V5R3 Control Language
STUNNING DEVELOPMENTS

By Dan Riehl and Guy Vig

In V5R3, IBM added 57 new CL commands and changed more than 240 commands. This isn't extraordinary because IBM enhances the set of CL commands every release. The big news is that some of the new and changed commands provide CL compiler features that CL application developers have been clamoring to see for many years. So let's look at the really cool CL stuff in V5R3.

New Integer Data Types

It has always been a pain to call IBM APIs from CL programs or to use CL programs as exit programs to IBM functions because almost all the APIs and exit program routines use binary integer parameters. You can use the built-in function %BIN to load numbers into character *CHAR variables and employ the *CHAR variables as replacements for binary integer parameters. Though this technique works, it's always been a clumsy patch.

V5R3 solves that problem with two new binary data types: signed and unsigned integers, which can be either two or four bytes in length. So now your DCL (Declare CL Variable) statements can include the new binary data types *INT and *UINT, as **Figure 1** shows.

Structured Programming Enhancements

One of CL's weak points has always been its lack of structured operations. You were limited to IF, ELSE, and GOTO. V5R3 finally brings CL program control flow close to what you can do in other languages by providing five new commands: DOWHILE, DOUNTIL, DOFOR, LEAVE, and ITERATE.

DOWHILE (test before loop). **Figure 2** shows the command syntax for the DOWHILE command. The DOWHILE COND parameter has the same rules as those for the IF command's COND parameter. Though you can specify up to 25 nesting levels, the effective nesting level probably should not exceed five or six. Excessive nesting makes code difficult to read, write, and debug. To end the DOWHILE processing, you use an ENDDO command:

```
DCL    VAR(&EOF)   TYPE(*LGL)   VALUE('0')

RCVF
MONMSG CPF0864   EXEC(CHGVAR    &EOF   '1')

DOWHILE    COND(*NOT &EOF)

     IF   (*NOT &EOF)   DO
          … do some commands
     ENDDO
     RCVF
     MONMSG CPF0864   EXEC(CHGVAR    &EOF   '1')

ENDDO
```

DOUNTIL (test after loop). **Figure 3** shows the command syntax for the DOUNTIL command. The key difference between DOWHILE and DOUNTIL is when the testing of the loop condition occurs. DOWHILE performs the test before the commands are executed; DOUNTIL tests after the commands are executed. So the commands within a DOUNTIL loop will always be executed at least once. Another difference is that DOUNTIL will continue looping so long as the loop condition evaluates as logically false, whereas DOWHILE will keep looping while the loop condition evaluates as logically true. Following is an example of a DOUNTIL command structure:

```
DCL    VAR(&EOF)   TYPE(*LGL)   VALUE('0')

DOUNTIL    COND(&EOF)

     RCVF
     MONMSG CPF0864   EXEC(CHGVAR    &EOF   '1')

     IF   (*NOT &EOF)   DO
          … do some commands
     ENDDO

ENDDO
```

DOFOR. **Figure 4** shows the command syntax for the DOFOR command. The DOFOR command has three required parameters — VAR, FROM and TO — and an optional parameter — BY — which defaults to +1. As the example in **Figure 5A** shows, the DOFOR loop will be performed 100 times. The variable &Counter will be set to the current iteration index, which is incremented by +1 on each iteration. The ENDDO command ends a DOFOR loop. The DOFOR loop in **Figure 5B** will be performed 10 times. The &Counter variable will start at 10 and will count down by -1 until &Counter is less than 1. **Figure 5C** shows that the DOFOR loop will use variables to set the FROM and TO values. These variables must be declared with *INT or *UINT specified for the TYPE parameter.

LEAVE. **Figure 6** shows the command syntax for the LEAVE command. The LEAVE command interrupts the loop processing of DOWHILE, DOUNTIL, or DOFOR. You may supply an optional label name, in which case, processing continues with the command following the ENDDO associated with the DOWHILE, DOUNTIL, or DOFOR command at the specified label name. In the following example, the DOFOR loop will be interrupted if variable &Check contains all blank characters, and control will pass to the command following the ENDDO command:

```
DCL    VAR(&Counter)   TYPE(*INT)   LEN(2)
DCL    VAR(&Check)   TYPE(*CHAR)   LEN(10)

DOFOR    VAR(&Counter)   FROM(1)   TO(10)
 … some commands here…
 IF    COND(&Check = ' ')    THEN(LEAVE)
  …some CL commands here…
ENDDO
```

Figure 1: Example DCL statement with new signed and unsigned integers

```
DCL  &Offset  TYPE(*UINT)        /* Unsigned 4-byte binary integer  */
DCL  &Offset  TYPE(*UINT) LEN(2) /* Unsigned 2-byte binary integer  */
DCL  &Length  TYPE(*INT)         /* Signed 4-byte binary integer    */
DCL  &Length  TYPE(*INT) LEN(2)  /* Signed 2-byte binary integer    */
```

Figure 2: DOWHILE command syntax

```
DOWHILE     COND(Logical variable or Logical expression)
```

Figure 3: DOUNTIL command syntax

```
DOUNTIL     COND(Logical variable or Logical expression)
```

Figure 4: DOFOR command syntax

```
DOFOR     VAR(Integer variable)   +
                FROM(Integer constant, variable, or expression)   +
                TO(Integer constant, variable, or expression)   +
                BY(Integer constant)
```

Figure 5A: DOFOR loop set to perform 100 times

```
DCL   VAR(&Counter)  TYPE(*INT)  LEN(2)

DOFOR   VAR(&Counter)  FROM(1)  TO(100)
 …   some CL commands
ENDDO
```

Figure 5B: DOFOR loop set to perform 10 times

```
DCL   VAR(&Counter)  TYPE(*INT)  LEN(2)

DOFOR   VAR(&Counter)  FROM(10)  TO(1)  BY(-1)
 …   some CL commands
ENDDO
```

Figure 5C: FROM and TO values set in DOFOR loop

```
DCL   VAR(&Counter)  TYPE(*INT)  LEN(2)
DCL   VAR(&Bottom)  TYPE(*INT)  LEN(2)
DCL   VAR(&Top)  TYPE(*INT)  LEN(2)

DOFOR   VAR(&Counter)  FROM(&Bottom)  TO(&Top)  BY(1)
 …   some CL commands
ENDDO
```

Figure 6: LEAVE command syntax

```
LEAVE    CMDLBL(Label Name)
```

Figure 7: Specifying the nesting level to return to

```
DCL   VAR(&Counter)  TYPE(*INT)  LEN(2)

TEST1: DOFOR   VAR(&Counter)  FROM(1)  TO(10)  BY(2)
 … some CL commands here
TEST2:    DOWHILE   COND(*NOT &EOF)
            … some CL commands here
             IF   COND(&Check = ' ')  THEN(LEAVE  CMDLBL(TEST1))
            … some CL commands here
          ENDDO
            … some CL commands here
       ENDDO
Next1:
```

Figure 8: ITERATE command syntax

```
ITERATE   CMDLBL(Label Name)
```

Figure 9: Specifying the DO loop to which ITERATE applies

```
TEST1: DOFOR   VAR(&Counter)  FROM(1)  TO(10)  BY(2)
 … some CL commands here
TEST2:    DOWHILE   COND(*NOT &EOF)
            … some CL commands here
             IF   COND(&Check = ' ')  THEN(ITERATE CMDLBL(TEST1) )
            … some CL commands here
          ENDDO
            … some CL commands here
       ENDDO
```

If you use nesting, you can specify the label name to indicate which nesting level to return to. In **Figure 7**, control passes to the command following the ENDDO command that's paired with the DOFOR command at label name TEST1, which in effect, passes control to label Next1.

ITERATE. **Figure 8** shows the command syntax for the ITERATE command. The ITERATE command causes a quick exit from the loop processing of DOWHILE, DOUNTIL, or DOFOR. You may supply an optional label name, in which case, processing continues with the ENDDO command associated with the DOWHILE, DOUNTIL, or DOFOR command at the specified label name. In the following example, if variable &Check contains all blank characters, control passes to the ENDDO command, and the loop is performed again unless the limit has been reached:

```
DOFOR VAR(&Counter) FROM(1) TO(10)
 … some commands here…
 IF COND(&Check = ' ')
THEN(ITERATE)
  …some CL commands here…
ENDDO
```

If you use nesting, you can specify the label name to indicate which DO loop the ITERATE command applies to. In **Figure 9**, control passes to the ENDDO command that's paired with the DOFOR command at label TEST1. If the DOFOR limit value has not been reached, the loop is executed again.

SELECT /WHEN /OTHERWISE. The SELECT command allows for a case structure that evaluates several conditions and performs several dependent actions (**Figure 10**). The SELECT command has no parameters. Each WHEN command defines a condition that's tested within a SELECT structure. If the first WHEN condition is true, the command associated with the WHEN command is performed (which may be a group of commands if the command associated with WHEN is a DO command), and control is passed to the command that follows ENDSELECT. If the first WHEN condition is not true, each successive WHEN command is checked until one is found that has a condition that's true or the last WHEN command in the SELECT structure has been checked. If a WHEN command is

Photo by Léo Lefebvre

Guy Vig at the IBM booth, during the Fall 2004 COMMON Expo, Metro Toronto Convention Centre

checked and has a condition that's true, and no command is specified (a null THEN), control is passed to the command that follows ENDSELECT. The OTHERWISE command specifies the action to take if no WHEN condition is true. OTHERWISE isn't required in a SELECT structure.

Support for Multiple Files

In the past, CL allowed only one DCLF (Declare File) command within a program. With V5R3, you can now specify up to five DCLF commands within a program, effectively increasing the number of files that can be processed in a CL program to five. When you process more than one file, specify the optional OPNID (Open file identifier) parameter, as **Figure 11** shows. Each OPNID must be unique within the program. If the MYDISPLAY file contains a field named CUSTNAME, the field &CUSTNAME is automatically declared in the CL program, just as it has always been. If the file CUSTFILE contains the field CUSTNAME, it will be declared in the CL program as &CUST_CUSTNAME.

The field name is always prefixed by the OPNID name followed by an underscore. The CL variables that DCLF generates can be up to 22 characters long, but CL variables declared using the DCL command are still restricted to 11 characters (10 characters plus the leading ampersand). So, to move the database field to the display field, the command is:

```
CHGVAR    &CUSTNAME    &CUST_CUSTNAME
```

Larger *CHAR Variables

The maximum size of character variables in a CL program has been 9,999 bytes. V5R3 raises that limit to 32,767 bytes. However, the DCLF command will only declare character variables for fields up to only 9,999 bytes. If an initial value is assigned to a character field, the maximum length of the initial value continues to be 3,000 characters.

More Parameters

Before V5R3, the limit of incoming parameters allowed on the PGM (Program) and TFRCTL (Transfer Control) commands was 40, and the limit for outgoing parameters on the CALL (Call Program) command was 99. Under V5R3, the new limit for the PGM, CALL, and TFRCTL commands is raised to 255 parameters.

Pass Parameters by Value

You can now pass parameters by reference or by value on command CALLPRC (Call Procedure). In the past, parameters had to be passed by reference, in effect, passing a pointer to the variable storage area. Here's an example of passing a parameter by value:

```
DCL   VAR(&ITEM) TYPE(*CHAR) LEN(50)
DCL   VAR(&PRICE)   TYPE(*DEC)   LEN(10 2)
DCL   VAR(&RTNCODE)   TYPE(*INT)   LEN(4)

CALLPRC    PRC(GET_PRICE) +
           PARM((&ITEM *BYVAL) (&PRICE)) +
           RTNVAL(&RTNCODE)
```

In this example, a copy of the value in variable &ITEM is passed to the GET_PRICE procedure. A pointer to variable &PRICE is also passed, which will allow the GET_PRICE procedure to change the value stored in the &PRICE variable. The maximum number of parameters that can be passed on CALLPRC remains at 300.

Help Text Generator for User-Written Commands

How many of your custom-written CL commands have online help? IBM-supplied commands all have online help, and your commands can have the same type of online help text that IBM provides.

iSeries NEWS has published many articles over the years that address the creation of User Interface Manager (UIM) help text for display files and CL commands. But most folks who write their own CL commands don't take the time to add online help because it's been a pain to learn and write the UIM markup language for the help text.

In V5R3, you can use a new tool not only to help provide help text for your own commands, but also to generate HTML command documentation using the online help text for your own commands or IBM-supplied commands. The new CL command GENCMDDOC (Generate Command Documentation) can generate UIM markup language source code or HTML source code. The following command will generate an HTML file containing complete documentation for the IBM command CRTDTAARA (Create Data Area):

```
GENCMDDOC CMD(QSYS/CRTDTAARA)
```

If you specify to use the command defaults, the generated HTML file will be stored in a stream file named QSYS_CRTDTAARA.html in the current working directory of the job.

The GENCMDDOC command in **Figure 12** will generate the UIM markup language for MYCOMMAND. The UIM source code will be stored in source file MYDOCS/QPNLSRC in member MYCOMMAND.

The help text that the command generates won't have the textual descriptions or much of the textual content of the full help document. But you will get all the UIM tags for the help sections, including command-level help, help for each parameter with

Figure 10: SELECT command structure

```
DCL    VAR(&Option)  TYPE(*INT)  LEN(2)

SELECT
  WHEN    COND(&Option = 1)  /* Do nothing */
  WHEN    COND(&Option = 2)  THEN(CALL GLMAST)
  WHEN    COND(&Option = 3)  THEN(DO)
        … some CL commands here
         ENDDO
OTHERWISE  CMD(SNDPGMMSG MSG('INVALID OPTION'))
ENDSELECT
```

Figure 11: Specifying the OPNID parameter

```
DCLF  MYDISPLAY     /* Optionally you can specify OPNID(*NONE) */
DCLF  CUSTFILE  OPNID(CUST)

RCVF    OPNID(CUST)
MONMSG    CPF0864  EXEC(GOTO EOF)

CHGVAR    &CUSTNAME     &CUST_CUSTNAME

SNDRCVF   /* When no OPNID is specified,                    */
          /* the file using *NONE as the OPNID is processed.*/
          /* In this case, that file is MYDISPLAY.          */
```

Figure 12: Generating UIM markup language for MYCOMMAND

```
GENCMDDOC    CMD(MYCOMMAND)
             TODIR('/QSYS.LIB/MYDOCS.LIB/QPNLSRC.FILE')
             TOSTMF('MYCOMMAND.MBR')  GENOPT(*UIM)
```

default values and special values, help for command examples, and help for the error messages that a user could monitor for.

Now all you need to do is edit the generated UIM, create the command help panel group by running the CRTPNLGRP (Create Panel Group) command, and re-create your command specifying the HLPID (Help identifier) and HLPPNLGRP (Help panel group) parameters to associate your command with your online help panel group.

The GENCMDDOC command is a byproduct of IBM's work effort in V5R3 to provide better accessibility and navigation for CL command documentation in the iSeries Information Center. GENCMDDOC provides a simpler-to-use interface to the Java CommandHelpRetriever class, which is part of the Java Toolbox support included with OS/400.

The Java Toolbox code uses XML and eXtensible Stylesheet Language Transformations (XSLT) to generate the HTML or UIM output.

Using the New Features Sooner

The rule of thumb has always been that you can't take advantage of CL functions added in the newest release if you plan to take your compiled CL objects back to an earlier release. However, V5R3 is *not* a "business as usual" release for the CL compiler.

The folks in Rochester are aware that customers won't be able to move all their systems to V5R3 at once and, therefore, might have to defer using the new CL compiler features for a year or more, depending on the size of the customer's enterprise.

The Rochester team is working on a set of PTFs to enable users to employ the new CL functions (except for multiple files and the GENCMDDOC command) in CL source code compiled *on a V5R3 system* specifying a target release of V5R2M0.

At the time this article went to press, the V5R3 PTF numbers were SI13505, SI13508, and SI13509. Two V5R2 PTFs, SI13416 and SI13417, provide fixes needed to the CL runtime to handle the new features. The V5R2 PTFs will *not* let you compile CL source code that includes the new features on a V5R2 system. You must do the compile on a V5R3 system specifying TGTRLS(V5R2M0) or TGTRLS(*PRV), save the CL objects specifying TGTRLS(V5R2M0) or TGTRLS(*PRV), and restore the objects on a V5R2 system that has the V5R2 PTFs applied.

Where Do We Go from Here?

More CL enhancements are planned for future releases, which may include subroutines, more data types (including pointers), more built-in functions, and some form of data structures. The goal is to make CL a multipurpose scripting language capable of accessing all IBM APIs, including both program-level and procedure-level APIs, which other ILE languages, such as ILE RPG IV and ILE C, can call today. TNG

Dan Riehl *is an iSeries NEWS technical editor, as well as president of and an instructor at The 400 School (www.400school.com). You can e-mail Dan at driehl@400school.com.*

Guy Vig *is a senior software engineer with IBM Systems Group. He has worked in Rochester, Minnesota, and Toronto, Ontario, on various parts of the operating system and compiler products for System/38, AS/400, and iSeries since joining IBM in 1978. As the "CL Architect," he has worked on revitalizing the CL language on iSeries. You can e-mail Guy at gwvig@us.ibm.com.*

Reprinted from an article which appeared in the July 2004 issue of iSeries News magazine.