167 VM update

Spring 2001

In this issue

VM Update

Published by
Software Diversified Services (SDS)
5155 East River Road
Minneapolis, MN 55421-1025
USA
www.sdsusa.com
sales@sdsusa.com
support@sdsusa.com
voice 763-571-9000
fax 763-572-1721

Editor
Phil Norcross
vu-ed@sdsusa.com
763-571-9000

Editorial Panel
Chuck Meyer, Chuck Meyer Systems, Inc., USA.

File formats
VM Update is published in pdf format, to be read with an Adobe® Acrobat® Reader. The Reader is available free of charge at www.adobe.com. Once the Reader is installed, Netscape and Microsoft browsers can display pdf files in browser windows.

Most of the code described in articles is also available in text or other formats that readers can readily copy to their VM machines.

Free subscription, back issues
VM Update is free of charge at www.sdsusa.com. At that site, SDS provides back issues through January 1997. Parts of older issues are available at www.xephon. com/archives/vmi.htm.

Contributions
SDS and *VM Update* welcome contributions. See "Contributing Articles" at www.sdsusa.com/vmupdate/vutoauthors.htm

Disclaimer
Readers are cautioned that, although the information in this journal is presented in good faith, neither SDS nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither SDS nor the contributing organizations or individuals accept any liability of any kind whatsoever arising out of the use of such material.

Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, EXECs, and other contents of this journal before making any use of it.

Reactions to the SPACER REXX routine

SPACER REXX, the routine presented in the last issue of *VM Update* (Winter 2001), provides great help for making readable lists of files: It inserts a blank line between files of different types. With this article, I want to present a similar tool for XEDIT sessions, and I want to add to your knowledge of pipelines by offering a performance improvement to SPACER REXX.

Filetype separator for XEDIT

SPACER REXX is to be used in a pipeline:

```
PIPE COMMAND LISTFILE * * S | SORT 10.8 | SPACER 10.8 | > SDISK FILES
```

SEPAR XEDIT is for an XEDIT session:

```
FILELIST * * S # stype # SEPAR 10 17
```

Both examples give you a list of files sorted by filetype, and when the filetype changes, a blank line is inserted.

I wrote SEPAR XEDIT for listings like the following: The customer I work for has 19 VM systems, and using a service machine, we get lists of files installed on each system. To list all files with filename VMPRF on VMPRF's 191 minidisk, for example, we enter

```
GEEFME VMPRF * VMPRF 191.
```

The result looks like figure 1 (next page). SEPAR XEDIT improves readability by inserting blank lines between files of different types, as in figure 2.

SEPAR XEDIT is listed at the end of this article, and available for downloading from *VM Update* as SEPAR.VMARC.

figure 1: without SEPAR XEDIT

```
 GEEFME   RESULTS  A1  V 80  Trunc=80 Size=154 Line=0 Col=2 Alt=1
====>
<!..+....1....+....2....+....3....+....4....+....5....+....6....+....7...
* * * Top of File * * *                                                  00000
--------- FILENAME FILETYPE FM F LRECL     RECS  BLKS   DATE       TIME    00001
VMKBTW01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:11  00002
VMKBSI02: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:10  00003
VMKBSH01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:09  00004
VMKBHK01: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:07  00005
VMKBIC01: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:07  00006
VMKBBR10: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00007
VMKBME01: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00008
VMKB2000: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00009
VMKBFF01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:01  00010
VMKBFF01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00011
VMKBLO01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00012
VMKBMN01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00013
VMKBSH01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00014
VMKBTW01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00015
 1=scr  2=X  3=qt 4=spjn  5=ctxt  6=?  7=b 8=f  9=vh  10=home 11=save   12=file
```

figure 2: with SEPAR XEDIT

```
 GEEFME   RESULTS  A1  V 80  Trunc=80 Size=167 Line=0 Col=19 Alt=0
====>
13 separator lines inserted
DMSXSU510I AUTOSAVED as 100001 AUTOSAVE A1
* * * Top of File * * *                                                  00000
--------- FILENAME FILETYPE FM F LRECL     RECS  BLKS   DATE       TIME    00001
                                                                          00002
VMKBTW01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:11  00003
VMKBSI02: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:10  00004
VMKBSH01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:09  00005
VMKBFR01: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:04  00006
VMKBME03: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:04  00007
VMKBBR10: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00008
VMKBME01: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00009
VMKB2000: VMPRF    MASTER   Z1 V    80       20     1 2000-03-07  1:40:03  00010
VMKBFF01: VMPRF    MASTER   Z1 V    60       20     1 2000-03-07  1:40:01  00011
                                                                          00012
VMKBBR01: VMPRF    MASTERX  Z1 V    80       20     1 1999-11-26 14:56:45  00013
VMKBFF01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00014
VMKBLO01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00015
VMKBMN01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00016
VMKBSH01: VMPRF    MASTERX  Z1 V    60       20     1 1995-10-12 11:51:12  00017
 1=scr  2=X  3=qt 4=spjn  5=ctxt  6=? 7=b  8=f  9=vh  10=home 11=save  12=file
```

Improving performance of SPACER REXX

I have a curious nature, and I care—maybe too much—for performance. I had a look at the SPACER REXX procedure. And I saw room for performance.

A bit simplified, the heart of SPACER REXX is as follows:

```
signal on error /* Jump to "error:" when at end-of-file */
'PEEKTO RECORD' /* Get the first record */

Do num_input=1
  If ?wdnum
    Then field = WORD( record,c1)
    Else field = SUBSTR(record,c1,c2-c1)
  If (num_input=1) Then field_sav = field
  If \(field_sav = field) Then Do
       field_sav = field
       'OUTPUT' spacer_rec    /* write out the SPACER record */
       End
  'OUTPUT' record                    /* write out the record */
  'READTO'        /* consume the previously-PEEKTO-ed record */
  'PEEKTO RECORD'         /* and then expose the next record */
  End
Error: Exit rc*(rc\=12)
```

In the code above, each and every line is handled by the Rexx code. A general assumption is that replacing Rexx DO loops by PIPE stages improves performance. For the above "problem" we can use what Melinda Varian calls a "sipping Pipeline," in which records are processed in groups. In general, the process looks like this:

```
   do forever
      'PEEKTO RECORD'                          /* Get a record */
      ... analyze the record to prepare handling a group
      'CALLPIPE *.input:',              /* read input records */
1)       '| WHILExxxx....'      /* Process a group of records */
         '| ...handle/rearrange records ',
         '|*.output:'               /* pass records to output */
   End
```

The key to this technique is the pipe stage marked "1)" above. It must be a stage that stops reading more records when a new group begins. Stages often used for this function are WhileLabel, ToLabel, ToTarget, or even TAKE nnn. The first two can only be used when a group is delimited by data in column 1 of the record. ToTarget is more general.

For our "insert separator" problem, however, we'd like a WhileTarget state. If it existed, we could code the following:

```
   signal on error           /* Jump to "error:" when at end-of-file */
   If ?wdnum                             /* What defines a "group" ? */
      Then Picfield = 'W'c1                        /* Word selection */
      Else Picfield = c1'-'c2-1                  /* Column selection */

   'PEEKTO RECORD'                          /* Get the first record */
   Do num_input=1
    If ?wdnum
      Then field = WORD( record,c1)
      Else field = SUBSTR(record,c1,c2-c1)
    If (num_input=1) Then field_sav = field
    If \(field_sav = field) Then Do
         field_sav = field
         'OUTPUT' spacer_rec          /* write out the SPACER record */
         End
    'CALLPIPE',
       '*:',                                    /* Get input records */
1)     '|WHILETARGET LOCATE' PicField '/'field'/',  /* Handle a group */
       '|*:'                                 /* Send group to output */
     'PEEKTO RECORD'              /* and then expose the next record */
     End
  Error: Exit rc*(rc\=12)
```

There are a few problems with line 1) in the above solution:

- There is no WhileTarget stage; we have to use something else. What we use is explained below.
- The LOCATE stage isn't safe when used with a word as search scope. Suppose you use

  ```
  PIPE COMMAND LISTFILE * * S | SORT W2 | SPACER W2 | > SDISK FILES A
  ```

  and the spacer code has "EXEC" stored in Rexx variable "field." `LOCATE W2 /EXEC/` will let filetypes `EXECOLD`, or `EXEC--` throug; string "EXEC" will be found in the second word. A waterproof solution is to use the PICK stage: `PICK W2 == /EXEC/`
- How the above code specifies the string to search isn't safe either: when Rexx variable "field" contains a slash or vertical bar (/ or |), the syntax of the LOCATE stage is wrong. Suppose "field" contains "xyz | klm" and "pickfield" contains 10-25. When Rexx resolves the statement, the following command is passed to CMS Pipelines:

  ```
  CALLPIPE *: |WHILETARGET LOCATE 10-25 /xyz|klm/ |*:
  ```

> The Pipeline scanner splits the arguments at the | character, so the Locate stage will complain for a missing string delimiter as it gets "10-25 /xyz" as parameters.

There is an easy solution. Use Rexx's C2X function to convert the string to hexadecimal notation and pass this hex string to LOCATE. So we code:

```
...LOCATE' picField 'X'c2x(field)
```

Two general warnings must be remembered:

WARNING 1: Often in your code you want to select lines in which a given word is equal to the search string. Using `LOCATE WORD n /xxx/` is dangerous then, not only in sipping pipes, but everywhere. You must use `PICK WORD n == /xxx/`

WARNING 2: To solve delimiter problems, exploit Pipe's hexadecimal string notation. For example:

```
'LOCATE X'c2x(string)
'PICK 1.10 == X'c2x(string)
```

or

```
'StrWhileLabel X'c2x(string)
```

So, instead of

```
'|WHILETARGET LOCATE' PicField '/'field'/',
```

we code

```
'|WHILETARGET PICK' PicField '== X'c2x(field),
```

Now we are ready to find an alternative to the nonexistant WhileTarget stage. What can we use? The stage ToTarget comes to mind. The argument of ToTarget is another Pipeline stage. ToTarget passes records down the pipeline until the argument stage produces a record.

The solution would be easy if we knew the string that followed the group we're about to handle:

```
'|ToTARGET PICK' PicField '== X'c2x(nextField),
```

But we don't know what string ends this group. The code that works is this:

```
'|ToTARGET PICK' PicField '\== X'c2x(field),
```

That solution may look a bit strange: We code a "not equal field" to handle the group containing our field. But carefully analyze the code; you'll understand that it does what we want.

NOTE: Do not abandon sipping pipelines because we need a confusing stage here. You can learn a lot more about sipping pipelines by reading Varian's paper: Go to http://pucc.princeton.edu/~pipeline/ and search for "Cramming for the Journeyman Plumber Exam; Part III: Dynamic Reconfiguration in CMS Pipeline."

There is one final issue to solve. In a Pipeline not only the input "file" can be at end-of-file, the output "file" can be there too.

Suppose the user only wants to get 20 lines and inserts a TAKE in the pipeline:

```
PIPE COMMAND LISTFILE * * S | SORT W2 | SPACER W2 | take 20 | > SDISK FILES A
```

This TAKE stage has consequences for our SPACER REXX. When SPACER REXX has produced 20 records, the TAKE ends and SPACER's output is at "end-of-file." In the original solution we basically had:

```
1)  signal on error           /* Jump to "error:" when at end-of-file */
2)  'PEEKTO RECORD'                          /* Get the first record */

    Do num_input=1
      ...
3)    if ... then 'OUTPUT' record   /* write out the separator record */
4)    'OUTPUT' record                        /* write out the record */
      'READTO'             /* consume the previously-PEEKTO-ed record */
5)    'PEEKTO RECORD'              /* and then expose the next record */
      End
   Error: Exit rc*(rc\=12)
```

At line 1, we tell Rexx to jump to "error:" whenever a non-zero return code is presented. Lines 2, 3, 4 and 5 can present a non-zero return code. Return code 12 on READTO, PEEKTO or OUTPUT means end-of-file. So our Rexx exec will not only end when there is no more input (line 2 or 5), but also when no more output is wanted (line 3 or 4).

The improved code basically looks like this:

```
1)  signal on error          /* Jump to "error:" when at end-of-file */
2)  'PEEKTO RECORD'                          /* Get the first record */

    Do num_input=1
    ...
3)  if ... then 'OUTPUT' record    /* write out the separator record */
4)    'CALLPIPE ...'                 /* write out a group of records */
5)    'PEEKTO RECORD'              /* and then expose the next record */
      End
   Error: Exit rc*(rc\=12)
```

We no longer use OUTPUT to pass the original records through. The following CALLPIPE is used instead:

```
   'CALLPIPE',
      '*:', /* Get input records */
      '|ToTARGET PICK' PicField '\== X'c2x(field), /* Handle a group */
      '|*:' /* pass to Output */
```

Three conditions can cause this CALLPIPE to end:

1. There are no more input records.
2. PICK found a record not containing the field, consequently ToTarget stops passing records through.
3. No more output is wanted.

In each case, CALLPIPE's return code is zero. How can the DO loop be ended then?

- There is no problem detecting end-of-file on input. The PEEKTO at line 5 will present a return code of 12.
- Only the OUTPUT at line 3 can detect end-of-file on output. But this statement is only executed when a new group starts. In our example, when the TAKE 20 stops taking records in the middle of a group, the DO loop indeed becomes a real DO FOREVER.

The problem can be solved by two means.

- The oldest solution is to explicitly test if the output stream is at end-of-file by using the STREAMSTATE command. STREAMSTATE gives a non-zero return code at end-of-file.

```
Signal on error
'PEEKTO RECORD'                          /* Get the first record */
Do num_input=1
 ...
 if ... then 'OUTPUT' record  /* write out the separator record  */
 'CALLPIPE ...'               /* write out a group of records    */
 'STREAMSTATE OUTPUT'         /* is more output wanted ?         */
 'PEEKTO RECORD'              /* and then expose the next record */
 End
Error: Exit rc*(rc\=12)
```

- A more modern solution is to use the EOFREPORT command. With EOFREPORT ALL the PEEKTO command will end with return code 8 when all output streams are at end-of-file. The heart of our solution now looks like:

```
Signal on error
'EOFREPORT ALL'        /* Stop also when no more output is wanted */
'PEEKTO RECORD'                           /* Get the first record */
Do num_input=1
 ...
   if ... then 'OUTPUT' record  /* write out the separator record */
   'CALLPIPE ...'                 /* write out a group of records */
   'PEEKTO RECORD'             /* and then expose the next record */
 End
Error: Exit rc*(rc\=12 & rc\=8)
```

The complete new code, SPACER2 REXX, is listed below and is available to download from the *VM Update* website as SPACER2.VMARC.

Comparing performance of SPACER and SPACER2

As mentioned earlier, the main reason to use CALLPIPE, is to improve performance. So I measured both SPACER procedures. The most elegant way to measure Pipelines is to use the RITA module, which can be found on MAINT 193, and is explained by Varian: go to http://pucc.princeton.edu/~pipeline/ and search for "Streamlining your Pipelines."

To use RITA, start the pipeline by entering "RITA . . ." instead of "PIPE . . ."

```
RITA COMMAND LISTFILE * * C | SORT 10.8 | SPACER 10.8 | > CDISK FILES A
RITA COMMAND LISTFILE * * C | SORT 10.8 | SPACER2 10.8 | > CDISK2 FILES A
```

Great was my surprise, or rather, my disappointment. The new pipe consumed 20% more CPU than the original one. What could be the explanation of this unexpected result?

To find the cause, I ran the same pipe, but against the S-disk. Now SPACER2 was clearly the winner—almost twice as fast. This table illustrates the results.

table 1: performance results

	Pipe with SPACER REXX	Pipe with SPACER2 REXX	Files	Separators inserted
C-disk	16.962 ms CPU used	20.484 ms CPU used	178	29
S-disk	59.387 ms CPU used	31.583 ms CPU used	700	26

To explain the difference, let's take two extreme examples, both with 100 lines of data to handle.

At on extreme, suppose all the files on the minidisk are of the same filetype:

- With SPACER REXX, the Rexx code has to inspect every line, one by one. The DO loop is executed 100 times.

- With SPACER2 REXX, the first line is handled by Rexx, then the CALLPIPE reads all the lines at once. Few Rexx lines must be interpreted. The DO loop is executed one time.

Now let's take the other extreme: all the files on the minidisk have a different filetype.

- With SPACER REXX, the Rexx code has to inspect every line, one by one. The DO loop is executed 100 times.
- With SPACER2 REXX, the CALLPIPE will only let one line through. The DO loop is executed 100 times.

The test results show that the CALLPIPE used by SPACER2 costs more than the OUTPUT and READTO used by SPACER. And that's not a surprise. The CALLPIPE arguments must first be analyzed and then stages must be initialized, a rather complex process.

But when the new construction can handle many records, the initial overhead is payed back.

Conclusion

I hope I have illustrated how CALLPIPE can be used and that it can perform better. On the other hand, CALLPIPE isn't faster in all cases.

Do not forget that the SPACER procedure does not have to do complex things with the records it passes through. It only looks at them; it doesn't change them at all. When writing a procedure that touches the records flowing through, you can gain performance by replacing Rexx code by stages included in the CALLPIPE.

SPACER2 REXX

```
/*****************************SPACER.REXX*********************************/
/* Add a "spacer" record when a specified field changes value         */
/*                                                                    */
/* SPACER field-spec <fill-character>                                 */
/*                                                                    */
/* For example --                                                     */
/* 'PIPE COMMAND LISTFILE * * A (DATE NOH' ,                          */
/* '! SORT 10.8 1.19' ,                                               */
/* '! SPACER 10.3' ,                                                  */
/* '! CONSOLE'                                                        */
/*                                                                    */
/* will sort LISTFILE output on FileType (10.8), and then add         */
/* a blank record between lines where there is a change in the        */
/* first 3 characters of FileType.                                    */
/* The added record will be as long as the right-most column          */
/* being compared (13, in the preceding example).                     */
/* If the stage had read "SPACER 10.3 -" or "SPACER 10.3 60"          */
/* then each inserted record has dashes (X'60') in the field.         */
/* "SPACER W2" (in this case) creates same results as "SPACER 10.8".  */
/*                                                                    */
/* 1996/11/15 CMSi/CHM Written by Chuck Meyer                         */
/* 1996/11/21 CMSi/CHM Allow "Wnnn" as operand (word number)          */
/* 18 Mar 2001: Kris Buelens: use CALLPIPE instead of OUTPUT & READTO */
/*                                                                    */
/**********************************************************************/
copyright = 'Copyright: Chuck Meyer Systems, Inc.; 1996'
version = '1996.11.21'

Signal ON ERROR

Parse Upper Source . . fn1 ft1 fm1 fn2 . 1 _source_
Parse Upper Arg c01 fil .       /* possibly 2 words of input  */
Parse Var c01 c11 '-' c12       /* maybe it's FROM-TO         */
Parse Var c01 c21 '.' c22       /* maybe it's FROM.LENGTH     */
Parse Var c01 'W' c31           /* maybe it's WORD number     */

?wdnum = DATATYPE(c31,'W')

Select /* determine the column-range to be compared */
  When DATATYPE(c11,'W') & DATATYPE(c12,'W') Then ss = c11 (c12 + 1)
  When DATATYPE(c21,'W') & DATATYPE(c22,'W') Then ss = c21 (c22 +c21)
  When DATATYPE(c01,'W')                     Then ss = c01 (c01 + 9)
  When ?wdnum                                Then ss = c31 '0'
  Otherwise                                       ss = 1 (10 + 1)
  End
Parse Var ss c1 c2 .

Select /* determine what the fill-record should look like */
  When (LENGTH(fil)=2) & DATATYPE(fil,'X') Then fx = X2C( fil)
  Otherwise                                     fx = LEFT(fil,1,' ')
  End
num_added = 0
num_input = 0

'EOFREPORT ALL' /* Make PEEKTO stop when no more input or output */
```

```
'PEEKTO RECORD'

If ?wdnum
    Then Picfield = 'W'c1
    Else Picfield = c1'-'c2-1
Do num_input=1
  If ?wdnum
    Then field = WORD( record,c1)
    Else field = SUBSTR(record,c1,c2-c1)
  If (num_input=1) Then Do
    field_sav = field
    x = COPIES(fx,LENGTH(field))
    If ?wdnum
      Then spacer_rec = COPIES(' ',WORDINDEX(record,c1)-1)    !! x
      Else spacer_rec = COPIES(' ',c1-1)                      !! x
    Drop x
    End
  If field_sav <> field Then Do
       field_sav = field
       num_added = num_added + 1
       'OUTPUT' spacer_rec     /* write out the SPACER record     */
       End
'CALLPIPE (NAME SPCR)',
   '*:',
   '!TOTARGET PICK' PicField '\== X'c2x(field),
   '!*:'
 'PEEKTO RECORD'               /* and then expose the next record */
 End

ERROR:
  /* RC 12= eof on input; rc=8= eof on output */
  rcx = rc*(rc<>12 & rc<>8)
         /* If called from a REXX prog, pass-back some numbers    */
  Signal OFF ERROR
  Address COMMAND 'PIPE (SEP ? )' ,
         'COMMAND PIPE (NOMSG 15)' ,
         'LITERAL' num_input num_added '! VAR' fn1 '1 ! HOLE'
  If (rc<>0) Then Say 'PIPE stage' fn1 'read' num_input 'records,' ,
              'inserted' num_added 'records ('num_input+num_added rc')'
  Exit rcx

/*====================================================================*/
```

SEPAR XEDIT

```
/* This XEDIT macro will insert a separator line whenever
   the content of part of the existing records changes.
    +-------------------------------------------------------------+
    ! format: ! SEPAR col1 col2 <separator_line> !
    +-------------------------------------------------------------+
 Example:
 -File before: VMSRES MAINT 191
               VMSRES MAINT 190
               VMSRES MAINT 490
               VMPK01 KRIS 191
               VMPK01 GUY 191
               VMPK02 MAINT 194
               VMPK02 KRIS 192
 -Enter "SEPAR 1 6 ==================" in XEDIT's commandline
 -File after:  VMSRES MAINT 191
               VMSRES MAINT 190
               VMSRES MAINT 490
               ==================
               VMPK01 KRIS 191
               VMPK01 GUY 191
               ==================
               VMPK02 MAINT 194
               VMPK02 KRIS 192

 Written by: Kris Buelens IBM Belgium; KRIS at VMKBBR01 1 Sep 1998*/

parse upper source . . myname mytype . syn .
c ='COMMAND'; cs=c 'SET'; ce=c 'EXTRACT'
parse upper arg c1 c2 separ
if c2='' then call Errexit 5,'Column range missing/incomplete'

if \datatype(c1,'W') then call ErrExit 5,'Invalid column:' c1
if \datatype(c2,'W') then call ErrExit 5,'Invalid column:' c2

c 'PRESERVE'
ce '/LINE/TOF/ALT'
cs 'Z' c1 c2
if rc<>0 then call ErrExit 5 1,'Invalid column:' c2

cs 'WRAP OFF';cs 'STAY OFF';cs 'LINEND OFF';cs 'IMAGE OFF';cs 'MSGM OFF'
cs 'MASK IMM' separ
if tof.1='ON' then c '+1'
allsep=xrange('00'x,'3F'x)xrange('41'x,'FF'x)
new=0
do new=0 by 1
   ce '/CURLINE'
   targ=substr(curline.3,c1,c2-c1+1)
   sep=verify(allsep,targ)
   if sep=0 then call ErrExit 5 1,'No separator char can be found for',
        'string' targ
   sep=substr(allsep,sep,1)
   c 'LOCATE \'sep!!targ!!sep
   if rc<>0 then Call Exit 0 1
   c '-1' c 'ADD'
   c '+2'
end
```

```
exit:
ERREXIT: /* general errorexit routine */
 parse upper source . . myname mytype . syn .
 parse arg erc rest
 if rest=1 then c ':'line.1 c 'RESTORE'
 if symbol('new')='VAR' then do
    c 'MSG' new 'separator line'left('s',new<>1) 'inserted'
    if new>0 then cs 'ALT' alt.1+1
 end
 do i=2 to arg() /* give errormessages (if any) */
    c 'EMSG' myname':' arg(i)
 end
 exit erc
```

Kris Buelens, Advisory Systems Engineer
IBM Belgium
dris_buelens@be.ibm.com

Contributing Articles and Code to *VM Update*

We welcome your inquiries and manuscripts. Please send them to vu-ed@sdsusa.com

VM Update magazine has changed hands. Its new publisher, Software Diversified Services, strives to continue *VM Update*'s service to the VM community. To that end, SDS invites, welcomes, and encourages your contributions to *VM Update*.

The scope of the magazine has not changed. *VM Update* welcomes your description and code of use to technicians, programmers, and managers running VM systems. Your articles might share modifications, fixes, utilities, short-cuts, hints, guidelines, evaluations, and advice.

Authoring needn't be difficult. The typical article copies code the author wrote for some other purpose and adds a few pages of explanatory text that *VM Update*'s editor will happily polish for you.

In return for the license to publish your article at its website, *VM Update* will pay you $125 per article. We understand that is less than Xephon used to pay. But remember we now distribute *VM Update* free of charge.

Along with that payment you also get worldwide attention via the Web, you get a contribution to your reputation and your employer's reputation, and you get to help your colleagues in the business of running VM. Last year, *VM Update* printed on paper went to about 150 VM professionals each month. By publishing free of charge on the Web, we expect to bring you more readers than that.

So please join SDS in the care and feeding of *VM Update*, a valuable resource in the worldwide community of VM professionals.

Please send your ideas and manuscripts to vu-ed@sdsusa.com. Further details are available at www.sdsusa.com/vmupdate/vutoauthors.htm. Thank you.