHGCMDDFT) command to change the parameter setting (if the parameter in question had a default already), or to create a PDM option of the compile command, which included the required keyword and parameter value.

The Command Analyzer Change exit point — introduced in V4R5 — changed the options to override command parameters substantially. The exit point is documented at [publib.boulder.ibm.com/infocenter/iseres/v5r4/index.jsp?topic=/apis/xcachg.htm](http://publib.boulder.ibm.com/infocenter/iseres/v5r4/index.jsp?topic=/apis/xcachg.htm). Using the Command Analyzer Change exit point, you can ensure that the following compile commands include the option to create debug information with the program object:

- Create RPG Program (CRTRPGPGM)
- Create CL Program (CRTCLPGM)
- Create RPG Module (CRTRPGMOD)
- Create Bound RPG Program (CRTBNDRPG)
- Create CL Module (CRTCLMOD)

You create the exit program with the following commands:

```
CrtRpgMod  Module( CXP001 )
           DbgView( *LIST )
CrtPgm     Pgm( CXP001 )
           Module( CXP001 )
           ActGrp( *CALLER )
```

To call the CXP001 Command Analyzer Change exit program whenever a compile command is invoked, you need to register it — the commands in [Figure 1](#) take care of that. In the example, the exit program is located in library QGPL, but you can change that to meet your specific requirements.

The Work with Registration Info (WRKREGINF) command verifies and changes or removes the registration:

```
WrkRegInf    ExitPnt( QIBM_QCA_CHG_COMMAND ), select option 8.
```

If for some reason you distribute your program objects and want to protect the source from being displayed or retrieved, remember to remove the observable program information from your programs before shipping them, as follows:

```
CHGPGM  PGM([ / ])
        RMVOBS(*ALL)
```

If required, you can adapt the CXPOO1 program to use it as a starting point for your own Command Analyzer Change exit programs. (To download the source code, go to [SystemiNetwork.com/code](http://SystemiNetwork.com/code).)

— Carsten Flensburg  
System i programming team leader  
Novasol,  
Copenhagen, Denmark

## 7 Debugging Tips

For those of us still using the good old green-screen debug and service job commands, I have collected seven tips that can make life a little easier when you want to run a service job or debug session.

1. When running the STRSRVJOB command, you can specify just the job name and skip the user name and job number, as in the following example: `STRSRVJOB JOB(DAILYUPD)` If only one job with the specified name exists, that job will be serviced. Otherwise, a list of all jobs by that name will display, each with its job status and job date; simply select the one you want to service.
2. If a batch job enters a message wait (MSGW) status because of an exception message, and you created the currently executing program with source debug information, follow these steps to display the failing statement in a source debug session:
  - a. Run the STRSRVJOB command against the job currently in the message wait status.
  - b. Run the STRDBG command against the qualified program name of the failing program (the library should be specified to avoid library list issues). Remember to specify the OPMSRC(\*YES) keyword on the STRDBG command if the program in question is an Original Program Model (OPM) type of program.
  - c. The Display Module Source screen displays and all debug commands are available to you (e.g., to inspect the current values of various program variables). This can be a great help in identifying the actual cause of the program failure.
3. If you have multiple active terminal sessions open and you forget which one is currently in debug mode, run the Display Debug (DSPDBG) command to identify the session in debug mode. The resulting panel ([Figure 2](#)) also displays the current debug attributes.

4. If you need to change one or more debug attributes after the debug session has been activated — for example, to allow update of production files — run the Change Debug (CHGDBG) command. [Figure 3](#) shows the CHGDBG command prompt displaying all attributes eligible to change in an active debug session.
5. If you are currently servicing another job on the system — for example, to run a debug session against a program in that job — you can use the Display Service Status (DSPSRVSTS) command to verify from which terminal session you ran the STRSRVJOB command. You can also view the current service status, including job information about the job that is currently being serviced and whether the debug status is currently active.
6. You can run the DSPSRVSTS command from a job to verify or reveal that it is currently being serviced by another job. The resulting Display Service Status panel includes information identifying the job that issued the STRSRVJOB command against your job.
7. The steps involved in setting up a debug session for a batch job are as follows:

- a. Submit the batch job to a job queue using the Submit Job (SBMJOB) command. It is important that you specify HOLD(\*YES) on the SBJOB command:

```
SBMJOB CMD(CALL PGM(QGPL/CBX001))
JOB(TEST) HOLD(*YES)
```

- b. Start servicing the submitted batch job. From your interactive session, run the STRSRVJOB command against the job you just submitted: STRSRVJOB JOB(TEST)
- c. Run the STRDBG command against the program you want to debug: STRDBG PGM(QGPL/CBX001) When the Display Module Source panel displays, press F12 to continue.
- d. Release the submitted job from your interactive session. This displays the Start Serviced Job panel, where you press F10 to go to the Command Entry display.
- e. Issue the DSPMODSRC command and enter the appropriate breakpoints in your program using the F6 key and cursor position. When finished, press F12.
- f. Exit the Command Entry display by pressing F3. This takes you back to the Start Serviced Job panel, where you press Enter to start the debug session.

— Carsten Flensburg  
System i programming team leader  
Novasol  
Copenhagen, Denmark

### Debug an Active Job's Error Message

With the error message still unanswered, first determine the qualified job name (job name, user, and job number) of the problem job and the program with the failure. You can use various commands such as WRKACTJOB or WRKUSRJOB to find this information (to find the program in error, review the error message in the job log or examine the call stack and look for the last user-written program). Next, from your command line, start a service job with command STRSRVJOB, specifying the

qualified job name of the problem job as the job to service. Doing so lets you enter debug commands for that job. After starting the service job, enter the STRDBG command, specifying the problem program you identified, and press Enter.

The debugger displays the program in error and is positioned to the line of code that precipitated the error. You can then explore the situation in the debugger and solve the user's problem more quickly. You'll also know right where to go to fix the bug. When finished debugging, end your debugging session with ENDDBG and end the service job with ENDSRVJOB. Remember, this tip works for both interactive and batch jobs.

— *Carmen Nuland*  
*Senior systems analyst*  
*Hal Leonard Corp.*  
*Winona, Minnesota*

## **Documentation**

### **Create a Screenshot Macro in Word**

We hated creating our documentation with actual System i screenshots because, when printed, they always use too much toner (black background). We started creating separate sessions with light-colored backgrounds just for copying, but that was too much of a hassle. Instead, we found a Word macro that copies the important parts and formats them to look like a System i screenshot.

First, you must create the macro(s) in Word:

1. Copy the code for the macro into the clipboard by selecting the text and then clicking Edit|Copy (or press Ctrl+C). You can download the macro at *SystemiNetwork.com/code*.
2. Open a blank Word document.
3. Select Tools|Macro|Macros (or press Alt+F8).
4. Type in a macro name and press the Create button.
5. When the Microsoft Visual Basic screen appears, paste the code into the window, deleting the duplicate "Sub" and "End Sub" lines, if necessary.
6. Close the Microsoft Visual Basic window to return to the regular Word editing window.
7. Right-click in a blank area on the toolbar and select Customize.
8. Select Macros from the Categories list. The macros are listed on the right side. Select the Macro command and drag to a position on the toolbar.

9. While the Customize window is still open, right-click the new toolbar macro and select Default Style to change it to an icon.
10. Right-click the new icon and select Change Button Image. Select an icon for this macro. There are 36 choices for the icon, and you can edit the icons if you wish.
11. Right-click again and change the Name to the name of the macro.
12. Close the Customize window.
13. When you close the current document, the new macro/icon combination is saved to your Normal.dot template and becomes available to all documents.

To use the macro, either use the Tools|Macro|Macros option again or create a toolbar button. To create a button:

1. Right-click over an empty area of the toolbar and select Customize.
2. In the Commands tab, select the Macros category. You should see your new macro on the right side.
3. Click and drag the macro command to the toolbar.
4. While still in Customize mode, you can right-click over the macro and change the name, change it to icon only (default style), change the icon used for the button, and select other options. When all settings are the way you want them, close the Customize window.

To use the macro on an iSeries Access screen, click the copy button (or press Ctrl+C) to copy text to the clipboard. If you select an area first, it will copy only that area; if you don't select anything, it will copy everything that is shown in the window. Go to a Word document and click the new macro button you created. Doing so copies the text and format into a windowlike layout ([Figure 4](#)). Please note, underlines will not get copied.

*— Carmen Nuland  
Senior systems analyst  
Hal Leonard Corp.  
Winona, Minnesota*

## IFS

### Mass Searches of Files for a Certain Value

As the Internet age progresses, the need for text-based files (e.g., .xml, .html, .txt, .csv, .logs) in the IFS becomes more prominent. However, the information contained within these files is not sorted for easy access. A way to easily search for information is to use the System i's QShell environment and the grep command. Say you recorded responses from web service requests, and in those responses were the HTTP headers, including each response's status code (e.g., 200=Ok, 500=Server Error). Now let's say that you have thousands of files in a directory and want to know which ones returned a server response of 500=Server Error. You can do this with the following command:

```
grep '500' /www/rxs/output/*.xml  
> /home/aaron/server500errors.txt
```

This command searches for the phrase "500" in all files ending with .xml in the /www/rxs/output directory and will send all round results to /home/aaron/server500errors.txt for your review. You can omit the > /home/aaron/server500errors.txt and display the results on the current screen. However, doing so can make for a busy screen because of line wrapping and such. To get into QShell, execute command STRQSH from the command line.

*— Aaron Bartell  
Programmer  
Krengel Technology Inc.  
Mankato, Minnesota*

### **Use QShell to Easily Create Utilities**

QShell provides a Unix-like command shell for i5/OS. This is a great environment for working with the IFS. Because the IFS is designed to work like a Unix file system, it fits nicely with the QShell paradigm.

To get started with QShell, you must have 5722-SS1 option 30 installed. This option is included as part of i5/OS; there's no charge aside from what you've already paid for the operating system. Once you've installed option 30, you can run QShell commands on your System i.

If you want to run interactive QShell commands, you can type STRQSH to get a Unix-like command line. However, I've found that the most interesting use of QShell is from within a CL program. I'm often asked how a CL program can get a list of files in an IFS directory or how to easily purge files in the IFS. QShell does both of these tasks.

[Figure 5](#) shows a CL program that gets a list of files in the /home/scottk/testdir folder of the IFS. It uses the QShell "cd" command to change to the appropriate directory and the "ls" command to get a list of files. QShell would ordinarily write these files to the screen, but an OVRDBF command overrides the output to a physical file (PF) in the QTEMP library. You can then read the PF using ordinary means, such as CL's RCVF command, to retrieve the list of files in your own program.

[Figure 6](#) demonstrates purging old files from the IFS. It uses the QShell "find" command to locate files in the /tmp folder (or any of its subfolders) that haven't been accessed in 30 days. Again, the output is written to a PF in the QTEMP library. The CL program then loops through the contents of this PF and deletes each file.

This only scratches the surface of what you can do with QShell. For more information, read about QShell at the Information Center under Programming|Shells and Utilities|QShell.

— *Scott Klement*  
**System iNEWS** technical editor

## Message Handling

### Message SubFile Window

I use diagnostic and escape messages in a number of utility programs. In most cases, all I need to know when I encounter an error is the text of this message; a message subfile is all that is necessary to show this. However, not all of the screens I work with (some of which date to the System 34) have message subfiles. In some cases, they do have message subfiles, but I want to make the exception more prominent.

I find that it is useful to have a program that checks for messages in a program queue, and if there are any, displays them in a message subfile and clears the queue. You can put this code either in an interactive program just before an EXFMT or right after a call (with an E extender or a blind monitor group). A window pops up to show any messages. If there are no messages, nothing shows. I don't have to reformat an old screen to show messages, and I don't have to check whether an exception occurred, as long as showing a message is all I want to do to handle the error.

All you need for parameters is the name of the program queue to show, a flag to say whether to shut down, and an optional title. The window includes an alarm to catch the user's attention.

— *Lynne Noll*  
*Senior programmer analyst*  
*Battenfeld Gloucester, SMS Group*  
*Gloucester, Massachusetts*

## PHP

### Configure PHP Dynamically with the ini\_set() Function

PHP's runtime configurations are stored in a file called PHP.ini. Dozens of settings, including error-handling directives, database options, and syntax conventions, are stored here. Although PHP.ini's location can vary, the default location in Zend Core for i5/OS is /usr/local/Zend/core/etc/php.ini.

For semipermanent, installation-wide settings, PHP.ini is effective. Many situations, however, call for a more dynamic approach. The ini\_set() function (visit the manual page at [php.net/ini-set](http://php.net/ini-set)) fills this need.

Applications can use ini\_set() to spontaneously alter most runtime settings. The changes take effect only in the calling script and expire when the script ends. Basing its parameters on PHP.ini's key names and values, ini\_set() is easy to use and useful in many situations, such as when

- a single PHP installation hosts multiple applications with different needs
- the developer lacks editing privileges or connectivity with PHP.ini
- the application must change settings on the fly in response to data
- developers want to experiment with settings without rebooting the web server
- you want to reduce the application's reliance on the development server's settings; the application can generate its own settings based on the environment in which it finds itself

[Figure 7](#) shows examples of using ini\_set().

— Alan Seiden  
Senior developer/technical lead  
Strategic Business Systems, Inc.  
Ramsey, New Jersey

## Process Check Boxes as an Array in PHP

PHP provides dozens of functions for processing arrays, and thus thrives on array structures. PHP's power to process web forms is attributable in part to its ability to read form data as arrays.

Check boxes allow users to select multiple items from a list on a web form. When a form containing check boxes is submitted, the browser sends only the values that have been checked. In PHP, an array called `$_POST` contains all values submitted by the form, including check boxes and other data. PHP can process the check boxes separately if you use a special naming convention.

If the check boxes in a list are named identically with a pair of brackets appended to the name (e.g., "item[]"), PHP can view the submitted check boxes as a distinct array.

[Figure 8](#) shows a list of check boxes that follow this convention in the script `choose.php`. The form is submitted to a second script (`process.php`) that reads the check boxes. At that point, the values will be in an array that can be easily and conveniently processed.

— Alan Seiden  
Senior developer/technical lead  
Strategic Business Systems, Inc.  
Ramsey, New Jersey

## Define a Constant Array in PHP

Constants encourage reliable, self-documenting code. Once defined, a constant's value cannot be changed by code that executes later. The constant's scope is global, accessible (read-only) from anywhere in the script. Despite their usefulness, constants in PHP normally are limited to scalar (single value) data types — boolean, integer, float, and string. However, with a little effort, programmers can use constants to store arrays.

The trick is to store the array as a string. Its value can then be assigned to a constant. Later, when the value is needed, you can convert the string back to an array. PHP's `serialize()` and `unserialize()` functions serve this purpose perfectly. `Serialize()` converts a compound object, such as an array, into a string; `unserialize()` restores the string back to its original type and structure.

[Figure 9](#) shows a constant array that is serialized at the beginning of the script, to be unserialized later when needed.

— Alan Seiden  
Senior developer/technical lead  
Strategic Business Systems, Inc.  
Ramsey, New Jersey

## RPG/CL

### Use CPYF to Ignore the Key



Occasionally, you might want to copy a keyed file sequentially, ignoring the key. The CPYF command defaults to using the keyed sequence for the target file. To get around this, change the following parameter:

```
CPYF      FROMFILE (X)   TOFILE (Y)   FROMRCD (1)
```

Changing FROMRCD from the default of \*START to a value of 1 tells the CPYF command to ignore the keyed sequence.

— Wally Day  
Senior programmer/analyst  
Miicor Consulting, Inc.  
Boise, Idaho

## The Q Command

I use Q more than any other System i command, so I created a version on nearly all of our client machines. It is useful not only for quick data views, but also for retrieving general information about a file. Creating the command is a snap and takes just a couple of minutes.

```
CRTDUPOBJ OBJ(RUNQRY) FROMLIB(*LIBL)
          OBJTYPE(*CMD) TOLIB(QGPL) NEWOBJ(Q)
CHGCMDDFT CMD(Q) NEWDFT('rcdslt(*yes)')
```

You can name the new command anything you like, and the target library can be something other than QGPL, but it should be a readily accessible library. Using the command is simple: On a command line, type q \*n file-name.

The first panel that appears is the selection criteria prompt. You can use this view to determine field names, data types and lengths, and descriptive information. Pressing F18 provides record format information. To view actual data values, type the appropriate selection criteria (or leave blank to view all data) and press Enter.

— Wally Day  
Senior programmer/analyst  
Miicor Consulting, Inc.  
Boise, Idaho

## Open Browser or Other PC Program from i5/OS

Perhaps the most common question I receive from the forums and from readers of my newsletter is, "How do I open a browser from my System i program?" Most users today use PCs running 5250 emulators to access the System i, and that makes it easy. There's a simple CL command called STRPCCMD that tells the 5250 emulator to run a PC command on the Windows system.

[Figure 10](#) shows a CL program that accepts a UPS tracking number as a parameter. When you call this program, it opens a browser on the PC that points to UPS's website for tracking packages.

A browser is just the beginning! You can run any PC command this way, as long as it's less than 123 characters long.

— Scott Klement  
**System iNEWS** technical editor

## Execute Commands with the system() Function

It's common to use the QCMDEXC or QCAPCMD API when you want to execute a CL command from an RPG program. However, you might find it more convenient to use a C runtime library function called system() to accomplish the same purpose. The system() function passes a command string to the command processor without the need to pass the length of the command string — or any other parameters for that matter.

To call the system() function, you simply pass it a pointer to the command string. [Figure 11](#) shows the suggested prototype (along with some necessary H-specs).

When it's time for your program to execute a command, you can refer to the prototype. The command string may be a variable, literal, named constant, or an expression. [Figure 12](#) shows a typical use.

The return code lets you check for the success or failure of the system() function. The return code is zero if the command is successful or 1 if the command fails. If you pass a null pointer to a string, system() returns -1, and the command processor is not called.

If the system() function fails (i.e., return code is 1), it sets a global variable \_EXCP\_MSGID with the CPF message ID. You can import this variable into your program to check for specific errors ([Figure 13](#)).

To use the system() function, you must refer to binding directory QC2LE when compiling and/or binding the program. The examples name QC2LE in the H-specs.

— *Bryan Meyers*  
**System iNEWS** technical editor

## Processing Several Files with the Same File Specification

Suppose you have an RPG program that needs to read a large number of files and perform the same processing for each of them, and you might not know the file names at compile time. Some of the files have different record lengths, and some of them even have different formats, although the critical fields correspond in each file (e.g., each file format has a field called Tranamount, Signed (13.2)). How can you read and process all these files without having to code separate file specifications for each one?

Try using a combination of the EXTFILE and USROPN F-spec keywords. If you can identify the record format with a record identification field, you can describe each format with input specifications. In the file specification, use the record length of the longest record. [Figure 14](#) shows a skeleton of what your code might look like.

— *Bryan Meyers*  
**System iNEWS** technical editor

## Finding the Day of the Week in RPG

To find the day of the week, you calculate the modulus 7 remainder of the difference in days between a DATE and a known Sunday:

```
day = %rem( %diff( DATE: d'1899-12-30':    *days ): 7 );
```

Day will contain 1 to 6 for Sunday to Friday or 0 for Saturday. You can check for 0 and set to 7 if required.

— Julian Monypenny  
**System iNEWS** technical editor

### Using Keys with Multiple Fields

RPG allows you to specify keys with multiple fields in three ways: key lists, key data structures, and lists of search arguments. A list of search arguments (or search list) is by far the most flexible. You can vary the number of search arguments from one operation to the next with the following:

```
setll (y.a: y.b) x;  
reade (y.a) x;
```

The purpose of each operation is obvious because the key fields are defined on the operations and not somewhere else in the program. Assignments are reduced because you do not have to move the contents of one field to the one specified on the key list or data structure. Additionally, search lists pass keys by value rather than by reference. Type conversion is performed on the list fields, allowing constants, expressions, or fields in different formats to be specified. For example, a zoned numeric search argument would be converted into the packed decimal format specified for a file.

— Julian Monypenny  
**System iNEWS** technical editor

### Do You Know FNDSTRPDM?

The Find String Using PDM (FNDSTRPDM) command has been on the System i for a long time, but it remains mostly unused.

You can use this command to search through a source file or database file for a text string. This in itself is pretty cool, but FNDSTRPDM also has great flexibility. I find it useful for searching through source files to find the occurrence of a field name. The command can produce a listing of all source members in which the field name is found. Once it finds the field, it can do a variety of other things to the source code (e.g., copy, rename, compile, save, delete). You don't have to search for a field name; the command will search for any text string in a source file or a physical file.

— Dan Riehl  
**System iNEWS** technical editor

### See 54 Lines of RPG at a Time

I love coding RPG in WDS*c*. I get many different benefits from it, but one of the best is the amount of code I can see at once. With SEU, I see only 19 lines of code at a time. When WDS*c* is set up with the correct font and the monitor set at 1024x768, I can view 54 lines of code at a time! To set the font to an optimal choice for space and readability, click Window|Preferences|LPEX Editor|Appearance|Change. Select Lucida Console for the font and set the point size to 8.

— Aaron Bartell  
Programmer  
Krengel Technology Inc.  
Mankato, Minnesota

## SEU

### Search Excluded Records in SEU

This technique is handy for searching and viewing only those records in SEU that meet specific search criteria. For example, suppose you are looking for all references to CUSTNAME. Rather than searching for each reference and dealing with all of the intervening code between each instance of the search string, you can look at all of the instances at once. Here are the steps:

1. Edit or view your source member in SEU.
2. Type X999999 in the first sequence number and press Enter (alternatively, you can type XX in the first sequence number and XX in the last sequence number). You will see "xxxxx data records excluded." The xxxxx will be replaced by the true record count. Don't worry, the lines are not deleted.
3. Press F14. Type the search string in the "Find" prompt. Tab down to "Occurrences to process" and change the value to 2=All. Tab down to "Records to search" and change the value to 2=Excluded. Press F16.
4. SEU will reload. You will see the search-matches interspersed with text lines that say "xx records excluded." To get back to the full (original) SEU view, press F5.

It's a neat trick when you need to perform mass changes to a field, and you do not want to mess with extraneous code. This tip is especially helpful in very large programs.

— Wally Day  
Senior programmer/analyst  
Miicor Consulting, Inc.  
Boise, Idaho

## SQL

### Create Meaningful System Column Names

SQL supports column names of more than 10 characters. If a column name exceeds 10 characters, SQL creates a 10-character system name by appending a sequential five-digit number to the first five characters of the column name. This can result in meaningless system names that are difficult to use in RPG and Query. For example, the following system names might be generated for supplier columns:

```
SUPPLIER_NAME = SUPPL00001  
SUPPLIER_ADDR1 = SUPPL00002  
SUPPLIER_PHONE = SUPPL00003
```

To ensure that you have meaningful system names for all the columns in your tables, you specify the For Column clause when defining columns whose names exceed 10 characters:

```
create table supplier (
supplier_name  for column suppname char(30),
supplier_addr1 for column suppaddr1 char(30),
supplier_phone for column suppphone char(20),
... )
```

The system name must be different from the column name, so omit the For Column clause from columns with short names unless you want to remove characters such as an underscore.

— Julian Monypenny  
**System iNEWS** technical editor

## DDS: Creating Long SQL Column Names

If you use DDS to create your files, you can define long column names for use with SQL. To do this, specify the ALIAS keyword for each field whose SQL column name is different from the field name:

```
D SUPPNAME          ALIAS ( SUPPLIER_NAME )
D SUPPADDR1         ALIAS ( SUPPLIER_ADDR1 )
D SUPPPHONE         ALIAS ( SUPPLIER_PHONE )
```

— Julian Monypenny  
**System iNEWS** technical editor

## Use a Subquery to Mimic a Join

SQL statements such as Update and Delete do not support joins. To limit these statements to rows in a matching table, you must use a subquery to mimic a join. To do this with one column is simple:

```
delete from x
where x.a in
(select y.a from y
  where x.a = y.a)
```

However, to mimic a join on multiple columns is more complex because a subquery can return only one result column. You can get around this limitation by matching the queries using a constant and specifying the join in the Where clause of the subquery:

```
delete from x
where 1 in
(select 1 from y
  where x.a = y.a
    and x.b = y.b)
```

— Julian Monypenny  
**System iNEWS** technical editor

## WDS*c*

### Qualifying Physical Files to Speed Development

Qualified data structures in RPG are great, and what I love most about them is the capabilities WDS<sub>c</sub> provides. One is the ability to display all of the fields in a data structure by typing the data structure name, adding the separating period, then pressing Ctrl+Space. This is very helpful when there are 50 or more fields in a data structure. You can also give physical files defined in the F-spec the same qualified data structure capabilities for easy coding in WDS<sub>c</sub>. To do this, first define your file with keyword PREFIX.

```
FORDHDRPF  if    e
k disk      prefix('HDR.')
```

Next, define a qualified data structure that has the same fields as the physical file.

```
D Hdr              e ds
qualified extname(ORDHDRPF)
```

Now, in your code, type 'Hdr.' and press Ctrl+Space to get a list of all fields in ORDHDRPF. This keeps keying errors to a minimum and speeds up development time drastically, because you don't have to go back and forth from one screen to the next trying to remember names of database fields. Note that you have to refresh the Outline view for this to work, because refreshing the perspective will cache locally the definitions of the physical files for WDS<sub>c</sub> to use when you press Ctrl+Space.

— Aaron Bartell  
Programmer  
Krengel Technology Inc.  
Mankato, Minnesota

## Don't Type So Much

When writing Java in Eclipse/WDS<sub>c</sub>, make sure to take advantage of Content Assist, which you can access by pressing Ctrl+Space. Content Assist tries to save you time by typing common pieces of code, such as import statements and variable names. Here are two examples.

**Example 1.** In a Java source file that is empty except for the class declaration, type:

```
private int minute = Calendar.MINUTE;
```

You will notice that the file does not compile because `java.util.Calendar` has not been imported. Instead of typing the import statement yourself, move the cursor to the end of the word `Calendar` and press Ctrl+Space. The Content Assist tool automatically imports `java.util.Calendar` into your class, and the error goes away.

**Example 2.** Often, you need a variable that is the same name as its Java class. For example, you might type something like

```
private StringBuffer stringBuffer;
```

Instead of typing `StringBuffer` twice, simply type

```
private StringBuffer
```

and then press Ctrl+Space. Content Assist automatically adds the variable name and assumes it is the same name as the class you are using but with the correct case.

— Chuck Caplan  
Java developer  
AFBA  
Columbia, Maryland

## Fix Errors More Efficiently

You can access the Eclipse/WDS Sc Quick Fix feature by pressing Ctrl+1. This feature allows you to fix common errors. Here are two examples that should help explain how valuable this feature is.

**Example 1.** Automatically create a method. In your code, you might want to make a call to a method that you have not yet created. Using the Quick Fix feature, Eclipse/WDS Sc automatically creates an empty method based on a method call that you write in your code. For example, if your code contains

```
String request = "hi";  
String response = newMethodCall(request);
```

your class will not compile because there is no method named newMethodCall. However, if you move the cursor to the end of newMethodCall and press Ctrl+1, you automatically create a method called newMethodCall. Doing so adds the following to your Java class:

```
private String newMethodCall(String request) {  
    // TODO Auto-generated method stub  
    return null;  
}
```

Notice that the Quick Fix feature is smart enough to add the String variable as a parameter to the new method.

**Example 2.** Fix common spelling mistakes. Sometimes, you might type so fast that you misspell a class or variable. The following variable declaration does not compile because StringBuffer is spelled incorrectly:

```
private StringBufer f;
```

However, if you place the cursor at the end of the error (in this case, at the end of the word StringBufer) and press Ctrl+1, you will be given several options to fix the mistake. One option is to change the declaration from StringBufer to StringBuffer. Of course, you could also automatically make a class called StringBufer, as explained in Example 1.

— Chuck Caplan  
Java developer  
AFBA  
Columbia, Maryland

## Create a New Template in Remote System Explorer (RSE)

Template support is available for ILE RPG, ILE Cobol, and C++ programs. You can use templates to predefine a structured description of coding patterns that recur in your source code. The LPEX editor supports the use of templates to fill in commonly used source patterns. For example, if you often use a particular coding pattern, you can avoid typing it each time that you want to use it by using a template. Invoking Content Assist at the point of this pattern in your code gives you a list of possible

templates. Selecting a template inserts the code into the LPEX editor. Keep in mind that for RPG source, you can use templates only for free-form RPG.

To create a template:

1. Select Window and click Preferences.
2. Expand Remote Systems|iSeries.
3. Select Templates.
4. Click New.
5. In the new template dialog box, type the name of the template, and select a language from the Context drop-down list.
6. In the Content field, type the code that you want Content Assist to insert.
7. Click OK twice.

To insert this new template type the template name in your code and press Ctrl+Space. The new template appears at the bottom of the list.

— *Carmen Nuland*  
*Senior systems analyst*  
*Hal Leonard Corp.*  
*Winona, Minnesota*

### **Add Code Snippets to Reuse in Remote System Explorer (RSE)**

Click Window|Show View|Other, expand Basic, and select Snippets to add another tabbed view to your screen. Right-click the snippets view and select Customize. Click New to create a new category (drawer). I called mine RPG. This is also where you can hide the other views.

Once you've created the category, right-click the snippets view and select Customize. Click New to create a new item (snippet). Write or paste your code in the window provided. I found that I needed to go back and clean out extra lines for code I pasted in once I had saved it.

To use the snippet, position your cursor in the code and double-click the snippet from the list.

— *Carmen Nuland*  
*Senior systems analyst*  
*Hal Leonard Corp.*  
*Winona, Minnesota*

### **Show Source Date**



By default, the editor doesn't show the source date as SEU does. To show the source date, right-click anywhere in your source code and select Source|Show Date Area. The date column now appears to the right of the source-code line numbers. To remove the date area column, right-click and uncheck Source|Show Date Area.

— *Jef Sutherland*  
**System iNEWS** technical editor

### Filter Source Code Lines

To quickly filter your source code lines to show only lines with a particular string (e.g., a variable name), do the following:

1. Highlight the search string, such as the variable name in your source code.
2. Right-click and select Selected|Filter Selection. The editor now shows only source code lines with the selected search string.
3. To remove the filter, press Ctrl+W.

— *Jef Sutherland*  
**System iNEWS** technical editor

### Mark Your Place

Set a quick mark in your source code so you can quickly return to a marked place in your source code.

1. Put your cursor on the line where you want a quick mark.
2. Press Ctrl+Q. You'll get a message at the bottom of the editor that says, "The quick mark was set at the cursor location."

Move anywhere in your source. Work, work, work! When you're ready to return to the quick mark, press Alt+Q to return the cursor to the line and position where you set the quick mark.

— *Jef Sutherland*  
**System iNEWS** technical editor

**Source URL:** <http://iprodeveloper.com/rpg-programming/tips-programming-and-development>