

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

12 AS/400 Programming Tips and Techniques

[System iNEWS Magazine](#)

[Azubike Oguine](#) [Bryan Meyers](#) [Barbara Morris](#) [Dan Darnell](#) [Dan Boyer](#) [David Robertson](#) [Jaslyn Wang](#) [Rich Tieslau](#)
[Terry Smith](#) [Carsten Flensburg](#) [Bryan Meyers](#) [Mohit Tandon](#) [Carsten Flensburg](#) [Terry Smith](#)

Azubike Oguine

Wed, 12/01/1999 (All day)

Once again another December is upon us, and this year, that means it's nearly time to start celebrating the passing of a century (and the coming of a new one). In case you can't wait until December 31st to start the party, here's something you can celebrate about today: We're back this year with another bundle of nifty AS/400 programming tips and techniques!

For this article, **NEWS/400** readers and technical editors submitted some of their favorite AS/400 programming tips. The result is a bundle of coding goodies just waiting for you to join in the fun.

Enjoy the party — we'll see you next year!

Using SEU to Connect Opcodes with Corresponding ENDxx's

Have you ever added a CASxx, DOxx, IFxx, or Select statement to an RPG program and forgotten to add the corresponding ENDxx? Or perhaps you want to ensure the statement you're coding is associated with the expected ENDxx. If your program is large, finding the correct ENDxx placement can be difficult. For RPG programs, you can select the indented compile listing option to check opcode pairings. Or, for an online method, you can use Source Entry Utility's (SEU's) exclude, hide, and show functions.

To use SEU, first position the program to the desired CASxx, DOxx, IFxx, or Select statement. Place an 'XX' (block exclude) on the first statement number following the CASxx, DOxx, IFxx, or Select. Then, scroll forward or use SEU's search function to find the next opcode that requires an ENDxx. Place 'XX' on the statement preceding that line, and press Enter. All statements between the XX's will be excluded, leaving only the opcodes requiring an ENDxx. (Be sure not to include any ENDxx statements within the exclude block.)

Repeat the process until only the desired opcodes and ENDxx statements are shown. You'll see only the CASxx, DOxx, IFxx, and Select statements and the corresponding ENDxx statements. Any missing ENDxx's will then be apparent.

To quickly remove statements from view that would otherwise need to be included within the exclude blocks, you can use SEU's hide function. For example, enter

```
h eval *all
```

on the SEU command line to remove from view all statements that contain the EVAL opcode.

SEU's show first and show last functions let you redisplay selected lines. Enter SF*n* or SL*n* at the leftmost position of the data record's excluded line to redisplay the first *n* or the last *n* lines. To redisplay all program lines, press command key F5.

Dan Boyer, Professional Staff, Compuware Corporation, Columbus, Ohio

Replacing *ENTRY PLIST

If you're writing your own RPG IV procedures, you're already familiar with prototypes and procedure interfaces. The prototype validates the parameters to pass to a procedure as well as the return value that the procedure will pass back to its caller. The prototype works in conjunction with a procedure interface definition in the called procedure that actually defines the parameters and return value.

Did you know that you can use the same prototype/interface combination for any type of call, not just procedure calls? The procedure interface can make completely obsolete the entry parameter list (*ENTRY PLIST) in a called program, even when you call the program dynamically. Not only do you gain the advantage of validating parameters at compile time when you use this technique, but using a procedure interface also organizes the definition of the parameter list into the D-specs, centralizing all the data definition into one place.

Be sure to include the EXTPGM keyword in your prototype to identify the program to call. The code in [Figure 1A](#) highlights the requirements in the called program. Previously, you would have coded an *ENTRY PLIST for this purpose ([Figure 1B](#)).

You'll also need a corresponding prototype in the calling program (include the prototype in a /COPY member). Use the CALLP (Call prototyped program or procedure) opcode instead of the CALL opcode to call the program ([Figure 1C](#)).

Bryan Meyers, Senior Technical Editor, NEWS/400

Update JDK Release Levels

When IBM shipped OS/400 V4R2, the Java Development Kit (JDK) release level of the AS/400 Java Virtual Machine (JVM) was 1.1.4. A PTF is available to bring the JDK in V4R2 up to the JDK 1.1.7 release level (the level shipped with V4R4). The PTF number is SF52615 for product 5769-JV1 (AS/400 Developer Kit for Java).

The JDK shipped with V4R3 is at release level 1.1.6. To bring that JDK to the 1.1.7 level, apply PTF SF52890.

IBM has promised to support the JDK 1.2 release (also called Java 2) on the AS/400, and that version should be available as you read this. Search the PTF cover letters at the IBM AS/400 Technical Support Web site (<http://as400service.rochester.ibm.com>) to determine whether this most recent JDK update is now available.

Dan Darnell, Technical Editor, NEWS/400

RPG Compares and Sequencing

The AS/400's RPG compilers offer some interesting options for controlling the way characters are compared or sorted when you run a program. If a specific sort sequence has already been applied to a

file read by the program, the file opcodes used against the file (CHAIN, SETLL, SETGT, READE, and READPE) work in accordance with the file's sort attributes (shown in [Figure 2A](#)).

If you want the behavior of RPG's compare opcodes (ANDxx, CABxx, CASxx, COMP, DOU, DOUxx, DOW, DOWxx, IF, IFxx, ORxx, WHEN, and WHENxx) to match the behavior of the file opcodes, you can use RPG's ALTSEQ/SRTSEQ keywords to accomplish this.

The ALTSEQ/SRTSEQ keywords also let you ignore case and handle national-language special characters in compare (sequence checking), array sorting (SORTA), and lookup (LOOKUP) operations. Specify ALTSEQ(*NONE) on the D-spec to exclude an array from H-spec alternate-collating sequence.

To see this tip in action, compile the source in [Figure 2B](#) and then place the program in debug mode using the STRDBG (Start Debug) command. Then use the step function (F10) to step through the program and see how the execution is affected by the settings applied.

Carsten Flensburg, Programmer/Analyst, Novasol Data AS, Copenhagen, Denmark

One AS/400, Multiple JDKs

In V4R4, you can have more than one release of the Java Development Kit (JDK) installed on your AS/400. This ability is extremely useful when you want to keep production applications running on one tried-and-true JDK release while developing code using a newer release.

To determine which release of the JDK to use, the AS/400's Java Virtual Machine (JVM) looks at the java.version property. For example, to execute the MyClass class using JDK 1.1.6, use the following java command in the QShell Interpreter:

```
java -Djava.version=1.1.6 MyClass
```

To run the same class using JDK 1.1.7, just change the java.version property setting:

```
java -Djava.version=1.1.7 MyClass
```

If you omit the java.version parameter, the JVM uses the most recently installed JDK release.

*Dan Darnell, Technical Editor, **NEWS/400***

Returning Retrieve API Output to a User Space

OS/400's retrieve APIs usually require you to specify the field into which the API will place the returned information. For APIs whose returned information has a fixed format, this requirement isn't a problem; you can easily specify a field as large as the amount of information you need. For certain APIs, however, the amount of information to be returned is unknown beforehand. Two examples are the QDBRTVFD (Retrieve Database File Description) API and the QDFRTVFD (Retrieve Display File Description) API. The amount of information each of these APIs can return varies widely depending on the attributes of the file specified in the call to the API.

To ensure you receive all available information without wasting memory space in your program, you can have the API write its returned value to a user space. [Figure 3](#) shows a piece of code for writing a display file's description to a user space. First, create the user space using the QUSCRTUS (Create User Space) API. Then set it for automatic extendibility using the QUSCUSAT (Change User Space

Attribute) API. With this setting, the system automatically extends the user space to accommodate all the information returned by the API.

Next, call the QUSPTRUS (Retrieve Pointer to User Space) API to retrieve a pointer to the user space into the pointer field of the RtnVar field. This maps the RtnVar field over the user space. Now, call the QDFRTVFD API (or some other retrieve API), specifying RtnVar as the field in which to return information. Doing so causes the API to write the returned information to the user space. To determine the amount of information returned by the API, you can use the returned information's Bytes Returned field.

Azubike Oguine, Analyst, Chevron Nigeria, Ltd.

API Parameters: Optional vs. Omissible

The OS/400 API manuals describe two kinds of optional parameters: those listed as "optional" and those listed as "omissible." You can leave optional parameters completely out of an API call, but any omissible parameters must be passed. "Omissible" simply means you can pass a null pointer as the parameter; if you're using RPG, you pass *OMIT as the parameter.

If you neglect to pass an omissible parameter, unexpected and unpredictable results can occur, such as storage corruption in mysterious places, storage-violation errors, or receiving the message "MCH3601 Pointer not set for location referenced."

Barbara Morris, IBM, Toronto, Ontario, Canada

Enable Restore Display Feature When Creating a Display File

As shipped by IBM, the default version of the CRTDSPF (Create Display File) command sets the command's RSTDSP (Restore display) parameter to *NO. If you have a comprehensive display program that calls screen to screen, it's better to set this parameter to *YES.

You can set parameter RSTDSP to *YES when you create the display file, or, to make *YES the permanent default parameter value, simply use the CHGCMDDFT (Change Command Default) command as follows to change the parameter's default value to *YES:

```
CHGCMDDFT  CMD (CRTDSPF)          +
          NEWDFT ( 'RSTDSP (*YES) ' )
```

Jaslyn Wang, Senior System Analyst, Monsanto Singapore Company, (Pte) Ltd., Singapore

Century Keying Shortcut

When a company's dates all fall into a 100-year period, windowing techniques are often used for Y2K compliance to avoid having users key in a century each time they enter a date. If you need a wider range of dates, you can default the century to the current one to ease the data-entry burden.

The RPG IV code in [Figure 4](#) lets a user key in either a six- or an eight-digit date. If a user enters "123199" during a year that begins with 19 (e.g., 1999), the program stores the date as 12311999. If a different century is required, the user can key it in explicitly — 12312099 or 12311899, for instance. The program validates entered dates before writing them to the file.

David A. Robertson, Systems Analyst, Highlander, Inc., Chatfield, Minnesota

Simplify HLL Calls in Net.Data

The Net.Data *direct call* language environment, released as a PTF for Net.Data in March, provides an easy way to call AS/400 programs. To call an AS/400 program, simply follow these steps:

1. Define the program and its parameters, as shown in the example in [Figure 5A](#). You can specify the entire path for the program (e.g., /Qsys.Lib/MyLib.Lib/GetCustSts.Pgm); however, a better way to specify the program library (or libraries) is through the EXEC_PATH variable in your Net.Data initialization. This way, you call the right version (e.g., test, production) without having to change code within the macro.

2. Call the defined program in a subsequent function within the macro, as in [Figure 5B](#). The original PTF that provided support for the direct call language environment worked well with C programs because it terminated all character-string arguments with a null byte (X'00'). The latest Net.Data PTFs (released in June) now provide a built-in variable (DTW_PAD_PGM_PARMS) that you can set to "YES" to specify that character-string arguments should be padded with blanks, simplifying calls to RPG and Cobol programs:

```
%Define DTW_PAD_PGM_PARMS="YES"
```

As of June 17, the PTFs for the direct call language environment are:

Release	Product	PTF
V4R4M0	5769DG1	SF57237
V4R3M0	5769DG1	SF57236
V4R2M0	5769TC1	SF57198
V4R1M0	5769TC1	SF57197
V3R7M0	5716TC1	SF57207
V3R2M0	5763TC1	SF57206

Additional documentation and examples for the direct call language environment are available at the Net.Data Web site, <http://www.as400.ibm.com/netdata>.

*Terry Smith, Technical Editor, **NEWS/400***

This tip is reprinted from the newsletter **AS/400 e-Developer**, September/October 1999.

Converting Character Data to Numeric in Query/400

Query/400 does not provide a native method for converting character data to numeric, but with the following hack, the conversion is possible. I have a character field (four digits) that's used for generic purposes in a particular database file. Although the field's format is character, the field sometimes represents numeric data. To perform math operations on such data in Query, you must first convert the field to numeric. By using the MICROSECOND keyword in the Define Result Fields section of Query, you can convert the character data to a numeric field for later use in calculations.

For example, create a field called TIMESTAMP using Query's Define Result Fields panel, specifying

```
'1988-12-25-17.30.00.00' || CHARFIELD4
```

in the panel's Expression column. The field CHARFIELD4 in this example represents a four-digit character field containing the characters 0 (zero) through 9 (nine).

Next, use the MICROSECOND keyword on the TIMESTAMP field to create a numeric copy of CHARFIELD4's character data. [Figure 6](#) shows what the Define Result Fields panel will look like. The MICROSECOND keyword takes the six rightmost digits in the TIMESTAMP field and converts them to numeric. In the example, the character field is only four digits long, so 00 is concatenated with the character field to create a valid timestamp with six digits of milliseconds.

If a given character field is larger than six digits, you'll need to break down the above process into multiple steps and perform a math operation on the individual numeric components to combine them.

Rich Tieslau, Auto Ventshade Company, Lawrenceville, Georgia

Shortcut Access to AS/400 Commands

Have you ever wanted to access the AS/400 command line from within an interactive SQL session — or run an AS/400 system command from within the Interactive Source Debugger (ISDB) or an AS/400 File Transfer Protocol (FTP) session? You can easily do so.

To access the AS/400 command line from within an SQL session, simply call program QCMD from the SQL command line.

To run an AS/400 command from within the ISDB, call the command prefixed by "SYS" and a blank. For example, to have a look at your spool files, you can run the command

```
SYS WRKSPLF
```

from the ISDB command line.

Similarly, to run an AS/400 system command from within an FTP session, call the command prefixed by "SYSCMD" and a blank. At V4R3 and later, you can also reach an AS/400 command line from an FTP session by typing "F21" (for F21=Command Line) at an FTP command line. At V4R3 and later, you can also reach an AS/400 command line from an FTP session by pressing F21 at an FTP command line.

Mohit Tandon, AS/400 Consultant, Tata Consultancy Services, Noida, India

Source URL: <http://iprodeveloper.com/development/12-as400-programming-tips-and-techniques>