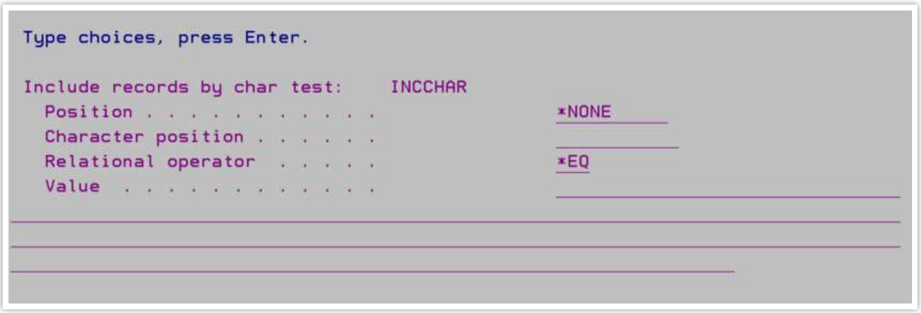
THE FOUR HUNDRED
Power Systems & IBM i Insight

GURU: USING MIXED LISTS TO ADD "DATA STRUCTURES" TO CL COMMANDS

January 15, 2024 Ted Holt

I can't remember the last time I worked on an RPG program that had no data structures, but it was probably on a System/34. Everybody uses data structures, and with good reason – they are as handy as a pocket. When writing CL commands, it is possible to include parameters that are formatted as data structures. IBM calls them mixed lists. In the following paragraphs, I show how this is accomplished.

You may not realize it, but you have used IBM-supplied commands that have mixed-list parameters. For example, the Copy File (CPYF) command has several such parameters: FROMKEY, TOKEY, INCCHAR, INCREL, and SRCSEQ. Here's the prompt for the INCCHAR parameter:


```
Type choices, press Enter.

Include records by char test:    INCCHAR
  Position . . . . . . . . . . .             *NONE
  Character position . . . . . .
  Relational operator  . . . . .             *EQ
  Value  . . . . . . . . . . . .
```

SIMPLE MIXED LISTS

To define a mixed list parameter requires the ELEM (element) keyword, like this:

```
             CMD        PROMPT('Do It')
             PARM       KWD(INCCHAR) TYPE(INCCHAR_T) +
                          PROMPT('Include records by char test')
 INCCHAR_T:  ELEM       TYPE(*CHAR) LEN(10) RSTD(*NO) DFT(*NONE) +
                          SPCVAL((*RCD) (*FLD)) EXPR(*YES) +
                          PROMPT('Field')
             ELEM       TYPE(*INT4) RSTD(*NO) EXPR(*YES) +
                          PROMPT('Character position')
             ELEM       TYPE(*CHAR) LEN(3) RSTD(*YES) DFT(*EQ) +
                          VALUES(*EQ *NE *GT *GE *LT *LE *NG *NL *CT) +
                          PROMPT('Relational operator')
             ELEM       TYPE(*CHAR) LEN(256) EXPR(*YES) +
                          PROMPT('Value')
```

There are four elements, each defined with its own data type, prompt string, and so forth. I don't have IBM's source code, so this is one way IBM may have defined that parameter.

This story contains code, which you can download here.

TFH Volume: 34 Issue: 02

TABLE OF CONTENTS

The command-processing program (CPP) receives a block of memory with five values in it. The first value is a two-byte integer that tells how many elements are in the mixed list. In this case, that value would be four. I always ignore this field because I don't need it for anything. The four input-field values follow.

Here's how the INCCHAR parameter could be processed with a CL CPP.

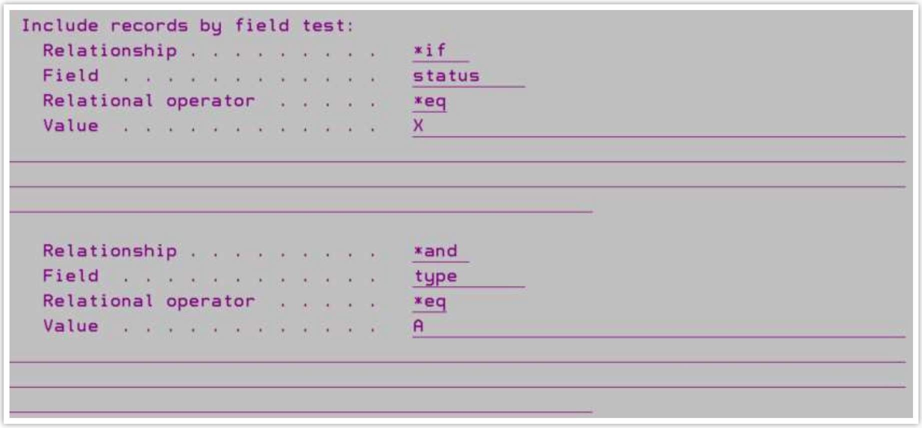
```
pgm parm(&IncChar)

dcl var(&IncChar)  type(*char)               len(275)
dcl var(&Field)    type(*char) stg(*defined) len( 10) DefVar(&IncChar  3)
dcl var(&Position) type(*int)  stg(*defined) len(  4) DefVar(&IncChar 13)
dcl var(&RelOper)  type(*char) stg(*defined) len(  3) DefVar(&IncChar 17)
dcl var(&Value)    type(*char) stg(*defined) len(256) DefVar(&IncChar 20)
```

The program receives the parameter as a single character string. I used defined storage to overlay the parameter with descriptions of the four elements. Defined storage is not exactly like an RPG data structure, but it serves the same purpose.

ARRAYS OF MIXED LISTS

If you like, you can pass an array of mixed lists to the CPP. I can show you what that looks like. Here's the prompt for the INCREL parameter of CPYF.



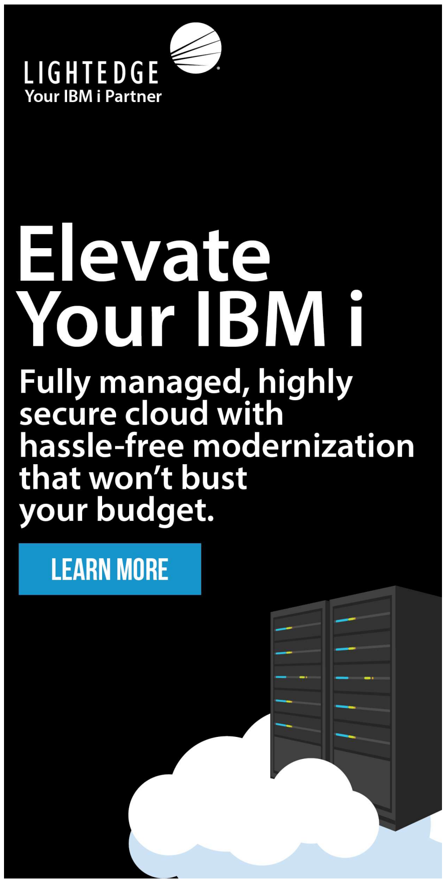
```
Include records by field test:
  Relationship . . . . . . . . .   *if
  Field  . . . . . . . . . . . .   status
  Relational operator  . . . . .   *eq
  Value  . . . . . . . . . . . .   X

  Relationship . . . . . . . . .   *and
  Field  . . . . . . . . . . . .   type
  Relational operator  . . . . .   *eq
  Value  . . . . . . . . . . . .   A
```

When you define an array of mixed lists, the command processor uses a different parameter structure. I think the best way to explain this process is with an example, so let me tell you about a project I worked on recently.

I was trying to understand the "logic" of a 40-year-old poorly-written RPG program. (Think GOTO and indicators.) I added a printer file to the program to log the values of variables, something like this:

```
 1 A Y 12.50 Y RA Y 12345
 2 A Y 12.50 Y RA Y 12345
 3 A Y  8.00 Y RA Y 12345
 4 C Y  6.00 Y DF Y 12345
 5 A Y       N DF Y 44444
 6 A Y 12.50 Y DF N 44444
 7 A Y 12.50 Y DF N 44444
 8 A Y 12.50 Y DF N 44444
 9 B Y       N RA N 44444
10 C N  8.00 Y RD N 12345
11 B Y  8.00 Y RD N 44444
12 A Y 12.50 Y RD N 44444
13 A N 12.50 Y DF N 12345
14 A N  6.00 N TS Y 12345
15 A N  6.00 Y DF Y 44444
16 A N 12.50 Y RD Y 44444
17 B N  8.00 Y TS Y 44444
18 C Y 12.50 Y TS Y 12345
19 C Y  8.00 Y TS N 44444
```


LIGHTEDGE
Your IBM i Partner
Elevate Your IBM i
Fully managed, highly secure cloud with hassle-free modernization that won't bust your budget.
LEARN MORE

CONTENT ARCHIVE

The Four Hundred
Four Hundred Stuff
Four Hundred Guru

```
20 B Y       N RA N 12345
21 C Y  6.00 Y RD Y 44444
22 C Y       Y RA N 77777
23 A Y 12.50 N TS Y 12345
24 A Y  6.00 Y RA N 33333
```

The spooled file told me about the various conditions and the output they led to without my having to unravel all the spaghetti code.

The Display Spooled File (DSPSPLF) command showed me the report, but did not give me a way to zero in on pertinent information. What I really needed was a way to see only the lines that met certain criteria. To accomplish that, I wrote a one-off utility, which I'll call SSF (Search Spooled File) for this example. SSF consisted of two objects – a command and a CL program.

My command allowed me to enter from one to three search criteria. For example:

```
SSF FILE(QSYSPRT) JOB(*) SPLNBR(*LAST) +
       POS1(6)  VAL1(N) +
       POS2(16) VAL2(TS) +
       POS3(19) VAL3(Y)
```

This says, "Show me only the report lines that have an N in position 6, TS in 16 and 17, and a Y in position 19."

```
*...+....1....+....2....+
14 A N  6.00 N TS Y 12345
17 B N  8.00 Y TS Y 44444
```

It's far from elegant, but it was adequate for my purposes and it took very little time and effort to throw together. The short investment in building this tool paid off handsomely, saving me a lot of time and my client a lot of money.

Later, on my own time, I decided to write a proper utility to selectively display lines of a spooled file. The new version of SSF includes these improvements:

- It supports a full set of relational operators: *EQ *NE *GT *GE *LT *LE *NG *NL *CT.
- It allows and/or/not logic
- It allows conditions to be grouped with parentheses.
- It uses an array of mixed lists to specify the search criteria.

The previous example looks like this under the new version of SSF.

```
SSF FILE(QSYSPRT) JOB(*) SPLNBR(*LAST) +
       SELECT(( *N   *N *N  6 *EQ N  ) +
              ( *AND *N *N 16 *EQ TS ) +
              ( *AND *N *N 19 *EQ Y  ))
```

The *N token means that I am not supplying a value for that element of the mixed list. Yes, this syntax is a bit cryptic, but that's CL, and it's not hard to key these parameters properly when you prompt the command.

Now I can do all sorts of searches. Here's another example:

```
SSF FILE(QSYSPRT) JOB(*) SPLNBR(*LAST) +
      SELECT((*N   *N *N   4 *EQ A) +
             (*AND *N '(' 16 *EQ DF) (*OR *N *N 16 *EQ TS ')'))
```

This says, "Show me the report lines that have an A in column 4 and either DF or TS in 16 and 17.

Now let's look at the code that makes this magic happen. First, here's how the mixed list is defined.

```
 CMD        PROMPT('Search Spoole File')
 PARM  SELECT TYPE(SELECTION) MAX(12) PROMPT('Selection criteria')

SELECTION: +
  elem TYPE(*CHAR) LEN(4) RSTD(*YES) VALUES(' ' *AND *OR) +
          EXPR(*YES) PROMPT('And/Or')
  elem TYPE(*CHAR) LEN(4) RSTD(*YES) VALUES(' ' *NOT) +
          EXPR(*YES) PROMPT('Not')
  elem TYPE(*CHAR) LEN(1) RSTD(*YES) VALUES(' ' '(') +
          EXPR(*YES) PROMPT('Open parenthesis')
  elem TYPE(*INT4) RSTD(*NO) EXPR(*YES) PROMPT('Position')
  elem TYPE(*CHAR) LEN(3) RSTD(*YES) DFT(*EQ) +
          VALUES(*EQ *NE *GT *GE *LT *LE *NG *NL *CT) +
          PROMPT('Test')
  elem TYPE(*CHAR) LEN(40) EXPR(*YES) PROMPT('Search value')
  elem TYPE(*CHAR) LEN(1) RSTD(*YES) VALUES(' ' ')') +
          EXPR(*YES) PROMPT('Close parenthesis')
```

There are seven elements in the mixed list. These allow me to enter *AND, *OR, *NOT and parentheses in the proper places. The SELECT parameter specifies MAX(12), which creates an array of mixed lists.

My CL command-processing program converts the data to a record-selection expression, which I pass to Open Query File (OPNQRYF).

```
%SST(SPLFDATA 4 1) *EQ 'A' *AND (%SST(SPLFDATA 16 2) *EQ 'DF' *OR
%SST(SPLFDATA 16 2) *EQ 'TS')
```

The CL program copies the spooled file into a scratch program-described database file, runs OPNQRYF and Copy from Query File (CPYFRMQRYF) to copy selected report lines into a second scratch file, then uses Display Physical File Member (DSPPFM) for the second scratch file to show me only the lines I wish to see.

My CL program receives the SELECT parameter as one long string. It's up to me to properly extract the data values from that string. Here's how that's done.

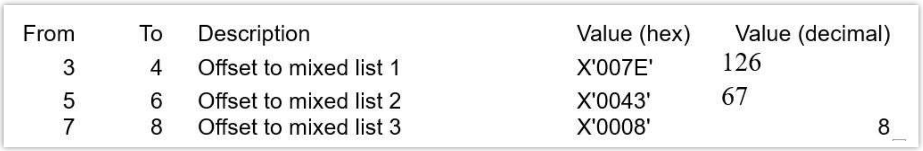
The first two bytes are an integer that tells me the number of mixed lists in the array. I use the %BIN function to retrieve this value.

```
pgm  ( . . . &inSlt)
dcl  &inSlt          *char    1024
dcl  &ListSize       *uint       2
ChgVar  &ListSize    %bin(&inSlt 1 2)
```

Following this is a series of two-byte integers telling where each data structure begins in the parameter string. There is one two-byte integer for every mixed list that was passed. Here is that data in hexadecimal:

```
007E00430008
```

Here's that information as a table.



From	To	Description	Value (hex)	Value (decimal)
3	4	Offset to mixed list 1	X'007E'	126
5	6	Offset to mixed list 2	X'0043'	67
7	8	Offset to mixed list 3	X'0008'	8

These values are offsets from the beginning of the parameter string, so a value of 67, for example, means 67 bytes from the beginning of the parameter string, which is position 68 of the string.

Each mixed list follows the offsets, from the last to the first. Each mixed list contains eight values – the number of values in the list, which is always 7 and can be ignored, followed by the seven fields defined by ELEM in the command definition.

Extracting the mixed list values from the string is messy, but not difficult.

```
dcl &inSlt         *char  1024
dcl &ListSize      *uint     2
dcl &Lx            *uint     4
dcl &Ox            *uint     4
dcl &Quote         *char     1 value('''')
dcl &ValueLen      *uint     4
dcl &SplfData      *char    10 value(SPLFDATA)
dcl &Offset        *uint     4
dcl &Element       *char    59
dcl    &AndOr      *char     4 stg(*defined) defvar(&Element  3)
dcl    &Not        *char     4 stg(*defined) defvar(&Element  7)
dcl    &OpenParen  *char     1 stg(*defined) defvar(&Element 11)
dcl    &Position   *int      4 stg(*defined) defvar(&Element 12)
dcl    &Test       *char     3 stg(*defined) defvar(&Element 16)
dcl    &Value      *char    40 stg(*defined) defvar(&Element 19)
dcl    &CloseParen *char     1 stg(*defined) defvar(&Element 59)
dcl &QrySlt        *char  1024
/* Build the query select expression */
chgvar  &ListSize    %bin(&inSlt 1 2)
dofor &Lx from(1) to(&ListSize)
   chgvar  &Ox       (&Lx * 2 + 1)
   chgvar  &Offset   (%bin(&inSlt &Ox 2) + 1)
   chgvar  &Element   %sst(&inSlt &Offset 59)
   chgvar  &ValueLen (%checkr(' ' &Value))
   if (&Position *ne 0) do
      ChgVar &QrySlt +
          ( &QrySlt *bcat +
                &AndOr *bcat &Not *bcat &OpenParen *cat +
               '%SST(' *cat &SplfData *bcat +
                %char(&Position) *bcat +
                %char(&ValueLen) *tcat ')' *bcat +
                &Test *bcat +
                &Quote *cat %trimr(&Value) *cat &Quote *cat +
                &CloseParen)
   enddo
```

Once the query selection string is built, all that's left is to create a scratch file and display it.

```
OpnQryF   file((&TempLib/&SplfData)) +
             QrySlt(&QrySlt) OpnID(&SplfData)
CpyFrmQryF  FromOpnID(&SplfData) +
              ToFile(&TempLib/&FileToShow) +
              MbrOpt(*REPLACE) CrtFile(*YES)
DspPFM &tempLib/&FileToShow
```

If I run this command:

```
SSF FILE(QSYSPRT) JOB(*) SPLNBR(*LAST) +
      SELECT((*N   *N *N   4 *EQ A) +
             (*AND *N '(' 16 *EQ DF) (*OR *N *N 16 *EQ TS ')'))
```

I see this:

```
*...+....1....+....2....+
 5 A Y       N DF Y 44444
 6 A Y 12.50 Y DF N 44444
 7 A Y 12.50 Y DF N 44444
 8 A Y 12.50 Y DF N 44444
13 A N 12.50 Y DF N 12345
14 A N  6.00 N TS Y 12345
```

```
15 A N  6.00 Y DF Y 44444
23 A Y 12.50 N TS Y 12345
```

You can download my code and play with it or use it.

If you're a fan of data structures in RPG – and who isn't? – you might like to incorporate mixed lists into CL commands. It works for me.

RELATED STORY

CL Command Definition Statements

Tags: Tags: 400guru, CL, ELEM, FHG, Four Hundred Guru, IBM i, RPG, System/34

End Of Support Announced For IBM Power Middleware Releases

Thoroughly Modern: How IBM i Shops Can Navigate The AI Landscape In 2024

LEAVE A REPLY

RECENT POSTS

SUBSCRIBE

PAGES

SEARCH

IBM i Community Shares 2024 Predictions
Thoroughly Modern: How IBM i Shops Can Navigate The AI Landscape In 2024
Guru: Using Mixed Lists To Add “Data Structures” To CL Commands
End Of Support Announced For IBM Power Middleware Releases
IBM i PTF Guide, Volume 26, Number 1
IBM i Chief Architect Will Gives N2i Some Platform Pointers
IBM Patches a Slew of Security Vulns in Db2 Web Query
A Few Power Systems Items At The Cusp Of The New Year
Four Hundred Monitor, January 10
IBM i PTF Guide, Volume 25, Numbers 51, 52, And 53

To get news from IT Jungle sent to your inbox every week, subscribe to our newsletter.

About Us
Contact
Contributors
Four Hundred Monitor
IBM i PTF Guide
Media Kit
Subscribe