September 1999

In this issue

3  Adding numeric values in columns
5  Writing reliable and secure procedures
21 A full screen console interface – part 14
38 REXX/CMS talks to VB over TCP/IP
47 The DG digest reader
52 VM news

© Xephon plc 1999
Editor
Robert Burgess

Disclaimer
Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon 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 Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever 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.

VM Update on-line
Code from VM Update can be downloaded from our Web site at http://www.xephon.com/vmupdate.html; you will need the userid shown on your address label.

Contributions
Articles published in VM Update are paid for at the rate of £170 ($250) per 1000 words for original material. To find out more about contributing an article, without any obligation, please contact us at any of the addresses above and we will send you a copy of our Notes for Contributors.
Adding numeric values in columns

ADDCOL is an XEDIT macro to add numeric values displayed in columns. You just need to specify a pair of columns and, optionally, the beginning and ending line (by default, the entire file will be considered). An example is shown in Figure 1 in which you want to sum the values between columns 42 and 47, from line 140 to line 151. The result is displayed as a message. Since there are lines that contain non-numeric values in the specified range, those lines are ignored, but a warning message is issued. A quick on-line help is provided simply by typing ‘ADDCOL’ or ‘ADDCOL ?’.

```
UNITS31  LISTING B1  F  133  Trunc=ØØ  Size=435  Line=136  Col=1  Alt=Ø
Total: 79662  Non-numeric values found and ignored
[...+....1....+....2....+....3....+....4....+....5....+....6....+....]
Ø0136    000843    000927    087477    012117    0103731
Ø0137    000666    000111    101783    026331    0110884
Ø0138    000667    000119    102405    026949    0111195
Ø0139    000668    000127    103027    027567    0111506
Ø0140    000665    000103    101161    025713    0110573
Ø0141    000037    000879    083745    008409    0101865
Ø0142
Ø0143 1 ==================================================================
Ø0144 Ø
00145    000038    000887    084367    009027    0102176
00146    000039    000895    084989    009645    0102487
00147    000040    000903    085611    010263    0102798
00148    000069    000135    103649    006185    0156436
00149    000069    000335    006849    003115    0023427
00150    000021    000165    003243    002122    0224817
00151    000044    000235    001639    005183    0155000
00152    000059    000164    442400    008545    0112455
00153    000070    000423    111271    001222    0116458
====> addcol 42 47 140 151
```

**Figure 1: Example display**

**SOURCE CODE**

```c
/*-----------------------------------------*/
/* ADDCOL macro for XEDIT */
/* Format: ADDCOL Col1 Col2 Line1 Line2 */
/* This macro adds numeric values found between the specified */
```
/*   lines and columns.                                             */
/*==================================================================*/
arg c1 c2 l1 l2 .
if c1 = "?" | c1 = "HELP" | c1 = "" then signal helpe
cmd = "command"
cmd "extract/line/size/msgmode/"
cmd "set msgmode on"
if l1="" then l1 = 1
if l2="" then l2 = size.1
if ~(datatype(c1,"W")&(datatype(c2,"W"))) then do
  msg "ERROR - Invalid columns specified"
  signal saida
end
if ~(datatype(l1,"W")&(datatype(l2,"W"))) then do
  msg "ERROR - Invalid lines specified"
  signal saida
end
if c2 < c1 then do
  msg "ERROR - Column2 is less than Column1"
  signal saida
end
if l2 < l1 then do
  msg "ERROR - Line2 is less than Line1"
  signal saida
end
len = c2-c1+1
total = Ø
do j = l1 to l2
  ":" j
  cmd "extract/curline/"
  number = substr(curline.3,c1,len)
  if datatype(number,"W") then total = total+number
  else non_numeric = 1
end
":" line.1
if non_numeric = 1 then msgx=" Non-numeric values found and ignored"
else msgx=""
msg "Total: "total msgx
saida:
  cmd "set msgmode" msgmode.1
exit
helpe:
say " Format:"
say " ADDCOL FirstColumn SecondColumn FirstLine LastLine"
say " FirstLine is optional and defaults to top-of-file"
say " LastLine is optional and defaults to end-of-file"
exit

Luis Paulo Figueiredo Sousa Ribeiro
Systems Engineer
Edinfor (Portugal) © Xephon 1999
Writing reliable and secure procedures

It is obviously important that tools perform well, are reliable, and are secure. We have often seen REXX procedures without an address COMMAND statement, or CMS Pipelines using the CMS device driver stage. In both cases, performance is wasted and the procedures are subject to breaking in an unpredictable manner.

In this article, we would like to explain the consequences of not using address COMMAND or using the CMS stage. In order to understand all the implications, we will firstly review how CMS resolves commands, the so-called CMS search order.

CMS COMMAND SEARCH ORDER

CMS commands (procedures or object modules) can be found on CMS mini-disks or in storage. Let’s first review how commands, and REXX procedures in particular, can reside in storage.

An EXEC can be loaded in the users’ private storage by issuing the command:

```
EXECLOAD fn [ft [fm [execname [exectype]]]]
```

The advantage of this is that the file does not need to be loaded from disk each time the procedure is invoked (resulting in reduced I/O). A logical consequence is that CMS will always use the copy in storage (if it exists) instead of the one residing on a mini-disk.

Since CMS Release 5, the so-called ‘installation segment’ (or INSTSEG), permits the systems programmer to load frequently-used procedures in a shared segment. The default name of the shared segment is CMSINST. Typically, you’ll find procedures like FILELIST and RDRLIST (with their accompanying XEDIT macros) in this segment.

In this case, not only the load from disk is avoided, but also all users on the system use the same copy in storage, because it is a shared segment (resulting in reduced I/O and paging). But, in order to allow a user to execute a private (enhanced?) version of a procedure from its own mini-disks or from its private storage (EXECLOADed), it should
take precedence over the INSTSEG resident version.

A filemode is therefore associated with the INSTSEG, to let it be in the filemode search order. The default filemode is S (this can be changed via the ‘SET INSTSEG ON mode’ command).

So, with the INSTSEG defaulting to S, CMS uses the following search order when searching for an EXEC:

1. Is the procedure EXECLOADed?
2. Is it on any of the accessed mini-disks with modes A to R?
3. Is it defined in the INSTSEG?
4. Is it on any of the accessed mini-disks with modes S to Z?

CMS Release 6 introduced ‘logical shared segments’. These shared segments can contain modules, EXEs, text decks, and even any other associated object, such as data files. The user can get access to the contents of the shared segment by issuing the ‘SEGMENT LOAD segname’ command.

Once this command is issued, MODULEs contained in the segment are considered to be NUCXLOADed. For procedures (EXEC, XEDIT, etc), it depends on what is specified when the segment is built. You can choose to make the EXEC a logical part of the INSTSEG, or to consider it as EXECLOADed. Of course, this choice influences the search order, as we have just seen.

Thus, when you enter a command in the CMS environment, CMS uses the following search order to locate the command for execution:

1. Search for an EXEC with the specified name using the order as explained above.
2. Search for a translation or synonym of the specified name. If found, search for an EXEC with the valid translation or synonym by repeating step one.
3. Search for a module with the specified command name:
   - Search for a nucleus extension module (NUCXLOADed or part of a logical segment).
– Search for a module in the transient area.
– Search for a nucleus-resident module.
– Search the table of active (open) files for a file with the specified command name and a filetype of MODULE. If more than one open file is found, the one opened first is used.
– Search for a file with the specified command name and a filetype of MODULE on any currently accessed disk or SFS directory, using the standard CMS search order (A to Z).

4 Search for a translation or synonym of the specified command name. If found, search for a module with the valid translation or synonym by repeating step three.

If the command is not known to CMS – that is, all of the above fails – it is passed to CP, provided IMPCP has not been set to OFF. If CP is also unable to recognize the command, you get an ‘Unknown CP/CMS command’ and a return code of ‘-3’.

For example, if you enter the command ‘X sauces cookbook’, CMS would search as follows:

1 Firstly, CMS searches for X EXEC. Here, assume that an X EXEC is not found.

2 CMS then searches the translation and synonym tables and finds that X is a synonym for XEDIT. Then, CMS repeats step one, searching for XEDIT EXEC. Again, assume that an XEDIT EXEC is not found.

3 Next, CMS searches for a CMS command with the name X. It is not found.

4 CMS again searches the translation and synonym tables and finds that it is a synonym for XEDIT. Then, CMS repeats step three, searching for XEDIT. The XEDIT command is found and processed. As a result, the file will be edited.

For more information on the CMS command search order, see the VM/ESA: CMS Command Reference. For additional information on searching for a translation or synonym, see the SET TRANSLATE or SYNONYM commands in the VM/ESA: CMS Application Development Guide.
As you can see, the search process can be quite elaborate. In your daily work you probably don’t realize what really happens in your system!

However, if you are responsible for developing programs or procedures that are to be used by a large number of users, you must be concerned with such aspects as performance, reliability, and security.

This is where the ADDRESS statement of REXX, and the CMS and COMMAND device drivers of CMS Pipelines come into play – we have to make a distinction between different environments where procedures are used.

For example, it makes a difference if a procedure is called from the CMS environment (by this we mean that the command is addressed to CMS directly). Most of the time it will be from the ‘Ready;’ prompt. However, it can also be from the XEDIT command line, provided the command is prefixed with the keyword CMS; in which case we call it a CMS EXEC procedure, or from the XEDIT command line, in which case we probably deal with an XEDIT macro whose filetype is XEDIT.

THE CMS EXECS

As just stated, these procedures are executed from the CMS environment. Their filetype is EXEC. Let’s discuss how commands inside those EXECs are handled and searched for.

Default situation

If no ADDRESS statement is coded in these REXX procedures, the implied default addressing is ‘address CMS’. This means that any CP or CMS command encountered in the procedure is passed to CMS as if it were entered directly at the terminal (or from the CMS environment).

The consequences are as follows:

- The statements can be coded in mixed case. CMS will translate them internally to upper case before execution (just as you can enter commands in mixed case at the terminal).
- The full command resolution, described earlier, applies.
From this, we can conclude that:

- The time to launch the commands will be longer, because of the long command resolution of CMS (leading to performance loss).
- You are never certain which command actually gets executed. Indeed, if there is a procedure, on any of your accessed mini-disks, that has the same name as the CMS command you actually expected to execute, the EXEC will take precedence because of the search order of CMS. The procedure is thus not foolproof!

For example, you want to execute a RENAME command in your procedure, but somehow you have access to a RENAME EXEC, which does an ERASE... This looks a silly example, but is totally feasible. In fact, this is just one of the possibilities you have if you want to create a virus on your VM system. Be prepared to have good back-ups! The result is that your procedure will execute the RENAME procedure, actually resulting in an ERASE!

System personnel frequently hear users complaining, saying things like ‘my procedure worked OK yesterday, but now it doesn’t work anymore, what have YOU changed in the system?’ The answer is that yesterday the user had not accessed your tools disk containing the RENAME EXEC, whereas today he has got the access.

Yet another problem that you may encounter is, for example, that you can no longer use the CP LINK command with a password in the procedure. This happens if, at system generation, an option was set by which passwords are refused as parameters to the command.

**Using ADDRESS COMMAND**

If you use the statement ‘address command’ in your REXX procedure, then things change completely:

- The commands passed to CMS or CP must be coded using upper case (as neither CMS nor CP will translate them for you anymore, and they only understand upper case commands). Note that we mention commands. The parameters can in some cases be passed using mixed-case characters. Commands such as EXECIO and PIPE use upper case parts of the parameters internally to recognize keywords, but leave other parameters unchanged.
• If an EXEC procedure is to be called, the procedure name has to be preceded by the keyword ‘EXEC’.

• Likewise, if a CP command has to be executed, it has to be preceded by the keyword ‘CP’ (do you remember those EXEC1 and EXEC2 days?).

Hence, with the address command you lose something (stricter coding rules) but you gain a lot:

• The performance of the procedure is augmented. Indeed, CMS will limit its search order to steps three and four. On the other hand, as a CP command needs to be prefixed by the CP keyword, it’s clear to CMS that it can pass the command directly to CP and its execution is not influenced by the IMPCP setting. CP is indeed an ordinary CMS command whose sole purpose is to call a CP service.

• An even more important advantage is that your procedure now becomes foolproof. Indeed, because you need to be explicit about whether you want to execute an EXEC, a CMS command, or a CP command, it’s impossible to fall into the same trap as described earlier with the RENAME EXEC.

• Finally, because most commands know whether they are called by another program and not from the terminal, they will not display error messages (eg ERASE will not display ‘File not found’), but will return an appropriate return code. In that case, the use of SET CMSTYPE OFF/ON is no longer required.

• At the same time, our problem with the LINK passwords is solved too. The password can now be included on the LINK command.

One last remark here. Only very recently we discovered that a user-defined synonym could still interfere with our procedure. Indeed, if you look back at the CMS search order, you’ll see that CMS also looks for synonyms or translated commands in step four.

Now, suppose the user has defined LIST to be a synonym of FLIST, with possible abbreviation to one character, while on the other hand, we in our procedure have coded something like:

```
Address command
'L * * F (STACK'
```
then, because of the CMS search order, the user’s synonym gets executed and thus results in the FLIST command that does not understand the STACK option.

If you want to be absolutely certain that the LISTFILE command gets executed (and thus synonyms are bypassed), you have to code the CMS commands without abbreviation, as follows:

Address command

`LISTFILE * * * F (STACK)` /* instead of `L * * F (STACK)` */

`QUERY DISK (STACK)` /* instead of `Q DISK (STACK)` */

Now, even if the user defines LISTFILE to be a synonym of FLIST, the real LISTFILE command will be executed, as step three will result in an immediate match, and synonyms are ignored. This is not fully explained in the chapter about CMS search order in the manuals.

**Conclusions for CMS EXECs**

To make your procedures safer and perform better, code an address command at the start. Then:

- CP commands must be prefixed by the CP keyword.
- A call to an external procedure must be prefixed with the EXEC keyword.
- CMS commands should not be abbreviated.
- Many CMS commands won’t produce error or warning messages any more.
- You can omit most of the SET CMSTYPE HT commands because error messages are suppressed. Remember that with HT you suppress almost all messages, even the more severe ones, while address command will only suppress the less important ones (such as ‘File not found’).

Note that coding ‘address ’’’ is equivalent to ‘address command’ – it’s just a useful short notation.

The only instance where address CMS is justified is when your procedure accepts commands from the user and has to execute them. Most users do not always make the distinction between CMS, EXEC, or even CP commands.
XEDIT MACROS

The default environment for an XEDIT macro (filetype XEDIT) is XEDIT itself. It means that an implicit address XEDIT is used. This is good, but sometimes you may want to execute a CMS or CP command in an XEDIT macro. Then the rules described above still apply, but there is an extra player in the game – XEDIT itself.

If you enter a command on XEDIT’s command line (or code it in an XEDIT macro), without being explicit about the environment you want to address, then XEDIT will take this command for itself first.

If XEDIT does not understand the command, then it passes it along to CMS – this is only true if IMPCMSCP (implied CMS and CP) is set ON (which is the default, but the user may have changed it). From there on, CMS treats the command as with an address CMS, and hence is not foolproof any more. If it’s not recognized by CMS, it will pass it to the CP level (depending on IMPCP setting).

Now, let’s take the example of the SET command. XEDIT, CMS, and CP all have some form of SET command. So, if we aren’t explicit in our command, then XEDIT will take it for itself. If the parameters are not recognized by XEDIT (because we actually wanted to execute a CMS or CP SET command), then XEDIT will return a bad return code.

So, on the XEDIT command line, we all know we have to use the CP or CMS prefixes in order to address the correct environment if the command is not exclusive to a specific environment.

In our XEDIT macros, we can temporarily switch from the address XEDIT status to the address COMMAND status. This can be done in two ways:

- Code an address COMMAND as a separate statement, followed by one or more CMS or CP commands, and finally an address XEDIT statement to return to the XEDIT environment, as follows:

  Address command
  'CP SPOOL CONSOLE START'
  'ERASE TEMP FILE A'
  Address 'XEDIT'

- Prefix each CMS or CP command with address COMMAND. This is useful if you have only one or a few non-XEDIT commands
to issue. Thus, something like this:

Address command 'CP SPOOL CONSOLE START'

While we’re on the subject, remember that XEDIT also makes a distinction between basic commands and macros. Furthermore, XEDIT also allows defining synonyms for its commands. So, we can have similar problems in XEDIT – executing macros when we want a basic command – as we had with CMS. This time, however, we can control the execution of the commands or macros by explicitly using either the COMMAND or MACRO prefix in our statements.

Note that the COMMAND prefix here has nothing to do with our address command, it’s entirely an XEDIT built-in command. For example:

'COMMAND DELETE 2'
'MACRO ALL /This/ | /That/'

An alternative to this, although not so easy to control and to code, is to use the SET MACRO ON/OFF and SET SYNONYM ON/OFF commands to control the execution of macros, synonyms, or commands. Don’t forget to restore the settings before returning to the user.

**Conclusions for XEDIT macros**

For XEDIT commands, you should:

- Control the execution of XEDIT commands or macros via the COMMAND or MACRO prefix.
- Direct a command to CMS or CP (incidentally, also the call to a CMS EXEC), by temporarily switching to address COMMAND.

One further remark here, again in favour of address COMMAND, is that XEDIT truncates all commands to 255 characters. When an address COMMAND is in effect, they are not sent to XEDIT, and hence truncation does not occur. CMS Pipelines, for example, can become very large strings.
OTHER ENVIRONMENTS

SUBCOM

More and more products include an interface to REXX, allowing REXX procedures to use the functions of the products. Examples are ISPF, SQL, GDDM, CPI-C, and many others (also on other platforms like OS/2).

In most cases, these products define themselves as sub-environments (or to use the exact term, subcommands or SUBCOMs) to CMS. So, just as for XEDIT, they define their proper environment that can be ‘reached’ by REXX via the ‘address’ statement. So, for example, to address ISPF, you would use an address ISPEXEC statement, while CPI-C would be reached via an address CPICOMMM statement.

CMS Pipelines

Most of what we’ve said above also applies in some way to CMS Pipelines. First of all, PIPE is a regular CMS command, so use the address COMMAND to execute it from procedures or XEDIT macros. But CMS Pipelines can themselves execute CMS or CP commands through the CMS, COMMAND, and CP device drivers. These drivers execute the commands and, instead of displaying the result to the terminal, pump the results into the pipeline.

So what’s the magic? For CMS commands, the CMS or COMMAND stages intercept the CMS terminal output before it reaches the screen and pump the lines into the pipeline. For CP commands, the CP device driver issues a regular diagnose X'8', but asks CP to return the results in a buffer instead of displaying them at the terminal (just like REXX does via the ‘diag(8)’ function).

What to use? COMMAND, CMS, or CP device drivers

Because both REXX and CMS Pipelines use the same internal interface to pass commands to CMS, it is not a surprise that the CMS device driver has the same drawbacks as REXX’s address CMS, and that the COMMAND device driver behaves as with an address COMMAND in REXX.

It’s a shame that both REXX and CMS Pipelines manuals are still
primarily showing address CMS and CMS stages. We presume this is because they want to open to the broadest public – who may not know the difference between CP and CMS commands or procedures.

So, let’s repeat the advantages of the COMMAND driver:

• There is no full command resolution (thus, you have to use the EXEC prefix to execute EXEC procedures) leading to better performance and more foolproof pipelines (fewer leaking pipelines, less chance that the user has to call a plumber to fix it, etc).

• Commands and parameters are not translated to upper case, so you must code them in the proper format (commands in upper case, parameters can sometimes be in mixed case).

• Error messages are eliminated in many cases. Headers are also often suppressed.

This last item is much more important for CMS Pipelines than it is for REXX procedures. These drivers pump the output of the commands into the pipeline. What happens to the error messages will be explained later.

**Conclusions for CMS Pipelines**

For CMS Pipelines you should:

• Use the COMMAND stage to execute CMS commands. Use the CMS stage only when you execute commands a user feeds to your procedure.

• To intercept output from CP commands in your pipelines, you must use the CP device driver. CMS or COMMAND can issue CP commands, but then the output goes to the terminal, not into the pipeline.

• If you have used REXX for a long time, you will be used to the fact that the CMS command ‘QUERY xxx (STACK’ does not return a header, whereas, for example, the stage ‘COMMAND QUERY xxx’ does. Never assume that the output you get when executing a command from the terminal will be the same as when
you issue it with a COMMAND stage. So, use a little pipeline to test this:

"PIPE COMMAND LISTDIR|TAKE 5|CONSOLE".

A typical pipeline using COMMAND looks like this:

```bash
/* Coding pipelines */
Address command  /* hope you're convinced now */
'PIPE'          /* need upper case due to address command*/
'|command LISTFILE * * A', /* pipeline parameters can be lower case,*/
    /* but parameters of COMMAND stage must */
    /* be coded in upper case again... */
'|specs w2 1'    /* pipe parameter, thus can be lower case*/
'|CONSOLE'      /* pipe parameter, or upper case... */
exit rc
```

HOW TO GET THE ERROR MESSAGES

We have one last problem to solve. We said that many CMS commands called with the COMMAND stage would not produce error messages. But there may be cases where we would like to get these error messages in the pipeline anyway, to be able to display them at the terminal to inform the user.

Do we have to revert to the CMS stage then to get the error messages back? No, there is another useful CMS command – namely CMDCALL. This command takes other CMS commands as parameters and lets CMS execute these commands as if they came from the terminal (and thus error messages and headers are produced as normal).

If we combine the COMMAND stage (or address command as a matter of fact), with this CMDCALL command, as follows:

```
pipe command CMDCALL LISTDIR | > LISTDIR OUTPUT A
```

then:

- Because of the COMMAND stage, we are certain to execute the LISTDIR command and not a synonym or homonym procedure.
- Because of the CMDCALL, LISTDIR thinks it is executed from the terminal and thus issues error messages, as you would get if the command was issued from the CMS environment. However,
in this case, the output is pumped into the next pipeline stage, from where you can further process it. For example:

```
'PIPE COMMAND CMDCALL LISTDIR' somedir,
   '|VAR EMSG', /* any error message ? */
   '|DROP',     /* drop header */
   '|STEM DIR.' /* save result... */
if rc<>Ø then do
   say 'Unable to list directory' somedir
   say 'Error message is: '
   say emsg
   exit rc
end
do i=1 to dir.Ø
 ...
end
```

**CMS Pipeline REXX stages – a special case**

Pipe stages written in REXX have a default addressing so that PIPE subcommands can be recognized (eg PEEKTO, OUTPUT, etc). This addressing takes the form of a PSW pointing somewhere into the PIPE module. So, address PIPE does not exist! Furthermore, if you code an address command (on its own), then you cannot address CMS Pipelines any more, unless you have saved the address, as shown in the third example below.

If you have to issue CMS or CP commands from within REXX stages, use one of the following methods:

- Use ‘CALLPIPE COMMAND ....|CONSOLE’ or ‘CALLPIPE CP ...’.
- Use address command ‘some CP or CMS command’.
- Use the following coding sequence:
  ```rexx
  PipeAddress=address()    /* save the plumber's address */
  address command
  ...                        /* CP or CMS commands here */
  address value PipeAddress  /* return to the plumber */
  ```
- Use a REXX subroutine, such as:
  ```rexx
  call command 'somecommand'
  ...
  COMMAND:
  ```
GENERAL CONCLUSIONS
An illustration of everything we have said in this article will probably further help you understand the different options. The procedure below will accept any command as parameter and execute it once for all of the following cases:

- address CMS cmd

- address COMMAND cmd

- address COMMAND ‘CMDCALL’ cmd

- ‘PIPE CMS’ cmd ‘| CONSOLE’

- ‘PIPE COMMAND’ cmd ‘|CONSOLE’

- ‘PIPE COMMAND CMDCALL’ cmd ‘|CONSOLE’.

This procedure might also be useful in order to analyse the returned information (headers, error messages):

```c
/* TCMD EXEC - Analyse results of commands */
parse upper arg cmd
Say '*** Case 1 : Result of "address CMS' cmd'" ***'
address CMS cmd ; Say 'Retcode:' rc

Say '*** Case 2 : Result of "address COMMAND' cmd'" ***'
address COMMAND cmd ; Say 'Retcode:' rc

Say '*** Case 3 : Result of "address COMMAND CMDCALL' cmd'" ***'
address COMMAND 'CMDCALL' cmd ; Say 'Retcode:' rc

Say '*** Case 4 : Result of "PIPE CMS' cmd'|CONSOLE" ***'
address command 'PIPE CMS' cmd'|CONSOLE' ; Say 'Retcode:' rc

Say '*** Case 5 : Result of "PIPE COMMAND' cmd'|CONSOLE" ***'
address '' 'PIPE COMMAND' cmd'|CONSOLE' ; Say 'Retcode:' rc

Say '*** Case 6 : Result of "PIPE COMMAND CMDCALL' cmd'|CONSOLE" ***'
address '' 'PIPE COMMAND CMDCALL' cmd'|CONSOLE' ; Say 'Retcode:' rc
```

Sample output can be seen below:

tcmd erase no file
Only cases one, three, four, and six display the error message. In cases one and four, an ERASE EXEC could break your procedure:

tcmd state no file w

The same remarks apply here as in the first run:

tcmd Listfile profile exec
PROFILE EXEC AØ
Retcode: Ø

*** Case 4 : Result of "PIPE CMS LISTFILE PROFILE EXEC|CONSOLE" ***
PROFILE EXEC AØ
Retcode: Ø

*** Case 5 : Result of "PIPE COMMAND LISTFILE PROFILE EXEC|CONSOLE" ***
PROFILE EXEC AØ
Retcode: Ø

*** Case 6 : Result of "PIPE COMMAND CMDCALL LISTFILE PROFILE EXEC|CONSOLE" ***
PROFILE EXEC AØ
Retcode: Ø

Ready;

These all produce the same result, but we were lucky that there wasn’t a Listfile EXEC procedure:

tcmd access . t
*** Case 1 : Result of "address CMS ACCESS . T" ***
Retcode: Ø

*** Case 2 : Result of "address COMMAND ACCESS . T" ***
Retcode: Ø

*** Case 3 : Result of "address COMMAND CMDCALL ACCESS . T" ***
Retcode: Ø

*** Case 4 : Result of "PIPE CMS ACCESS . T|CONSOLE" ***
Retcode: Ø

*** Case 5 : Result of "PIPE COMMAND ACCESS . T|CONSOLE" ***
Retcode: Ø

*** Case 6 : Result of "PIPE COMMAND CMDCALL ACCESS . T|CONSOLE" ***
Retcode: Ø

Ready;

This gives the same output in all cases, but, if we issue the same command with an invalid directory name, then we get:

tcmd list . wrongdir
*** Case 1 : Result of "address CMS LISTDIR .WRONGDIR" ***
DMSJLD1184E Directory VMSYS:MAINT.WRONGDIR not found or you are not authorized for it
Retcode: 28

*** Case 2 : Result of "address COMMAND LISTDIR .WRONGDIR" ***
Retcode: 28

*** Case 3 : Result of "address COMMAND CMDCALL LISTDIR .WRONGDIR" ***
DMSJLD1184E Directory VMSYS:MAINT.WRONGDIR not found or you are not authorized for it
Retcode: 28

*** Case 4 : Result of "PIPE CMS LISTDIR .WRONGDIR|CONSOLE" ***
DMSJLD1184E Directory VMSYS:MAINT.WRONGDIR not found or you are not authorized for it
Retcode: 28

Retcode: 28
*** Case 5 : Result of "PIPE COMMAND LISTDIR .WRONGDIR|CONSOLE" ***
Retcode: 28

*** Case 6 : Result of "PIPE COMMAND CMDCALL LISTDIR .WRONGDIR|CONSOLE" ***
DMSJLD1184E Directory VMSYS:MAINT.WRONGDIR not found or you are not authorized for it
Retcode: 28

Having said all this, we now have clear consciences!

Kris Buelens and Guy De Ceulaer
Advisory Systems Engineers
IBM (Belgium) © IBM (Belgium) 1999

A full screen console interface – part 14

Editor’s note: this month we continue the code for the full screen console interface for Disconnected Service Machines (DSM). This article is an extensive piece of work which will be published over several issues of VM Update. It was felt that readers could benefit from the entire article and from the individual sections. Any comments or recommendations would be welcomed and should be addressed either to Xephon or directly to the author at fernando_duarte@vnet.ibm.com.

* Add temporary entry to RND table

ADDTEMP EQU * Create temporary entry
 USING ADDTEMP,R6
 ST R14,ADDTSV14
 LA RØ,RNDSIZE Entry length in double words
 LINK OBTAIN Allocate storage
 LR R5,R1 Address entry
 L RØ,RNDPTR Add it to list
 ST R5,RNDPTR
 ST RØ,RNDFWD
 SR RØ,RØ
 ST RØ,RNDOPT1 Clear all option bytes
 ST RØ,RNDPIDS Clear Send PATHID
 ST RØ,RNDPIDR Clear Receive PATHID
 OR RNDOPT1,RNDOTMP This is a temporary entry
 L R14,ADDTSV14

BR    R14  
SPACE  
DROP R6  
SPACE 3  
ENTRY RNLSTX,RNLNX,RNLSDX  
ENTRY RNLFLG01,SBININD,SBINIRS  
EXTRN RNCNAME  
SPACE  
ACSCCODE DC V(CSCSVP)  
ACSCDATA DC V(CSCSVPD)  
CNPRSV14 DS F  
STPRSV14 DS F  
SDPRSV14 DS F  
RCPRSV14 DS F  
CHKTSV14 DS F  
TMPRSV14 DS F  
ADDTSV14 DS F  
SPACE  
CC2    EQU   2  
SPACE  
RNLFLG01 DS X  
CC2    EQU   2  
APPCWKCN EQU   X'80'  
APPCWKST EQU   X'40'  
APPCWKSD EQU   X'20'  
APPCWKRC EQU   X'10'  
APPCWKCH EQU   X'08'  
APPCWKTM EQU   X'04'  
SPACE  
SBINIL DC F'32'  
SBINI DC H'32'  
DC H'0'  
DC F'28'  
SBINIID DC C'<CSC>$ID'  
SBININD DC C' '  
SBINIRS DC C' '  
RBINIL DC F'32'  
RBINI DS CL32  
RBINIID EQU RBINI+08,8  
RBININD EQU RBINI+16,8  
RBINIRS EQU RBINI+24,8  
SPACE  
TIMEPT DS D  
TIMEMIN DC F'30'  
TIMEMIC DC F'60000000'  
*IMEMIC DC F'00400000'  
TIMEINT DC X'00000400'  
TIMECTL0 DS F  
SPACE 3  
CSCDATA
CSCUSA ASSEMBLE

CSC uses only APPC/VM to establish links with other CSC service machines. It allows you to monitor DSMs controlled by another CSC service machine on the same or another VM node in a TSAF collection. Please note that this code is not fully tested. To activate it:

- Authorize the CSC service machines to define LOCAL or GLOBAL resources, adding an IUCV statement to the CP directory similar to:

  IUCV *IDENT RESANY GLOBAL

- Add class 07 to the USER statements in the configuration file.
- Add LOCAL and REMOTE statements to both configuration files. For example:

  CONSOLES on VM1                     CONSOLES on VM2
  LOCAL CONVM1 RSØ1 GLOBAL            LOCAL CONVM1 RSØ2 GLOBAL
  REMOTE CONVM2 RSØ2                  REMOTE CONVM2 RSØ1

where RSØ1 and RSØ2 are APPC/VM resources, and CONVM1 and CONVM2 are node names given to the CSC service machines. If both machines are on the same VM node replace option GLOBAL with LOCAL.

Once the link between the two CSC machines is established, from a session with CONVM1 use the command CONNECT CONVM2 to monitor the DSMs controlled by CONSOLES on VM2.

PROGRAM CODE

```
TITLE 'CSCUSA - CSC Remote Node User Commands'
CSCUSA START X'01AFC8'
PRINT NGEN
CSCHDR                        APPC/VM User Commands
```
* Process APPC/VM User Commands
* 
* Input R5 points to the RND entry
* R4 addresses the APPC/VM receive buffer
* 
* USING UIDSECT,R8              UID (user) Block
USING RNDSECT,R5              RND Table
USING APPSECT,R4              APP (APPC/VM) command format

SPACE
ST    R5,USASVØ5              Save RND entry address
LA    R14,USA9ØØ              Return address for routines
CLC   APPCMD,APPC$SS          Start session
BE    SSESSION
L     R1,SSSPTR               Address Active Sessions List
USA1ØØ   LA    R2,APPVMID              Address user name for message
LA    R3,APPORIG              Address original node name
MSG   Ø559                    Invalid data received
B     USA9ØØ

    USA8ØØ   LA    R2,APPVMID              Address user name for message
LA    R3,APPORIG              Address original node name
MSG   Ø56Ø                    Session not found
B     USA9ØØ
BACK
SPACE 3

* Start session for connected user
* 
* Runs on connected node
*
SSESSION EQU *       <CSC>$SS Start session
LA  R0,UIDSIZE       Length of UID block
LINK OBTAIN          Allocate storage
LR  R8,R1            Address new allocated block
MVC UIDSECT(UIDSIZEB),APPDATA Copy originating UID block
L   R0,SSSPTR        Add it as an Active Session
ST  R0,UIDFWD
ST  R8,SSSPTR
LA  R0,UIDSCRSZ      Get screen size
LINK OBTAINP         Allocate storage (page aligned)
ST  R1,UIDSCRN       Save address in UID block
ST  R1,UIDSCRNA      Alternate screen not used yet
LA  R0,UIDBUFFSZ     User buffer size
LINK OBTAINP         Allocate storage (page aligned)
ST  R1,UIDBUFF       Save address in UID block
L   R0,RNDPIDS       Load Send PATHID
ST  R0,UIDPIDRM      That's the way to go back
LA  R2,UIDORIG       Original node name
LA  R3,UIDVMID       Original user-id
L   R8,SSSPTR        Address list of active sessions
SS1ØØ    L   R8,SSSPTR   Skip first entry just created
LTR   R8,R8          Check for end of table
BZ    SS2ØØ          Found it, some good news
CLC   UIDORIG,Ø(R2)  Compare node names
BNE   SS1ØØ          Not this one
CLC   UIDVMID,Ø(R3)  Compare user-ids
BNE   SS1ØØ          Not this one
SS4ØØ    L   R8,UIDFWD   Skip first entry just created
ST  R0,UIDCLASS      Store new classes
DROP  R1             SPACE
MVI   UIDOPT1,UIDRMTE Reset all but Remote User option
OI    UIDOPT2,UIDAUTO Put new session in refresh mode
OI    UIDOPT2,UIDTIME Display time, but that is all
NI    UIDOPT2,X'FF'-UIDDATE-UIDUSER-UIDEXC
MVI   UIDOPT3,X'ØØ'  Reset all other display options
MVI   UIDOPT4,UIDBTL+UIDBHDR Rebuild screen Title and Header

MVI UIDOPT5,UIDA$SC  Session Created
MVI UIDOPT6,UIDA$RN  Refresh CNN on user screen
MVI UIDCOL1,X'ØØ'  Display from column one
GO CSCUIN  Build user buffers
GO CSCUSCRH  Rebuild Header line
L R5,USAVØ5  Address RND entry
BAS R14,SEND  Send data back to the user
LA R2,UIDVMID  Address original user-id and node
LA R3,UIDORIG
MSG Ø550  Display info message
B USA900
SPACE

SS600  L R8,SSSPTR  Address current UID block
LA R4,CSCLOCAL
MSG Ø552,USER  Session already active
B SS800
SPACE

SS700 LA R4,CSCLOCAL
MSG Ø553,USER  User not authorized
B SS800
SPACE

SS800 MVI UIDOPT1,UIDRMTE  Reset all but Remote User option
MVI UIDOPT5,UIDA$SR  Set only Session Rejected
MVI UIDOPT6,UIDA$RU  Release UID block after Send
BAS R14,SEND  Send message back
B USA900
SPACE 3
*

Start Session accepted - Session Created
*

Runs on local and routing nodes
*

CSESSION EQU *  <CSC>$SC Session Created
NI UIDOPT3,X'FF'-UIDCREQ  Remember connect is complete
LA R0,UIDCSCSZ  Length of work CSCBUFF
LINK OBTAIN  Allocate storage
ST R1,UIDCSC  Save address in UID block
TM UIDOPT1,UIDRMTE  Is this first level connect
BO CS100  No, it is ping-pong
LA R0,UIDSCRSZ  Yes, load screen size
LINK OBTAINP  Allocate alternate screen
ST R1,UIDSCRNA  Save address in UID block
CS100 L R0,RNDPIDS
ST R0,UIDPIDCN  Store Send (connect) PATHID
LA R2,UIDVMID  Address user-id for message
LA R3,RNDNODE  Address connected node name
MSG Ø551  Display info message
BAS R14,DISPLAY  Display screen from connected
B USA900
SPACE 3
* Start Session not accepted - Session Rejected

* Runs on local and routing nodes

RSESSION EQU *    <CSC>$SR Remote session not created
BAS R14,DISPLAY    Just do it
NI UIDOPT1,X'FF'-UIDCONN Connect was rejected
NI UIDOPT3,X'FF'-UIDCREQ Connect request is complete
B USA9ØØ
SPACE 3

* Process user request and send results back
* Runs on connected and routing nodes

DSESSION EQU *    <CSC>$SD Process user request
LA R1,APPLLEN Address data length prefix
A R1,APPLLEN Address end of data
LA RØ,APPDATA Address data area (skip header)
SR R1,RØ Calculate data length
TM UIDOPT1,UIDCONN Are we connected?
BZ DS1ØØ No, process command
L RØ,UIDPIDCN Load PATHID to next node
LA R5,RNDPTR Address RND table
DSØ2Ø L R5,RNDFWD Scan it
BZ DSØ8Ø End of table found, link lost
CLM RØ,B'11ØØ',RNDPIDS Look for return link
BNE DSØ2Ø Not this one
TM UIDOPT3,UIDCREQ Really connected?
BZ DSØ3Ø Yes, forward data to next node
LA R3,RNDNODE No, we are connect pending
MSG Ø569,USER Send message to user
GO CSCBLD Build user screen
B DS2ØØ Ignore user request
SPACE
DSØ3Ø ST R1,UIDCSCL Store length in UID block
L R2,UIDCSC Address UID work buffer
BCTR R1,Ø Prepare to EXecute
EX R1,DSMVC Copy input data to work buffer
OI UIDOPT5,UIDA$SD Set option
BAS R14,SEND Forward data to next node
B DS9ØØ
SPACE
DS1ØØ LA R2,CSCBUFF Address input buffer
BCTR R1,Ø Prepare to EXecute
EX R1,DSMVC Move data to CSCBUFF
LA R1,CSCBUFFE+1(R1) Address end of data
ST R1,CSCBUFFF Save address for CSCUSC
MVI Ø(R1),C' '              Terminate data for MSG scanner
ST  R5,DSESV05              Save address of RND entry
GO  CSCUSC                  Process user command
LTR R8,R8                   Was it a Disconnect?
BZ  DS900                   Yes, user is gone (disconnect)
TM  UIDOPT1,UIDCONN        Was it a Connect?
BO  DS900                   Yes, do not send any data back
DS200 OI UIDOPT5,UIDA$SU    Remember what to do next
L  R5,DSESV05              Restore our RND entry
BAS R14,SEND                Send results back to the user
B  DS900

SPACE

DS800 MSG 8888
DS900 B  USA900
SPACE
DSMVC MVC Ø(*-*,R2),APPDATA Copy user command
SPACE 3

* Send data from connect node back to user
* Runs on local and routing nodes
*
USESSION EQU * <CSC>$SU Send data back to user
BAS R14,DISPLAY Just do it
B  USA900
SPACE 3
*
* Session ended
* Runs on local node
*
ESESSION EQU * <CSC>$SE Session ended
ST  R14,ES ESV14
NI  UIDOPT1,X'FF'-UIDCONN Reset Connect option
LA  R0,UIDCSCSZ
L  R1,UIDCSC
LINK RELEASE Release work CSCBUFF
TM  UIDOPT1,UIDRMTE Is user remote?
BO  ES100
LA  R0,UIDSCRSZ
L  R1,UIDSCRNA
LINK RELEASE No, release alternate screen
L  R0,UIDSCRN Load screen buffer address
ST  R0,UIDSCRNA Store as alternate screen
ES100 OI UIDOPT4,UIDBSCR+UIDBTL+UIDBHDR Rebuild user screen
OI UIDOPT6,UIDA$RN Update CNN on user screen
TM  UIDOPT1,UIDRMTE Are we still remote
BO  ES300 Yes, do not build 3270 DS yet
GO  CSCUSCRH Rebuild Header line
GO  CSCBLD Build user screen (3270 DS)
L R2,UIDSCRN         Address Data Stream built
A R2,UIDSCRNL        First free byte
MVC 4(L'COMMCCNN,R2),COMMCCNN Create <CSC>CNN command
MVC 4+L'COMMCCNN(L'CSCLOCAL,R2),BLANKS Clear CNN field
NI UIDOPT6.X'FF'-UIDA$RN Reset CNN option
LA R0,4+L'COMMCCNN+L'CSCLOCAL
ST R0,0,(R2)          Store length prefix
A R0,UIDSCRNL         Add lengths
ST R0,UIDSCRNL        Store length of new Data Stream
ES300 BAS R14,DISP#ES Display it
L R5,USASVØ5          Restore address of current RND
LA R2,UIDVMID         Address user-id
LA R3,RNDNODE         Address node name
MSG Ø562               Display info message
L R14,ESESV14         Display it
BR R14
SPACE 3

* Terminate connected session
* Runs on connected and routing nodes
* TSESSION EQU * <CSC>$TC Terminate session
LA R2,UIDVMID         Address user-id
LA R3,RNDNODE         Address node name
MSG Ø574               Display info message
TM UIDOPT1,UIDCONN    Are we still connected?
BZ TS100               Yes, terminate next session
GO CSCUSATC
B USA900
SPACE
TS100 BAS R14,RELEASE Release buffers and UID block
B USA900
SPACE 3

* Send user request to destination node
* Runs on local node. Invoked by CSCUSC
* CSCUSASD RELOC Send data to destination node
L R0,UIDPIDCN
LA R5,RNDPTR          Address RND table
SD100 L R5,RNDFWD      Scan it
LTR R5,R5
BZ SD000              End of table found, not defined
CLM R0,'1100',RNDPIDS  Not this one
TM UIDOPT3,UIDCREQ    Is user Connect pending?
BZ SD200              No, process request
LA R3,RNDNODE         Yes, address remote node name
**Process CONNeCt command**

* Runs on local and routing nodes

**CSCUSACN RELOC**

* TM UIDOPT1,UIDRMTE Is user already connected?
  * BO CONN2ØØ For now only one level possible
  * SR RØ,RØ No table to search
  * GO CSCSCN Get node name
  * BNZ CONN3ØØ Nothing found, display error
  * LA RØ,8 Check word length
  * CR RØ,R1 Too long
  * BL CONN4ØØ
  * SR RØ,RØ Check for extra parameters
  * GO CSCSCN
  * BZ CONN5ØØ Something found, that's bad
  * LA R5,RNDPTR Address RND table

**CONN1ØØ**

L R5,RNDFWD Scan it

LTR R5,R5

BZ CONN6ØØ End of table found, not defined

CLC RNDNODE,SCANUPP Compare node names

BNE CONN1ØØ Not this one

TM RNDOPT1,RNDOLCL Is this the local node

BO CONN7ØØ Yes, destination must be remote

TM RNDOPT1,RNDOSND+RNDORCV Is link up (send and receive)

BNO CONN8ØØ No, display message
OI    UIDOPT1,UIDCONN      Set connect option
OI    UIDOPT3,UIDCREQ     Remember connect is in progress
OI    UIDOPT5,UIDA$SS    Start Session
BAS   R14,SEND           Contact the connected node

CONN9ØØ BACK
SPACE
CONN2ØØ MSG  Ø874,USER    User already connected
  B    CONN9ØØ
SPACE
CONN3ØØ MSG  Ø310,USER    Missing operands
  B    CONN9ØØ
SPACE
CONN4ØØ MSG  Ø87Ø,USER    Operand too long
  B    CONN9ØØ
SPACE
CONN5ØØ MSG  Ø311,USER    Invalid operand
  B    CONN9ØØ
SPACE
CONN6ØØ LA    R2,SCANUPP   Destination node not found
MSG  Ø871,USER  
  B    CONN9ØØ
SPACE
CONN7ØØ MSG  Ø872,USER    Cannot connect to local node
  B    CONN9ØØ
SPACE
CONN8ØØ LA    R2,RNDNODE   Link not active
MSG  Ø873,USER  
  B    CONN9ØØ
SPACE 3
*
* Process DISConnect command
*  
*    Runs (supposedly) on connected node
*  
CSCUSADN RELOC               Process DISConnect command
TM    UIDOPT1,UIDRMTE       Is user remote?
BZ    DISC4ØØ               No table to search
SR    RØ,RØ                 Check for extra operands
GO    CSCSCN                 Something found, display error
BZ    DISC5ØØ               Address RND table
L     RØ,UIDPIDRM           Scan it
LA    R5,RNDPTR             Scan it
LTR   R5,RNDFWD             End of table found, link lost
BZ    DISC6ØØ               Look for return link
CLM   RØ,'B'1100',RNDPIDS   Not this one
BNE   DISC1ØØ               Is link up (send and receive)
TM    RNDOPT1,RNDOSND+RNDORCV No, display message
BNO   DISC8ØØ               Address original user-id and node
LA    R2,UIDVMID           Address original user-id and node
LA    R3,UIDORIG
MSG    Ø554           Display info message
OI    UIDOPT5,UIDA$SE Session ended
OI    UIDOPT6,UIDA$RU Release UID block after Send
BAS    R14,SEND      Send data back
DISC9ØØ BACK
SPACE
DISC4ØØ MSG    3212,USER User not connected
B      DISC9ØØ
SPACE
DISC5ØØ MSG    Ø311,USER Invalid operand
B      DISC9ØØ
SPACE
DISC6ØØ LA    R2,UIDORIG
MSG    Ø871,USER     Return link lost
B      DISC9ØØ
SPACE
DISC8ØØ LA    R2,RNDNODE
MSG    Ø873,USER     Link not active
B      DISC9ØØ
SPACE 3
*
* Send data back to user for display
*  
* Runs on connected node
*
CSCUSADP RELOC Send data back to user
L     R0,UIDPIDRM
LA    R5,RNDPTR     Address RND table
DP1ØØ L     R5,RNDFWD Scan it
LTR    R5,R5
BZ    DP6ØØ         End of table found, link lost
CLM    R0,'B'11ØØ*,RNDPIDS Look for return link
BNE    DP1ØØ         Not this one
TM    RNDOPT1,RNDOSND+RNDORCV Is link up (send and receive)
BNO    DP8ØØ         No, display message
OI    UIDOPT5,UIDA$SU Option to send data to user
BAS    R14,SEND
DP9ØØ BACK
SPACE
DP6ØØ LA    R2,UIDORIG
MSG    Ø871,USER     Return link lost
B      DP9ØØ
SPACE
DP8ØØ LA    R2,RNDNODE
MSG    Ø873,USER     Link not active
B      DP9ØØ
SPACE 3
*
* Process pending requests
* Runs on all nodes

CSCUSAPD RELOC

L R0,RNDPIDS
Load IUCV PATHID (first 2 bytes)
L R1,SSSPTR
Address list of active sessions

PD100
LTR R8,R1
Scan list
BZ PD800
End of list found, all done
L R1,UIDFWD
Address next entry
TM UIDOPT5,UIDAPEND
Anything pending on this one?
BZ PD100
No, try next one
TM UIDOPT5,UIDA$SS
Is it sending data?
BZ PD200
No, check receiving path
CLM R0,B'1100',UIDPIDCN
Check IUCV PATHID
BNE PD100
Not this one
B PD300
We found it, process request

SPACE

PD200 CLM R0,B'1100',UIDPIDRM
Check receiving path
BNE PD100

PD300
NI UIDOPT5,X'FF'-UIDAPEND
Reset pending option
BAS R14,SEND
Process pending request
B PD900

SPACE

PD800
NI RNDOPT2,X'FF'-RNDOFND
Nothing left, reset option
* B PD900

SPACE

PD900 BACK
SPACE 3

*  

* Terminate connected session

* Runs on local node

* Output R2 addresses the current UID block if not released
  or the previous UID block otherwise

CSCUSATC RELOC

L R0,UIDPIDCN
Terminate connected session
LR R2,R8
Address of current UID block
LA R5,RNDPTR
Address RND table

TC100
L R5,RNDFWD
Scan it
LTR R5,R5
End of table found, link lost
CLM R0,B'1100',RNDPIDS
Look for return link
BNE TC100
Not this one
TM RNDOPT1,RNDSND+RNDOFND
Is link up (send and receive)
BNE TC900

OI UIDOPT5,UIDA$TC
Terminate connected session
OI UIDOPT6,UIDA$RU
Release UID block after Send
BAS R14,SEND
Send request to connected node
* Terminate sessions affected by one APPC/VM link
* Input R5 addresses RND entry being processed

```
CTR R8,R8       Was UID block released?
BZ TC9ØØ       No, use current as previous
LR R2,R8
TC9ØØ BACK
SPACE 3
*
* Was UID block released?
* No, use current as previous
* Input R5 addresses RND entry being processed
* Clean-up dead sessions
CTCUSACL RELOC
ST R5,CLNSVØ5  Save RND entry address
LA R8,SSSPTTR  Address list of active sessions
CLEAN1ØØ L R8,UIDFWD  Scan table
LTR R8,R8      Check for end of table
BZ CLEAN9ØØ  Found it, all done
TM UIDOPT1,UIDCONN  Is user connected?
BZ CLEAN6ØØ  No, check if remote
L RØ,RNPDIDS
CLM RØ,B'11ØØ',UIDPIDCN  Is user connected on this link?
BNE CLEAN6ØØ  No...
LA R2,UIDVMID  Address user and node names
LA R3,RNDNODE
MSG Ø58Ø      Display info message
MSG Ø581,(USER,NOCMD)  Tell the user about the failure
BAS R14,ESSESSION  Display new user screen
B CLEAN1ØØ  
SPACE
CLEAN6ØØ TM UIDOPT1,UIDRMTE  Is user remote?
BZ CLEAN1ØØ  No, check all sessions
L RØ,RNPDIDS
CLM RØ,B'11ØØ',UIDPIDRM  Is user remote on this link?
BNE CLEAN1ØØ  No...
LA R2,UIDVMID  Address user and node names
LA R3,RNDNODE
MSG Ø582      Display message
TM UIDOPT1,UIDCONN  Is user also connected (routing)
BO CLEAN7ØØ
BAS R14,RELEASE  No, just release UID block
B CLEAN8ØØ  
SPACE
CLEAN7ØØ GO CSCUSATC  Terminate all forward sessions
L R5,CLNSVØ5  Restore RND entry address
CLEAN8ØØ LR R8,R2  Address previous entry
B CLEAN1ØØ  Scan all sessions
SPACE
CLEAN9ØØ L R5,CLNSVØ5  Restore RND entry address
BACK
SPACE 3
*
```
* Send data to next node
* Input R5 addresses RND entry being processed

SEND EQU * Send data to connected node
TM RNDOPT2,RNDOSPG Is last Send finished?
BZ SEND100
OI RNDOPT2,RNDOPND No, link has pending requests
OI UIDOPT5,UIDAPEND Session has a pending request
BR R14

SPACE
SEND100 ST R14,SNDSV14
L R4,RNDSBUFF Address APPC/VM Send buffer
MVC APPORIG,UIDORIG Identify originating node name
MVC APPVMID,UIDVMID Identify originating user name
TM UIDOPT5,UIDASOND Send data to connected node?
BZ SEND200 No, it must be something else
NI UIDOPT5,X'FF'-UIDASOND Reset option
MVC APPCMD,APPC$SOND Copy command name
L R2,UIDCSC Address work buffer
L R1,UIDCSCN Length of data
LA R0,APPDATA(R1) Address end of data
BCTR R1,Ø Prepare to EXECute
EX R1,SENDMVC Copy data to APPC buffer
B SEND700 Send the data

SPACE
SEND200 TM UIDOPT5,UIDASOND Check type of request
BZ SEND220
NI UIDOPT5,X'FF'-UIDASOND Reset Start Session request flag
MVC APPCMD,APPC$DSN Copy command name
MVC APPDATA(UIDSIZEB),UIDSECT Send a copy of the UID block
LA R0,APPDATA+UIDSIZEB Address next available byte
B SEND700 Send the data

SPACE
SEND220 TM UIDOPT5,UIDASEN Check type of request
BZ SEND240
NI UIDOPT5,X'FF'-UIDASEN Reset Session Ended flag
MVC APPCMD,APPC$SEN Copy command name
LA R0,APPDATA Address next available byte
B SEND700 Send the data

SPACE
SEND240 TM UIDOPT5,UIDATC Check type of request
BZ SEND300
NI UIDOPT5,X'FF'-UIDATC Reset Terminate Connected
MVC APPCMD,APPC$TC Copy command name
LA R0,APPDATA Address next available byte
B SEND700 Send the data

SPACE
SEND300 TM UIDOPT1,UIDCONN Are we connected?
BO SEND320 Yes, do not rebuild screen
ST R5,SNDSVØ5               Save RND entry address  
TM UIDOPT4,UIDBHDR             Should we rebuild the Header?  
BZ SEND31Ø                      No, so don't do it  
GO CSCUSCRH                     Rebuild Header line  
SEND31Ø GO CSCBLD           Rebuild user screen (327Ø DS)  
L R5,SNDSVØ5                    Restore our RND entry address  
SEND32Ø L R4,RNDSBUFF                 Address Send buffer  
L R3,UIDSCRNL                    Length of screen from CSCBLD  
L R2,UIDSCRN                     Address of screen Data Stream  
LR R1,R3                           Copy length for MVCL  
LA R0,APPDATA                    Address APPC data area  
MVCL R0,R2                         Let's do it  
LR R1,R0                           Address of next available byte  
TM UIDOPT5,UIDA$SU               Just send data back to the user?  
BZ SEND40Ø                      Address Send Buffer  
NI UIDOPT5,X'FF'-UIDA$SU               Yes, reset option  
MVC APPCMD,APPC$SU               Send data back  
B SEND60Ø  
SPACE  
SEND40Ø TM UIDOPT5,UIDA$SC         Is this a Session Created  
BZ SEND500                           Address Send Buffer  
NI UIDOPT5,X'FF'-UIDA$SC               Yes, reset option  
MVC APPCMD,APPC$SC               Session Created command  
B SEND600                             Send the data  
SPACE  
SEND500 TM UIDOPT5,UIDA$SR         Is this a session rejected  
BZ SEND510                          Address Send Buffer  
MVC APPCMD,APPC$SR               Session Rejected command  
B SEND700                             Send the data  
SPACE  
SEND510 EQU *  
MSG 1111                             
MSG 2222                             
MSG 3333                             
BACK  
SEND60Ø TM UIDOPT6,UIDA$RN         Refresh CNN on user screen?  
BZ SEND700                          Address Send Buffer  
NI UIDOPT6,X'FF'-UIDA$RN               Yes, reset option  
MVC 4(L'COMMCCNN,R1),COMMCCNN        Create <CSC>CNN command  
MVC 4+L'COMMCCNN(L'CSCLOCAL,R1),CSCLOCAL  
LA R0,4+L'COMMCCNN+L'CSCLOCAL        Set length  
ST R0,0(,R1)                          Store command length  
AR R0,R1                              Address next available byte  
SEND700 LR R1,R0                         Copy end address to R1  
LA R2,APPLEN                          Address CSC data  
SR R0,R2                              Calculate length of CSC data  
ST R0,APPLEN                          Store length prefix  
SR R1,R4                              Address next available byte  
STH R1,APPLL                           Store APPC/VM Logical Rec length  
GO CSCRNCSD                             Send the data
Do we need the UID block

No, release it

* Send screen Data Stream to user
* Runs on local and routing nodes
* DISP#ES is invoked by ESESSION and CSCUSASD

Send data to user
Address data area
Get length of all CSC data
End address of CSC data
Get length of 3270 data stream
Address user alternate buffer
Copy for MVCL
Store screen length
Move data to user screen

Send return PATHID
Address RND table
Scan it
End of table found, link lost
Look for return link
Not this one
Set option
Send data back one node

Editor’s note: this article will be continued next month.

Fernando Duarte
Analyst (Canada)
REXX/CMS talks to VB over TCP/IP

Wouldn’t it be nice if we were able to:

• Insert the contents of a CMS file directly into a Word document with just a few mouse-clicks.

• Create an Excel spreadsheet containing the names of all your VM DB2 tables, without having to bother about database connectivity software.

• Insert a formatted and comprehensive description of a VM DB2 table into a Word document.

As administrator of a VM/ESA system, I am often required to write some kind of system documentation. Of course, in ‘the old days’ this documentation was in the form of CMS files, but nowadays I prefer to use Microsoft’s Word and Excel. Until now, this meant either having to type VM system information or using a file transfer program. I thought it would be great to have direct access to various kinds of VM system information directly from Word or Excel. This led me to build the InfoServer, a system consisting of some CMS/REXX procedures and Visual BASIC for Application (VBA) macros which communicate over a TCP/IP socket connection. The server part runs in a disconnected CMS machine, the clients are implemented as VBA macros in Word documents and Excel worksheets.

I have only just started using InfoServer, so the three clients are merely examples of what might be achieved. The configuration file INFOSRV CONFIG allows some flexibility, so that the procedures INFOSRV1, INFOSRV2, and INFOSRV3 should need no changes. For each request type, a request handler procedure needs to be created in the CMS machine (see IS0001, IS0002, and IS0003 as examples). To prevent access from unwanted clients, I also included a rather basic security check mechanism based on IP addresses.

CMS COMPONENTS

The CMS components of InfoServer are as follows:

• INFOSRV CONFIG – the configuration file for the server.
• INFOSRV1 EXEC – the InfoServer start-up procedure.
• INFOSRV2 REXX – the InfoServer main stage.
• INFOSRV3 REXX – the InfoServer client request handler.
• IS0001 REXX – request handler for InfoClient1 (REQ1).
• IS0002 REXX – request handler for InfoClient2 (REQ2).
• IS0003 REXX – request handler for InfoClient3 (REQ3).

MS OFFICE COMPONENTS
The MS Office components are as follows:
• InfoClient1.wks – client example: get DB2 table list into an Excel spreadsheet.
• InfoClient2.doc – client example: get CMS file contents into a Word document.
• InfoClient3.doc – client example: get DB2 table description into a Word document.

Prerequisites for InfoServer are VM/ESA 2.3 with TCP/IP, CMS Pipelines, and REXX/SQL; Word 97; Excel 97; and Microsoft Winsock Control/Library.

REXX COMPONENTS

INFOSRV CONFIG

/*====================================================================*/
/* INFOSRV CONFIG */
/*====================================================================*/
/* THE PORT ON WHICH INFOSERVER WAITS FOR CLIENT CONNECTIONS */
[PORT]
4444
/* CLIENTS WHICH ARE ALLOWED ACCESS TO INFOSERVER */
/* THREE WORDS FOR EACH CLIENT DEFINITION: */
/* IPADDRESS */
/* ALIAS */
/* CODEPAGE */
[CLIENTS]
121.33.77.123 PC1 STANDARD
121.33.77.124 PC2 437
/* REQUEST DEFINITIONS: */
/* WORD1: REQUEST ID */
/* WORD2: FILENAME OF PIPELINE STAGE WHICH HANDLES REQUEST */

[REQUESTS]
REQ1      IS0001
REQ2      IS0002
REQ3      IS0003

INFOSRV1 EXEC

/*--------------------------------------------------------------------------------*/
/* Name         : INFOSRV1 EXEC */
/* Application   : InfoServer */
/* Usage        : Procedure */
/* Arguments    : - */
/* Result       : - */
/* Function     : InfoServer Start-up Procedure */
/* This procedure initializes the InfoServer */
/* It should be called in the PROFILE EXEC of the disconnected */
/* CMS machine running the InfoServer */
/* It starts a pipeline with TCPLISTEN and directs the stream */
/* to INFOSRV2 REXX */
/*--------------------------------------------------------------------------------*/
.pipe < infosrv config a',
  | nlocate 1-3 - - - -
  | nlocate 1-2 -/*- - -
  | frlab [PORT] | drop 1'
  | tolabel [CLIENTS] | take 1 | var port'
  .vmfclear'
say 'InfoServer initializing on port' port
.pipe tcplist' port | infosrv2'

INFOSRV2 REXX

/*--------------------------------------------------------------------------------*/
/* Name         : INFOSRV2 REXX */
/* Application   : InfoServer */
/* Usage        : Pipeline Stage Command */
/* Arguments    : - */
/* Result       : - */
/*--------------------------------------------------------------------------------*/
/* Function : InfoServer Main Stage */
/* This stage must be run after a TCPLISTEN stage, as in INFOSRV1 */
/* InStream Ø : OutStream Ø produced by TCPLISTEN */
/* OutStream Ø : - */
/* This pipeline stage procedure waits for incoming requests from */
/* clients, performs security checking, and starts a separate */
/* pipeline with INFOSRV3 REXX for each client connection */
/*====================================================================*/
/*====================================================================*/
/* read client definitions */
/*====================================================================*/
callpipe < infosrv config a'
   | nlocate 1-3 - - - 
   | nlocate 1-2 -/*-*- 
   | frlab [CLIENTS] | drop 1' 
   | tolabel [REQUESTS] | stem clients'
/*====================================================================*/
/* main server loop */
/*====================================================================*/
say 'InfoServer now waiting for client connections'
signal on error
do forever
   peekto peeky' /* wait for client connection */
   ipstruct=substr(peeky,65,16) /* client network address */
   callpipe var ipstruct | socka2ip | var ipstruct'
   client_ipaddr=word(ipstruct,3) /* client ip address */
   /* security checks */
   allowed=Ø
   do i=1 to clients.Ø
      parse value clients.i with ipaddr name codep
      if client_ipaddr = ipaddr
         then
d         allowed=1
         leave
      end
   if allowed
      then
         say 'Request from client:' ipaddr name
else
do
   'readto rubbish' /* consume unwanted request from client */
   say 'Request from unknown client' client_ipaddr 'refused'
   iterate
end
/*====================================================================*/
INFOSRV3 REXX

```rexx
/* and pass it a record */
.addpipe *.output: | i: fanin ,
.| tcpdata | elastic | infosrv3' client_ipaddr codep name , | i:'
.callpipe *: | take 1 | *:'
.sever output'
end
exit
error: exit RC*(rc<>Ø)
```

INFOSRV3 REXX

```rexx
/* Name        :  INFOSRV3 REXX */
/* Application :  InfoServer */
/* Usage       :  Pipeline Stage Command */
/* Arguments   :  ipaddress codepage name */
/* Result      :  - */
/* Function    :  Handle InfoServer Client Request */
/* InStream Ø :  one data record from the client */
/* OutStream Ø :  data to be returned to client */
/* This procedure handles a client request. */
/* It is called by INFOSRV2 REXX for each request */
/* It calls the pipeline stage defined in INFOSRV CONFIG for the */
/* request and handles all translation according to the clients */
/* codepage */
```
do forever
    'READTO request_record'
    if rc <> Ø then leave
    request_record=xlate('FROM',request_record)
    req_id=word(request_record,1)
    /* check if request is defined in config file */
    if module.req_id = ',' then do
      say 'Unknown request' req_id 'cannot be processed'
      orec= 'Unknown request' req_id 'cannot be processed'
      'output' xlate('TO',orec)
      return
    end
    /* process request and return requested information */
    say 'Request:' req_id ' Module called:' module.req_id
    callpipe var request_record ,
    'callpipe var request_record' ,
    ,|' module.req_id ,
    ,| var retval
    /* append end-of-data marker to output stream data */
    retval=retval || '##eod##'
    'output' xlate('TO',retval)
  END
  return
  /*====================================================================*/
  /* xlate()                                                            */
  /* translates as string according to the clients codepage             */
  /*====================================================================*/
  xlate:
  parse arg opt,the_string
  /* translate input according to codepage for this client */
  select
    when translate(codepage) = 'NONE' then nop
    when translate(codepage) = 'STANDARD'
    then do
      if translate(opt) = 'TO'
      then opt='E2A'
      else
        opt='A2E'
      callpipe var the_string ,
      ,| xlate opt ,
      ,| var the_string
    end
  ISØ001 REXX
  /*====================================================================*/
  /* Name        :  ISØ001   REXX                                       */
  /*====================================================================*/
/* Application :  InfoServer                                  */
/* Usage       :  Pipeline Stage Command                      */
/* Arguments   :  -                                           */
/* Result      :  -                                           */
/* Function    :  Request Handler for REQ1 (DB2 Table List)  */
/* InStream Ø :  Request_Record                              */
/* OutStream Ø :  Requested_Data_Stream_for_Client           */
/*====================================================================*/
/*====================================================================*/
/* main processing logic                                        */
/*====================================================================*/
'readto request_record'
parse value request_record with req_id req_data
dbname=strip(req_data)
call do_connect
call do_select
ADDRESS COMMAND ,RXSQL COMMIT RELEASE'
return
/*====================================================================*/
/*====================================================================*/
/* perform SQL CONNECT                                          */
/*====================================================================*/
do_connect:
USER_STRING='SQLOWNER IDENTIFIED BY HANDSOFF'
ADDRESS COMMAND ,RXSQL CONNECT' user_string ,TO' dbname
sqlrc=rc
if sqlcode ¬= Ø | sqlrc ¬= Ø
do
call error_message
return
end
return
/*====================================================================*/
/*====================================================================*/
/* prepare and execute the SELECT statement                    */
/*====================================================================*/
do_select:
STMT='SELECT TNAME,CREATOR,TABLETYPE,REMARKS' ,
     'FROM SYSTEM.SYSCATALOG' ,
     'ORDER BY CREATOR, TNAME'
ADDRESS COMMAND ,RXSQL PREP STAT1' stmt
if rc ¬= Ø
do
call error_message
return
end
address command 'RXSQL OPEN STAT1'
if rc ≠ Ø
then
do
call error_message
return
end
orec=''
cnt=Ø
do forever
address command ,
'RXSQL FETCH STAT1 TNAME,CREATOR,TABLETYPE,REMARKS'
IF RC = 4 & SQLCODE = 100 THEN LEAVE
if rc ≠ Ø
then
do
call error_message
return
end
cnt=cnt+1
orec=orec || strip(creator)'.'tname'&'
if tabletype = 'R'
then
orec=orec || 'Table'
else
orec=orec || 'View'
orec=orec || '& ' || remarks'@
END
'output' orec
say cnt ,recordssent. Length' length(orec)
address command 'RXSQL CLOSE STAT1'
address command 'RXSQL COMMIT'
address command 'RXSQL PURGE STAT1'
return
/*====================================================================*/
/* SQL error messsage routine                                       */
/*====================================================================*/
error_message:
SAY 'ERROR'
output 'ERROR'
return

IS0002 REXX
/*====================================================================*/
/* Name          :  IS0002  REXX                                      */
/*====================================================================*/
/* Application   :  InfoServer                                      */
/*====================================================================*/
/* Usage         :  Pipeline Stage Command                           */

/* * Arguments : - */
/* Result : - */
/* Function : Request Handler for REQ2 (CMS File Grabber) */
/* */
/* InStream Ø : Request_Record */
/* OutStream Ø : Requested_Data_Stream_for_Client */
/* */
/*====================================================================*/
/*====================================================================*/
/* main processing logic */
/*====================================================================*/
/* readto request_record' */
vaddr2='1234' /* must be unused vaddr */
fmode='V'     /* must be unused fmode */
parse value request_record with req_id req_data
parse value req_data with user vaddr mname mtype
   .callpipe cp link' user vaddr vaddr2 .rr'
   .callpipe cms acc' vaddr2 fmode
   .callpipe cms listfile' mname mtype fmode , | var fileline'
/*====================================================================*/
/*====================================================================*/
/* exit if file not found */
/*====================================================================*/
if pos('NOT FOUND',translate(fileline)) > Ø then
   do
      orec=user vaddr mname mtype ,not found'
      orec=right(length(orec),8) || orec
         .output' orec
      .callpipe cms rel' fmode ,(det'
         return
   end
/*====================================================================*/
/*====================================================================*/
/* read file, prepare output stream */
/*====================================================================*/
   .callpipe <, mname mtype fmode , | stem recs.'
   orec=''
   do i=1 to recs.Ø
      rec=strip(recs.i)
      orec=orec || right(length(rec),8) || rec
   end
   .callpipe cms rel' fmode ,(det'
   .output' orec
   return

Editor's note: this article will be continued next month.

© Xephon 1999
The DG digest reader

Many VM users make use of Richard Schafer’s MailBook program to manage their mail. MailBook evolved from the popular public domain program Ricemail (also by Schafer), and is written as an XEDIT application, allowing it to be customized through user-written XEDIT macros.

One such customization greatly enhances the user’s ability to traverse electronic mailing list digests, specifically digests produced by L-Soft’s Listserv mailing-list manager. Such digests collect together all postings to the list during a given period (usually daily) and send them out as a single mailing to subscribers to the list, the collected postings being preceded by a list of topics discussed.

The collection of postings appears in the digest in the order in which they were received, rather than being rearranged to group together those with the same topic. In a digest containing many postings, this makes it difficult to follow a single topical thread. The DG digest reader, used in conjunction with MailBook, solves this problem. One simply positions the cursor on the desired topic in the topic list. Then subsequent use of the PF2 key takes the reader through all postings in the digest dealing with the selected topic.

DG requires no installation procedures. Just copy it to any accessed disk with name DG XEDIT and, after the digest is open in MailBook, type DG. To see a brief set of instructions, type ‘DG ?’. While DG is designed to find its way through Listserv digests, it can be easily modified to negotiate its way through digests from other mailing list managers by anybody reasonably proficient at writing XEDIT macros.

Further information on MailBook is available at http://www.mailbooksoftware.com and further information on Listserv can be found at http://www.lsoft.com/listserv.stm.

SOURCE CODE

/* DG XEDIT
DG is used in conjunction with the popular VM MAILBOOK mail reader
to follow the thread of a particular topic in a LISTSERV digest*/
mailing. These mailings have a numbered list of topics at the beginning of the digest, for example:

There are 6 messages totalling 262 lines in this issue.
Topics of the day:
1. Choosing a wood plane (2)
2. Wood plane
3. Polyurethane finishes (3)

the topic names being taken from the Subject: lines of the individual postings and the numbers in parentheses indicating how many postings in the digest have the indicated topic.

After opening the digest with the MAIL program, begin by typing DG on the command line. Then select a digest topic by moving the cursor to the line containing the topic description, and press PF2. The first article on the topic will be displayed.

Successive depressions of PF2 will display the remaining articles whose subject lines contain the selected topic. (The search for the topic is case-insensitive.) Once the topic is exhausted, the file will be repositioned to the topic list as it was when the topic was selected. (For your convenience, the topic will be marked as "read" by an asterisk after the topic number.) To escape a thread before exhaustion, press PF1.

Note that if you select a topic "Wood plane" say, this will also display those postings with subject line "Choosing a wood plane".

To reset PF1-PF3 to their usual functions, press PF3 or type DG EXIT */ ADDRESS XEDIT ADDRESS CMS "GLOBALV SELECT MAIL GET ACTIVE.NOTEBOOKS"
IF ACTIVE.NOTEBOOKS = "" THEN DO /* Not running MailBook. */ "MSG DG cannot be used except in conjunction with the MailBook".
"mail reader." EXIT Ø
END
globgrp = LEFT("$DG$",8)
ARG arg
IF arg = "?" THEN DO /* Display instructions. */
PARSE SOURCE . . fn ft fm .
"PIPE <" fn ft fm "; TOLAB */| DROP 1 | CONSOLE"
EXIT Ø
END
IF arg = "EXIT" THEN DO /* User pressed PF3. */ ADDRESS CMS "GLOBALV SELECT" globgrp. /* Clean up PF keys. */ "GET pf1 pf2 pf3 pfline"
IF pf1 = "" THEN "SET PF1" pf1 IF pf2 = "" THEN "SET PF2" pf2 IF pf3 = "" THEN "SET PF3" pf3
IF pfline = "" THEN DO
    "SET CTLCHAR" 'FC'X "OFF"
    "SET RESERVED" pfline
END

"CURSOR CMDDLINE"
ADDRESS CMS "GLOBALV SELECT" globgrp "PURGE"
CALL msgexit "DG terminated; PF1-PF3 restored."
END

"EXTRACT /CURSOR /CMDLINE /CURLINE /SIZE /LINE /PF1"
line = LINE.1
init = POS('FD'X,PF1.2) > Ø        /* True if DG already initialized. */
IF CURSOR.1 = CMDDLINE.2 THEN DO      /* Then cursor on command line. */
    IF init & arg = "" THEN arg = 'FD'X     /* If already initialized. */
        /* treat as if PF1. */
SELECT                                      /* Based on argument supplied, if any. */
    WHEN arg = "" THEN DO                  /* User typed command. */
        "-* FIND Subject:"                  /* Try to find digest name. */
        IF RC = Ø THEN DO
            "EXTRACT /CURLINE"
            PARSE VAR CURLINE.3 . digestname .
            digestname = digestname "Digest:  "
        END
        ELSE digestname = ""
        "-* FIND _"                          /* Try to find topic list. */
        IF RC = Ø THEN "LOCATE /1./"       /* Line containing first topic. */
        IF RC = Ø THEN DO
            "EXTRACT /LINE"
            topicbeg = LINE.1 - 1            /* Start of topic list. */
            DO UNTIL CURLINE.3 = ""           /* Find line after last topic. */
                "LOCATE ¬/./"
                IF RC = Ø THEN "EXTRACT /LINE /CURLINE"
            ELSE LEAVE
        END
        IF RC ¬= Ø THEN CALL msgexit,          /* Too bad about that. */
            "Topic list could not be found; DG not usable."
topicend = LINE.1 - 1                   /* End of topic list. */
line = topicbeg                           /* Start out with topic list on screen. */
ADDRESS CMS "GLOBALV SELECT" globgrp "PUT topicbeg topicend"
"EXTRACT /PF1 /PF2 /PF3 /CTLCHAR /RESERVED *"
pf1 = PF1.1 PF1.2; pf2 = PF2.1 PF2.2; pf3 = PF3.1 PF3.2
ADDRESS CMS "PIPE STEM RESERVED. | LOCATE / F1 =/ | VAR pfline"
PARSE VAR pfline pf1 pf2 pf12 pf13 pf14 "F1 =" "F4 =" pf1end
esc = CTLCHAR.2
IF esc = "" | POS('FC'X,esc||CTLCHAR.4)Ø,    /* Then user has */
| pf14 = "" | pf1end = ""                 /* non-std SET RESERVED lines. */
    THEN pfline = ""
ELSE DO                                     /* OK to mess with PF prompt lines. */
    ccdef = "PROTECT" pf1 pf12 pf14 pf13
    pf1beg = pf10 "YEL REV PSØ HIGH F1 =TopCList F2 ="
pflend = "Topic F3 =QuitDG "esc'FC'X" F4 ="pflend
ADDRESS CMS "GLOBALV SELECT" globgrp,
  "PUT pf1 pf2 pf3 pfline pflbeg pflend line"
"SET PF1 ONLY MACRO DG" 'FD'X
"SET PF3 ONLY MACRO DG EXIT"
"SET CTLCHAR" 'FC'X ccdef
END
CALL newtopic,
digestname|"Put cursor at topic and press PF2 to begin thread."
END
WHEN arg = 'FF'X THEN               /* PF2, but no current topic. */
  CALL msgexit "No topic currently selected."
WHEN arg = 'FE'X THEN            /* PF2; continue current thread. */
  ADDRESS CMS "GLOBALV SELECT" globgrp "GET topic start"
    /* search string, line to start searching. */
  WHEN arg = 'FD'X THEN               /* PF1; user cancelled topic. */
    CALL newtopic "Select another topic with PF2."
  OTHERWISE
    CALL msgexit "Command DG" arg "has no meaning."
END
ELSE SELECT                           /* Case cursor on command line. */
  WHEN CURSOR.3 < 1 | CURSOR.3 > SIZE.1 THEN
    CALL msgexit "Cursor not in text area."
  WHEN arg = 'FD'X THEN DO  /* User pressed PF1 while menu displayed. */
    ADDRESS CMS "GLOBALV SELECT" globgrp "GET topicbeg"
    line = topicbeg                                  /* refresh menu. */
  ADDRESS CMS "GLOBALV SELECT" globgrp "PUT line"
  CALL newtopic
END
OTHERWISE
  ":"CURSOR.3
"EXTRACT /CURLINE /LINE"       /* Pick up contents of file line. */
  topicline = LINE.1           /* Topic is now on current line. */
ADDRESS CMS "GLOBALV SELECT" globgrp "GET topicbeg topicend"
IF topicline < topicbeg | topicline > topicend THEN
  CALL newtopic "Selected line does not appear to be in topic list."
PARSE VAR CURLINE.3  a "." +2 topic
  topic = SPACE(topic)
PARSE VALUE REVERSE(topic) WITH b c
PARSE VAR b i ")" j "(" k /* Strip off any topic frequency count. */
  IF i = "" & DATATYPE(j,"W") & k = "" THEN topic = REVERSE(c)
PARSE VAR topic topic 61            /* Shorten search target enough */
    /* to avoid missing folded Subject: lines. */
  "-1 EXTRACT /LINE"
  start = LINE.1
  ":"topicline "REPLACE" OVERLAY("","" ,CURLINE.3,POS(".",CURLINE.3)+1)
    /* Mark topic selected. */
  line = topicline /* for positioning topic menu next time displayed. */
ADDRESS CMS "GLOBALV SELECT" globgrp "PUT line topic"
END
/* Get next occurrence of current topic. */

"EXTRACT /WRAP /CASE /STAY /VARBLANK /MSGMODE"
"SET WRAP OFF"
"SET CASE MIXED IGNORE"
"SET STAY OFF"
"SET VARBLANK ON"
"SET MSGMODE OFF"

:"start+2
"LOCATE" '6A'X||"Subject:"||'6A'X||"&"||'6A'X||topic||'6A'X
locrc = RC
"SET WRAP" WRAP.1
"SET CASE" CASE.1 CASE.2
"SET STAY" STAY.1
"SET VARBLANK" VARBLANK.1
"SET MSGMODE" MSGMODE.1
IF locrc ¬= Ø THEN DO
PARSE VAR topic top25 26 1 top28 29
IF topic ¬= top28 THEN top28 = top25"...
CALL newtopic,
"Topic """"top28"""" exhausted. Select another topic with PF2."
END
"-2 EXTRACT /LINE /PF2"
start = LINE.1
ADDRESS CMS "GLOBALV SELECT" globgrp "PUT start"
IF PF2.2 = "MACRO DG" 'FF'X THEN DO
"SET PF2 ONLY MACRO DG" 'FE'X
ADDRESS CMS "GLOBALV SELECT" globgrp "GET pfline pflbeg pflend"
IF pfline ¬= "" THEN
"SET RESERVED" pflbeg"Con"pflend
END
PARSE VAR topic top48 49 1 top5Ø 51
IF topic ¬= top5Ø THEN top5Ø = top48"...
CALL msgexit "Press PF2 to follow topic """"top5Ø"""".
newtopic:                       /* Come here to return to topic list. */
"SET PF2 ONLY MACRO DG" 'FF'X
ADDRESS CMS "GLOBALV SELECT" globgrp.
"GET pfline pflbeg pflend line"
IF pfline ¬= "" THEN "SET RESERVED" pflbeg"Sel"pflend
:"line
"EXTRACT /CURLINE"
"CURSOR FILE" line+1 MAX(POS(",",CURLINE.3),1)
msgexit:            /* Come here to display message, return to XEDIT. */
PARSE ARG msg
"MSG" msg
EXIT Ø

Editor’s note: readers wishing to discuss the material in this article can contact the author at bec@nysernet.org.

Ben Chi
NYSERNNet (USA) © Xephon 1999
VM users can benefit from the integration of Aonix’s System/390 Web-based reporting tool, UltraQuest Reporter, with the Web server component of Sterling Software’s VM:Webgateway, providing end users with secure access to mainframe data for optimized ad hoc reporting via a Web browser.

UltraQuest Reporter capitalizes on the Web to enable easy access to mainframe reports, while VM:Webgateway ensures end-to-end security. In addition to leveraging mainframe security, VM:Webgateway exploits Secure Sockets Layer (SSL) technology and supports emerging multi-tier encryption security standards.

UltraQuest Reporter features a two-tier architecture that eliminates the need for additional mid-tier levels of hardware and software. Users can access multiple mainframe data sources such as DB2, IMS, IDMS, VSAM, and Teradata directly with a client Java application.

For further information contact:
Aonix, 595 Market Street, 12th Floor, San Francisco, CA 94105, USA.
Tel: (415) 543 0900.
Sterling Software, 1800 Alexander Bell Drive, Reston, VA 22091, USA.
Tel: (703) 264 8000.
Sterling Software, Sterling Court, Eastworth Road, Chertsey, Surrey, KT16 8DF, UK.
Tel: (01932) 587000.

Tivoli has announced Version 3.1 of ADSM storage manager for VM/ESA, for network back-up, archiving, and disaster recovery.

Enhancements include administrative clients to control server activities, define storage management policies for workstation files, and set up schedules for automated back-up and archive services.

Other new features include support for back-up archive clients that allows users to restore or retrieve files from an ADSM server, support for backing-up and archiving files on a variety of disk and tape devices, Web browser interfaces to support remote administration and remote back-up-archive operations, and support for customized reporting and analysis via an SQL interface.

Version 3.1 also includes support for the ADSM Hierarchical Storage Management clients and the optional Server-to-Server Virtual Volumes, which provides for on-line disaster recovery, better workload balancing across multiple servers, and sharing of system resources such as large robotic tape libraries and drives.

For further information contact:
Tivoli Systems, 9442 Capitol of Texas Highway North, Arboretum Plaza One, Austin, TX 78759, USA.
Tel: (512) 436 8000.

* * *

xepohon