In this issue

3 Using parse to improve performance
13 The saga of a migration to a PC Server 500
23 FILELIST utility (FLUTIL)
39 A full screen console interface – part 16
52 VM news
Articles published in VM Update are reviewed by our panel of experts. Members of the panel include Reinhard Meyer (Germany), Philippe Taymans (Belgium), Romney White (USA), Martin Wicks (UK), and Jim Vincent (USA).

Subscriptions and back-issues
A year’s subscription to VM Update, comprising twelve monthly issues, costs £180.00 in the UK; $275.00 in the USA and Canada; £186.00 in Europe; £192.00 in Australasia and Japan; and £190.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1990 issue, are available separately to subscribers for £16.00 ($23.00) each including postage.

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.
Using parse to improve performance

In a series of two articles, the authors will look at ways of improving the performance of REXX applications. The second article in this series will examine how to handle long strings; this first article shows how parse can be used to improve performance.

MEASURING PERFORMANCE

We measured almost all the variants described in this series of articles in three environments – VM, OS/2 (with classic REXX), and Windows 95 (with OO-REXX). This is not complete: for example, on VM we could have measured compiled EXECs, and worked with OO-REXX on OS/2. However, other tests have revealed that, on OS/2, classic REXX and OO-REXX don’t differ much. Compiled EXECs on VM are usually much faster than interpreted REXX but, in most cases, the best techniques for interpreted REXX remain the best when compiled.

The times used to compare the alternatives are Virtual CPU seconds for VM and elapsed time on OS/2 and Windows. In code examples where a SAY is part of the code, we removed the SAY when measuring (terminal I/O is very heavy). To get meaningful numbers, we placed the statements to execute in a DO loop and iterated, for example, 10,000 times. The iteration factor is not important, because we have made a relative comparison of the alternatives. For example, if the first alternative takes 1.00 second, the second 1.50 seconds, and the third 0.90 seconds, we’ll say that the second alternative performs 50% worse (+50%), the third 10% better (-10%).

For these measurements, we used our CpuTime subroutine. We include it here to encourage you to measure your own experiments. Here is the routine, and an example to call it:

```/* */
parse arg n m ; if n='' then n=1000 ; if m='' then m=10
call CpuTime 'R'
do m
  t=''
do n
    t=t 'AAA' in 'BBB'
end
```
end
say CpuTime('E') 'simply append'
call CpuTime 'R'
do m
  t=''
do n
    tt='AAA' in 'BBB'
    t=t tt
  end
end
say CpuTime('E') 'with iterim var'
exit
CpuTime:
parse source OpSys .
if OpSys='CMS' then do
  if arg(1)='R' then OldVtime=c2d(substr(diag('C'),17,8))/1000000
  else return c2d(substr(diag('C'),17,8))/1000000 - oldVtime
  return OldVtime
end
else do
  parse value time('S') time('L') with secs ' ' '.' +Ø tsecs
  if arg(1)='R' then OldVtime=space(secs tsecs,Ø)
  else return space(secs tsecs,Ø)- oldVtime
  return OldVtime
end

LESSON 1 – DON’T TAKE ANYTHING FOR GRANTED
In an effort to improve performance, we often measure alternatives and then choose the best one. Not only do we then apply the best technique, but we try to remember it for daily use. We also tend to generalize our findings; however, sometimes this is wrong. For example, we often measure things on VM and assume that the best REXX code on VM will also perform well on OS/2. You’ll learn from these articles that this isn’t always the case. We also tend to assume that what is best for handling short strings is also best for long strings (wrong again, as we’ll see).

INITIALIZING MANY VARIABLES – USING PARSE
Some time ago, we found that the use of parse can improve performance. Our first example shows how one can set many variables to ‘null’ (an empty string).

Initialization of many variables is often coded as follows:
Using parse VALUE, an alternative is:

```
parse value '' with RcConnectOk RcHelpKey RcUserid,
            RcDone RcMore RcStHelp RcHelp,
            RcStorSort RcStorUser RcStDone
```

This second alternative is about 21% faster on OS/2, and 43% better on VM. There is no difference on Windows 95. You may say, “Who cares, I only do this once in my EXEC”. But, if you apply the same technique over and over again, and your EXECs are used a lot, or by many people, gains can be considerable. The parse solution also has another advantage. When you trace interactively, the trace pauses only once as opposed to 10 times for the separate statements.

Parse is so powerful that there are many other ways to use it, as you’ll see below. You can probably use parse more than you do at the moment.

THE BASICS

For those less familiar with parse, it is appropriate to explain how a basic parse works. For more information read the REXX Reference Manual. Taking a simple example:

```
parse upper arg fn ft fm rest
```

the first word of the input is placed in variable ‘fn’, the second in ‘ft’, the third in ‘fm’. Anything that follows the third word is placed in ‘rest’ (some people ignore this, and are unaware that ‘rest’ may even include some leading blanks). With parse, you can also scan for characters or strings in the input. For example:

```
parse upper arg fn ft fm rest '(' option1 options
```

Here parse works as follows. Firstly, the input is scanned for a right parenthesis; anything before it will be placed in the variables ‘fn’, ‘ft’, ‘fm’, and ‘rest’, as in the first example. Anything following the parenthesis is placed in variables ‘option1’ and ‘options’, using the same rules as explained in the first example. With parse, you can also split at specific columns, and even parse the same input multiple
times. In our last example, we extract the first two words and character 1 and character 2:

\[
\text{parse upper arg } w1 \ w2 \ . \ 0 \ \text{col1} \ 2 \ \text{col2} \ 3 \ .
\]

**Advanced use of parse**

Now that we have covered some basics, we can start with the real work – the advanced use of parse. You can often profit from an existing parse to initialize variables to ‘null’, for example the parse ARG that many EXECs use anyway. Let’s extend the parse shown above:

\[
\text{parse upper arg fn ft fm rest } \\
\text{WorkWithTrd Spec407Bug TracePipe chartid}
\]

Note that we code two quotes following ‘rest’. These two quotes are a special form of searching (as explained above), but searching for ‘nothing’ has been defined as ‘go to the end of the input’. Without these quotes, ‘rest’ would contain the fourth word of the input and ‘WorkWithTrd’ would not become empty when the input was five words or more.

So, thanks to these two quotes, the parse works as in a previous example, but we also initialize ‘WorkWithTrd’, ‘Spec407Bug’, ‘TracePipe’, and ‘chartid’ to null. Not only can many variables be set to ‘null’, you can also assign values. Let’s first show a bad-looking example:

\[
\text{parse value 1 } \ 0 \ 1 \ 0 \ \text{'1999-01-10'} \ \text{date('B')} \ \text{with opos item,} \\
\text{voorhef MaxNbr LastTotWdDate TodayBase hist. item. ,} \\
\text{eigenaars banken MuntDate. risk. totWd.}
\]

This code has the good performance of parse, but I would never recommend using such a parse to gain speed. A bit of reformatting makes it readable:

\[
\text{parse value 1 } \ 0 \ 1 \ 0 \ \text{'1999-01-10'} \ \text{date('B')}, \\
\text{with opos item voorhef MaxNbr LastTotWdDate TodayBase . ,} \\
\text{'' hist item eigenaars banken MuntDate. risk. totWd}
\]

By aligning the variable names with the values, it becomes clearer what the variables are set to. The section ‘TodayBase . ’ ’hist’ requires some extra explanation. By coding two quotes, you tell parse to go to the end of the input string, and, as ‘hist’ follows the two quotes, it will become empty (just as the other variables following ‘hist’). By coding a dot, we make sure that ‘TodayBase’ gets only one word of the input,
and not anything that’s left; any leading or trailing space is removed as well (in the above example, this is not really required because our input has the right number of words).

On VM, the gains with parse are again considerable (17%). The PC platform is another story – on OS/2, parse is 32% heavier; on Windows 95, parse is worse too (18%). Let’s use parse to add an element to a stem with one instruction. One often encounters the following:

```plaintext
ln=Ø
   do xxx
      .....  then do /* Add new element to a stem */
      ln=ln+1       /* Increment counter         */
      lin.ln = 'stuff RcDone= RcMore='
      ln.Ø=ln      /* Put new count in "stem.Ø" */
   end
end
```

With parse, this can be done in one instruction:

```plaintext
lin.Ø=Ø
   do xxx
      .....  then /* Add new element to a stem */
      parse value lin.Ø+1 'stuff RcDone= RcMore=',
      with t       lin.t
      Ø  lin.Ø   .
   end
```

Comparing the results, on VM we gain a bit; on OS/2 we lose, but as the results are unstable we won’t publish comparisons. Both constructs can be improved – there is no need to set the ‘stem.0’ value at each iteration:

```plaintext
ln=Ø
   do xxx
      .....  then do /* Add new element to a stem */
      ln=ln+1
      lin.ln = 'stuff RcDone= RcMore='
   end
   lin.Ø=ln
end
```

The parse solution can be improved as well:

```plaintext
ln=Ø
   do xxx
```
In the table in Figure 1, you can see that VM solutions with parse win again, but the gains are not very high. Even though the gains are small, we like this parse as an elegant solution. Notice that the ‘small’ improvement of not setting the ‘stem.0’ element at each iteration leads to a gain of about 20%.

INTERMEDIATE CONCLUSIONS

On VM, using parse as shown in these examples is always better than assigning variables one by one. On OS/2 and Windows 95 the story is different – parse is not always faster. We think that REXX on a PC needs more time to build strings – and where strings become input to parse, performance degrades. The longer that string is, the more degradation.

PARSING RECORDS

Many programmers still use the SUBSTR function to parse records in pieces. For example:

```plaintext
name = substr(record,1,10) /* name */
number = substr(record,11,10) /* number*/
account = substr(record,21,10) /* account-id */
```
Again, we can gain performance by using parse. An exact mimic of the above code would be:

```plaintext
parse var record name +10, /* name */
   number +10, /* number*/
   account +10, /* account-id */
   product +10, /* prod number */
   amount +10 /* amount to pay*/
```

But, in this case, because the fields follow each other, a shorter, and even faster, solution exists:

```plaintext
parse var record name   +10, /* name */
   number   +10, /* number*/
   account  +10, /* account-id */
   product  +10, /* prod number */
   amount   +10 /* amount to pay*/
```

A quick test revealed that parse can be more than twice as fast (except on Windows 95). Figure 2 contains the comparisons – a negative % means less CPU consumption.

Because the results are so impressive, one further alternative is added, namely parsing in words. This alternative does not, however, produce exactly the same result as the three others – the elements ‘name’, ‘number’, etc will no longer contain blanks.

Let’s have a closer look at the second and third tests. The only difference is that in the second test we are very explicit, and give a start and end for each element. The resultant overhead is much higher than
we had expected. This example should convince you not to code things that are not required.

Because parsing in words seems so much cheaper, you should try to use it when possible – and don’t forget that you can mix word parsing with column parsing. Here is an example that also illustrates that you can go back and forward in the input:

```plaintext
parse var myrecord 54 name +2Ø, /* name */
1ØØ number, /* number*/
account, /* account-id */
product, /* prod number */
. /* take first 3 words only */
41 amount +10 /* amount to pay*/
```

To complete the comparisons, we’ll compare parsing records containing words. The first solution uses the WORD() function, the second uses parse. You might expect that WORD() would cost more if more words have to be extracted. Two measurements were done – extracting five words and extracting 10 words. The code for five words follows:

```plaintext
name       =word(record,1)
number     =word(record,2)
account    =word(record,3)
product    =word(record,4)
amount     =word(record,5)
```

--- compared to ---

```plaintext
parse var record name ,
    number ,
    account ,
    product ,
    amount .
```

<table>
<thead>
<tr>
<th>Test description</th>
<th>VM</th>
<th>OS/2</th>
<th>Windows 95</th>
</tr>
</thead>
<tbody>
<tr>
<td>5 or 10 words with WORD()</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
<tr>
<td>5 words with parse</td>
<td>-75%</td>
<td>-60%</td>
<td>-62%</td>
</tr>
<tr>
<td>10 words with parse</td>
<td>-82%</td>
<td>-75%</td>
<td>-67%</td>
</tr>
</tbody>
</table>

*Figure 3: Comparison of WORD() and parse*
The results shown in Figure 3 should convince you that WORD() can be costly and parse wins with ease.

**AVOID USING WORD() AND WORDS()**

As shown in the last example, WORD() can become costly – because extracting a word means that REXX has to scan the input for spaces. The farther away the word is located in the string, the more scanning is required. The WORDS() function also needs scanning. As an example we’ll code an EXEC that shows how many files of each filetype exist on a given mini-disk. People with more than basic REXX skills might produce code as shown below (this task can be done very easily with a short PIPE command, but we want to improve your REXX skills!):

```rexx
cnt.=Ø
ftypes=''
do queued()
   parse pull . ftype .
   if wordpos(ftype,ftypes)=Ø then ftypes=ftypes ftype
   cnt.ftype = cnt.ftype+1
end
do i=1 to words(ftypes)
   ftype=word(ftypes,i)
   say ftype 'count =' cnt.ftype
end
```

Here, REXX often has to scan the string stored in ‘ftypes’ (WORDPOS), and at the end WORDS() and WORD() require more scans. We can be a bit cleverer, and code the following:

```rexx
cnt.=Ø
ftypes=''
do queued()
   parse pull . ftype .
   if cnt.ftype=Ø then ftypes=ftypes ftype
   cnt.ftype = cnt.ftype+1
end
   do while ftypes<>''
      parse var ftypes ftype ftypes
      say ftype 'count =' cnt.ftype
   end
```

We avoid the use of WORDPOS() in the first loop; in the second loop we use parse to ‘eat’ the list of filetypes.
On VM, for 203 files with 67 different filetypes, the gain was 56%. For a similar EXEC on OS/2 (253 files, 64 different extensions, and 3 iterations), the first solution costs 0.10 seconds, the second only 0.03 seconds, a gain of 70%.

The OS/2 REXX code for this exercise follows:

call SysFileTree 'E:\*','FILE.','.OF'
call time 'R'
do 3
   cnt.=Ø
   exts=''
   do f=1 to file.Ø
      parse upper var file.f '.' ext .
      if wordpos(ext,exts)=Ø then exts=exts ext
      cnt.ext = cnt.ext+1
   end
   do i=1 to words(exts)
      ext=word(exts,i)
      /*avoid terminal IO     say ext '=' cnt.ext*/
   end
end
bad=time('E')
say file.Ø words(exts)
drop cnt. exts /* restart with clean situation */
call time 'R'
do 3
   cnt.=Ø
   exts=''
   do f=1 to file.Ø
      parse upper var file.f '.' ext .
      if cnt.ext=Ø then exts=exts ext
      cnt.ext = cnt.ext+1
   end
   do while exts<>''
      parse var exts ext exts
      /*avoid terminal IO     say ext '=' cnt.ext*/
   end
end
say 'words' bad 'clever=' time('E')

Kris Buelens and Guy De Ceulaer
Advisory Systems Engineers
IBM (Belgium) © IBM (Belgium) 1999
The saga of a migration to a PC Server 500

This is a true story, but, as they say in the movies, ‘Any resemblance to people, either living or dead, is purely coincidental’.

THE URGENCY

In March 1996, our IBM vendor came to our office to present a new machine – the PC Server 500. He explained that this machine, which was the size of a ‘large’ PC Server, could advantageously replace our old mainframe, an IBM 4381 Model Release 14.

Indeed, we were running a VM on a IBM 4381 from the early 1980s. Our applications had been frozen for a few years – nothing new was developed on this platform any more because the tendency was towards ‘downsizing’. The mainframe was becoming over dimensioned for what it was running. We were also experiencing disk crashes. From mid-1995 until the end of 1996, we had more than ten disk problems, the majority of these being disk crashes! The worst case was in one week when we had two crashes, one after the other, and spent days replacing them and restoring data.

The proposal of the IBM vendor was interesting not only in terms of hardware solution, but also in terms of cost saving, because of a smaller IBM group level, which directly impacted on software licence costs and power consumption costs! It was not difficult to calculate that the return on investment for replacing our mainframe with a PC Server 500, including support for the installation, would be less than 18 months!

THE PC SERVER 500

The PC Server 500 is the size of a very big PC. It is delivered with external keyboard, mouse, and a 15 inch monitor. The standard configuration includes a CD-ROM drive.

The disk configuration was two RAID array controllers, each with a 2.3GB disk. The ‘mainframe’ part of this machine is a 390 card and two 390 channel cards, required to connect external ‘old’ mainframe equipment. In our case, these were needed to connect the following:
• 3745 communication controller.
• Local 3275 controller.
• 3780 tape controller.
• Fast IBM line printer.

Comparing the old 4381 equipment size to this new technology, I was not surprised by the size, weight, and electrical consumption comparison, shown in Figure 1.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Mainframe</th>
<th>PC Server</th>
<th>Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Weight (kg)</td>
<td>5,415</td>
<td>23</td>
<td>23.5</td>
</tr>
<tr>
<td>Volume (m3)</td>
<td>17.04</td>
<td>0.11</td>
<td>155</td>
</tr>
<tr>
<td>Ground surface (m2)</td>
<td>9.65</td>
<td>0.17</td>
<td>57</td>
</tr>
<tr>
<td>Electricity (kW/h)</td>
<td>15</td>
<td>0.3</td>
<td>50</td>
</tr>
</tbody>
</table>

*Figure 1: Comparison table*

TO BUY OR NOT TO BUY

It was obvious that migrating to the machine was a good choice. Unfortunately, because of the slow process of administration in the company, it took more than six months to have the approval to buy the PC Server 500. We were already in the fourth quarter of 1996 when the ‘go’ signal was given. I re-contacted our vendor and was (partially) surprised to hear that the model we were ready to buy had already been replaced by more recent ones. He came back the week after, presenting an RS/6000 model as the platform of choice for the 390 card! Knowing all the difficulties we had during the past six months to get approval for the PC Server 500 machine, I explained to him that restarting the entire process of approval for a different machine could lead to an endless loop – machines changing so quickly that our decision would never fit with the currently available model. This second cycle fortunately stopped because this new configuration,
using an RS/6000, was twice the price of the PC Server 500, and finally the vendor understood that it was a case of either ‘now the PC Server’ or ‘never an RS/6000’.

We had to go through a third party to purchase this machine because IBM doesn’t sell it directly. The same applied to customization and installation. As I will explain, if you encounter this situation, any problem arising later on can make life very difficult. The contract was signed two weeks afterwards – we were already in the first quarter of 1996.

THE CONTRACT IS SIGNED

In addition to the hardware assembly of the PC Server 500, the contract included the preliminary operating systems installation (OS/2 and VM/ESA), and the migration of our current VM/HPO 5 (without any major modification) on top of this VM/HPO. The vendor responsibility, as far as our environment transfer was concerned, was a little ambiguous – the contract mentioned fifteen days of system engineer’s support to have our VM/HPO and all applications operational on the PC Server 500. It was not clear what would happen if those fifteen days were not sufficient to achieve the objective, but we were optimistic about the issue and didn’t pay too many attention to these clauses.

I have to mention at this point that the supplier did not analyse in depth what our precise hardware and software configuration was before the contract was signed, especially pieces related to data communication (VTAM, etc) – they were also confident in the simplicity of the project.

PRELIMINARY INSTALLATION

The PC Server 500 was delivered with a preliminary standard OS/2 configuration, and a specialist came to our office for two days to complete the operating systems installation. Because the plan was to run VM/HPO as a guest for VM/ESA, his task was the installation and customization of VM/ESA and pre-configuration of a guest VM/HPO – in this case, simply a few VM directory definitions. He spent time explaining the philosophy of running a VM/ESA in this
hardware environment. In the meantime, the supplier contacted IBM to sub-contract to them the transfer of the VM/HPO from the old platform to the new one! I was not really surprised. At the beginning of the project (March 1996) I had a discussion with an IBM system engineer who was theoretically dedicated for that part of the job.

VM/HPO CONFIGURATION AND TRANSFER TESTS

At this stage, everything was within schedule. The IBM system engineer came on a Monday morning. His planned assignment was for one week. What I had to prepare before his arrival was a full back-up (stand-alone dump/restore, etc) of the mainframe disks, and a way to restore that data. As we planned to switch tape units several times between the mainframe and the PC Server for back-up/restore, as well as other equipment for testing (fast printer, 3745, 3174, 3274, etc), we succeeded in getting an old IBM switching unit. This allowed bus/tag switching of controllers between two channels. This operation is much simpler than having to shut down systems, disconnect bus/tag cables, reconnect them to the other system, etc. With such a switching unit, switching a tape controller takes only a few minutes, without any down time – drain it on one side, physically turn a switch, and start it on the other side, or vice versa to reverse back to the original configuration. (This old equipment weighed more than 600 pounds!) Our sub-contractor responsible for hardware was not aware that this unit was so big and heavy and went to his company warehouse to bring it back in his car. We had to rent a small truck to get it on site! After installing this unit, he did not read the operating instructions carefully and powered it on without caution. The power consumption of this device was greater than the circuit could deliver, and all computer (and other) equipment on the fourth floor went down! You can imagine how happy the 200 affected users were.

Finally, everything was ready for the IBM engineer – and for me.

For those of you – and I suppose this is the majority – who are not familiar with a PC Server configuration in relation to VM operating systems, let’s summarize the whole thing by saying that it is very simple. Every VM device has a PC Server equivalent. Between the VM device and its PC Server equivalent, an OS/2 device driver creates the interface.
We started on a Monday morning in April 1996. The IBM engineer quickly discovered that the pre-configuration of the VM/HPO was wrong. Indeed, VM disk volumes correspond to (large) OS/2 files and, when you configure such a file, you have to specify the first cylinder and last cylinder numbers. We had some single density 3380s, so cylinders go from zero to 884, and not from one to 885 as pre-configured. So we had to recreate definitions and format many volumes. The next day, the environments were ready. VM/ESA was up and running, and a VM/ESA user-id was the VH/HPO operating system. A full back-up, volume by volume, of the old VM/HPO system was performed and I stayed at the office the following night to restore those volumes on the PC Server. As we had two tape units, I was running two stand-alone restore jobs from the MAINT user-id of the VM/ESA, restoring to the VM/HPO defined and equivalent volumes. This was when I saw that the disk throughput of the PC Server 500 was almost a third of the 3380 disk throughput. The next morning, we made a few application tests from a 3270 terminal. It was OK.

The IBM engineer’s tasks were completed and on schedule! Now we had another engineer from the supplier coming to customize VTAM. This took several days. I was less involved because my VTAM knowledge is mainly VTAM operation-related. I know about simple configuration, but here the engineer had to work in a VM/HPO environment, that OS running in a VM/ESA environment. Again, everything was completed on time.

In our existing configuration, all users are running a 3270 emulator. This is the one from Novell (NetWare for SAA). In this configuration, the connection is established as follows: from his PC, the user starts a piece of software, which is the emulator. This looks for an SAA gateway. This gateway is, in fact, a simple Server running NetWare and the gateway software from Novell. That Server service is configured to dynamically distribute SNA addresses. The communication to the mainframe is via a single 3174, equipped with a network interface card. Two series of 32x3270 ports are defined on it. This corresponds to 64 addresses defined in the gateway.

Having only one 3174, so we had to wait until non-working hours to switch the 3174 and start tests from several PCs. We ran critical applications successfully. It was amazing to realize that those
‘mainframe’ applications that were running on a mainframe of several tons were now running in a 23-kilogram box. I discovered a few problems. One was that batch jobs failed. After investigation, I found that those jobs issue the CMS ‘format’ command to format their temporary A mini-disk, as defined in the VM directory. The biggest batch job needed a 100 cylinder mini-disk. Knowing that many batch jobs were regularly executed on the machine, each requiring this format, I decided to change the directory entries of the batch user identification, defining the A mini-disk as permanent. This was a good workaround for this problem. I knew that the format command, apart from the batch case, was not used a lot. I submitted the problem to the IBM engineer who came for the PC Server installation. After discussion, we came to the erroneous conclusion that the format failed because we had forgotten to CP-format the 3380 volume dedicated to temporary space.

Because this was the only Server problem encountered, and now apparently solved, we decided to do the real data migration over the following week-end. On Friday evening, I started the automated mainframe back-ups. Three of us were available during the Saturday and we each did a third of the restore. On Saturday evening, the PC Server 500 was in production mode. I had changed the VM logo with a few words confirming that the system was now running on a PC Server. On Monday morning, we were proud to say to our management that the task was completed. This was a little optimistic, as you will see.

THE ‘FORMAT’ PROBLEM ARISES AGAIN

I don’t precisely remember how the ‘format’ problem came to the surface again, but it did. The assumption that the 3380 volume was not correctly CP-formatted was wrong. Issuing a CMS format on a permanent mini-disk gave a CMS error. It took me half a day to discover that, indeed, the CMS format appeared to have successfully executed but I could not figure out why the return code was not zero. This was annoying because some functionality, not critical but important, was no longer operational.

The day after, I had a call from the IBM system engineer, who had found the problem. When a CMS format command is issued, without
specifying to which cylinder the format has to be applied, the format does not check for the last cylinder via a CP query command, but simply writes on cylinders from the first one until it gets a specific error code from CP. At that time, it knows the work is completed and OK, and returns zero. This was developed in this way because it saved a system (CP query) request. The reason the format returned a non-zero return code after migration to the PC Server was that, when trying to write beyond the last cylinder, VM/ESA gets a return code from the 3380 disk driver (interface with OS/2 disk, etc), which is not the same as the one it would get from a real 3380. So it reports a different error to VM/HPO, which, in turn, reports it to the format module. That one now returns a non-zero value to the user because it did not get the expected return code from CP.

The workaround is to issue the format command by specifying the last cylinder number as an optional parameter. If this is specified, the format stops at the cylinder that is passed as argument and, of course, does not get a error from CP, so it terminates correctly.

To implement the workaround, I wrote a format EXEC in REXX. This queries the system about the mini-disk size and itself issues the format command (when the module is launched) with the optional last cylinder parameter.

This EXEC was placed on the CMS system disk. Any piece of software that does not call the module explicitly will now go through the EXEC and, for those, the problem is definitively fixed. I noticed that a few tools in REXX issued the command ‘address CMS command’ and, later on, the format, the result being the implicit call to the module, so these EXECs had to be changed accordingly.

DATABASE CORRUPTION
The PC Server 500 ran for less that a week when, in the middle of the afternoon, the whole system stopped working. It appeared to be frozen – locked by something that we could not figure out. After a complete power down, we restarted it and it worked correctly again. The system engineer came to the office and ran some 390 channel card diagnostic programs. The machine was still running in production mode. Many users were entering data and suddenly, after the system engineer launched one of his tools on the OS/2 Server, the complete OS/2
system crashed and, with it, VM/ESA and VM/HPO. The guy was not very proud of this! And was even less so when I told him, after having restarted the system again, that a database was corrupted.

We run a large application based on a SQL/DS database (the SQL relational database for VM). Restarting the database was not a problem, but, as soon as the first user of this application launched it, SQL/DS crashed. We sent a message to all users via electronic mail, stating that the application was down until further notice. Again, it took me several hours to put my finger on the guilty data. The crash appeared to be related to the access of one specific 4KB page of data, this data pertaining to a specific database table accessed at each initialization of the application. With the assistance of a SQL/DS specialist, we succeeded in ‘repairing’ that 4KB page so that the application was available again – after almost two days of downtime.

ROLLBACK TO THE MAINFRAME

The initial problem of the frozen OS/2 system was not yet solved. Because the physical configuration of the channels was different from the original (we had only two channels on the PC Server 500 and had to reconfigure several external devices such as 3174, 3745, tape drives, system printer) we went in that direction to track the guilty parts. But the problem occurred twice more in a single day. This was too much for our user community and the system engineer was not very optimistic about fixing the problem within the coming days!

We had a status meeting and decided to rollback to the mainframe until the problem was solved. This operation cost me another day and another night! The back-up was performed on the PC Server 500. This took much longer than the equivalent on the mainframe, but time lost here was gained with the faster restore on the mainframe. The day after, everything was back to normal.

THE VTAM SOLUTION

The supplier of the PC Server 500 was contacted and given a complete description of what we knew. Of course, it was difficult to get him to agree that his company had to fix the problem, because, although the contract mentioned the objective of an operational system running on the PC Server 500, the 15 days of support for the migration were
already used, and any additional unpaid day was a net loss for them. Anyway, we succeeded in having a system engineer come to help us because, in the meantime, we were convinced that the problem originated in VTAM.

I have shortened this part of the story – in fact this process took more than a week. After many discussions with IBM to get the necessary support, the result was negative. IBM had not supported VM/HPO for some time – a fact known to all IBM customers. On the other hand, I had the feeling that the problem was related to communication between user terminals (emulator) and the system – something deriving from having too many users connected at the same time, some kind of collision having more chance happening when the VTAM traffic increases.

To demonstrate this, I wrote a REXX EXEC. This had a main loop – at each cycle, 24 random alphanumeric lines were displayed on the screen and then the screen was cleared. When the test was ready, we switched the 3174 to the PC Server 500 (after working hours) and started a dozen CMS sessions from PCs in the department. On each of them, we started the EXEC. Once the 12 EXECs were running, it took half a minute to have the system locked. We did the test several times, with fewer or more CMS sessions. We found that with more sessions the lock came faster, and with fewer it took several minutes. So my guess was correct – some kind of collision that VTAM could not handle correctly. We were happy to have found the cause of the problem, but not happy that it was VTAM, because this was the VTAM running on the VM/HPO, and that operating system had been withdrawn for many years!

Indeed, we had no support from IBM concerning this problem, because it appeared on an obsolete operating system. IBM was ready to help if we could ‘move’ the problem onto VM/ESA.

We contacted the PC Server supplier again, who agreed to send a VTAM specialist. That person stayed at our office for almost one week. He installed the VTAM for VM/ESA and the configuration of those two VTAMs, one for VM/HPO and the new one on VM/ESA, was changed in such a way that the real control of communication devices was performed by the VM/ESA. We made the same test prepared to reproduce the problem and, miraculously, the problem did
not occur any more. In fact, the VTAM on VM/ESA could handle congestion problems better than the old VTAM on VM/HPO. The resulting configuration between these two VTAMs was more complex, but it ran.

THE FINAL MIGRATION
Now customized with back-up/restore, I did the migration during one night (another one!) and in the morning the first users arrived and were able to run the application on that small PC Server 500.

SOME CONCLUSIONS
It is always dangerous to give a fixed price for an, apparently, simple fixed objective. In this case, the supplier didn’t make much profit, believe me! In fact, globally, he lost money. I remember another case where a programmer gave a price of £1,500 for migrating a business basic application from a Unix multi-station environment to a client/server environment (PCs and NetWare server). He worked hard for more than two weeks to achieve the objective.

Also, I have to acknowledge that all parties – the supplier, IBM, and ourselves – were too enthusiastic about the project. Upfront, the supplier didn’t realize that we were running a very old operating system. And during the test, we were too enthusiastic, seeing our applications running in such a small box, without any changes.

EPILOGUE
We are now three years on and my job has changed; I work for another company. However, I know these VM applications are still running – a kind of in-house developed ERP. However, it will not survive 31 December 1999 – some kind of BIG Year 2000 problem. Can you imagine, VM/HPO with VTAM, SQL/DS 2.x, all this old stuff, with around 100 days to go. I hope they will replace it before then, but the last news I heard was that they were still looking for replacement packages – within 100 days of the catastrophic day.

Philippe Taymans (Belgium) © Xephon 1999
FILELIST utility (FLUTIL)

INTRODUCTION
FLUTIL provides an XEDIT prefix subcommand interface to the CMS ERASE, RENAME, and COPYFILE commands. It provides an easy way to operate with multiple files simultaneously.

FLUTIL is a powerful tool, created for use by system support staff. Power users may also simplify their work with CMS files using this handy utility.

FLUTIL is written in Assember and REXX.

FILELIST UTILITY USAGE
This FILELIST utility is written as an XEDIT macro. It is called by typing FLUTIL on the command line in the FILELIST environment. This means that FLUTIL does not work stand-alone, but executes only within FILELIST.

FLUTIL copies the list of files generated by the FILELIST EXEC as a new file in the XEDIT ring, allowing the user to use prefix subcommands, against the file names.

FLUTIL uses the following PF key settings:
• PF01 – display FLUTIL help file.
• PF02 – return to refreshed in advance FILELIST screen.
• PF03 – return to refreshed in advance FILELIST screen.
• PF04 – move to the top of FLUTIL list.
• PF05 – move to the bottom of FLUTIL list.
• PF06 – return to refreshed in advance FILELIST screen.
• PF07 – scroll back one screen on the FLUTIL list.
• PF08 – scroll forward one screen on the FLUTIL list.
• PF09 – return to refreshed in advance FILELIST screen.
• PF10 – sort FLUTIL list by filename, filetype, and filemode.
• PF11 – sort FLUTIL list by filetype, filename, and filemode.
• PF12 – return to refreshed in advance FILELIST screen.

To delete, copy, or rename a single file, type D, C, or R in the prefix area of the line that contains the target filename.

To delete, copy, or rename a group of files, type DD, CC, or RR in the prefix area of both the first and last lines with the target filenames.

If the selected group is larger than one screen, then use PF04, PF05, PF07, and PF08 to scroll to the next filename.

The prefix commands CC and RR only mark files to be copied or renamed. Target filename and filetype masks must then be entered.

The mask may contain letters from A to Z, numbers from 0 to 9, square brackets ([]), or an asterisk (*). It is defined as follows:

\[
<\text{prefix part}|[*|<\text{postfix part}>,
\]

where:

• \(<\text{prefix part}\> – \text{string }[\text{counter]} \] [\text{counter}] \text{string}
• \(<\text{postfix part}\> – \text{string }[\text{counter]} \] [\text{counter}] \text{string}
• ‘\*’ is the place holder for source filename or filetype.

The string may contain letters and numbers only, with no special characters.

Examples of the string are:

• REL101
• VER202
• BETA.

The counter is used to generate sequences. The following counters may be declared:

• Numeric counter – contains only numbers.
• Alphabetic counter – contains only letters.
• Mixed counter – contains numbers and letters.

If it contains more than one character, then the change to the right-most character determines the changes to the remaining characters in the counter. Numeric characters define the sequence 0,1,...9 and, by analogy, alphabetic characters define the sequence A,B,...Z.

Square brackets are used to declare different counters. Only one counter may be defined for two masks. It can be located in filename mask or in filetape mask, but not in both.

Examples of simple counters are shown in Figure 1.

<table>
<thead>
<tr>
<th>Counter</th>
<th>Description</th>
<th>Generated sequence</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric [3]</td>
<td></td>
<td>3, 4, ...9</td>
</tr>
<tr>
<td>Numeric [00]</td>
<td></td>
<td>00, 01, ... 99</td>
</tr>
<tr>
<td>Alphabetical [D]</td>
<td></td>
<td>D, E, ... , Z</td>
</tr>
<tr>
<td>Mixed [2A4]</td>
<td></td>
<td>2A4, 2A5, ... 9Z9</td>
</tr>
</tbody>
</table>

Figure 1: Examples of simple counters

For RENAME tasks, the target filemode is not entered but is copied from the original filemode.

For COPYFILE tasks, an additional ‘Replace’ field is displayed. If this is not blank, then the REPLACE option is set in all generated CMS COPYFILE commands.

You should note that names generated by declared masks are truncated from the right to a length of eight characters. Figure 2 shows examples of masks that may be used to copy or rename files.
<table>
<thead>
<tr>
<th>FNs</th>
<th>FTs</th>
<th>FN mask</th>
<th>FT mask</th>
<th>target FNs</th>
<th>target FTs</th>
</tr>
</thead>
<tbody>
<tr>
<td>A/AA</td>
<td>B/BB</td>
<td>*</td>
<td>*</td>
<td>A/AA</td>
<td>B/BB</td>
</tr>
<tr>
<td>PLI/PLI</td>
<td>001/002</td>
<td>*VER1</td>
<td>*</td>
<td>PLIVER1/PLIVER1</td>
<td>001/002</td>
</tr>
<tr>
<td>A/B</td>
<td>ASM/ASM</td>
<td>*ASM[1]</td>
<td>ASSEMBLE</td>
<td>ASM1/BASM2</td>
<td>ASSEMBLE/ASSEMBLE</td>
</tr>
<tr>
<td>AA/BB/CC</td>
<td>BB/BB/BB</td>
<td>NUM[00]</td>
<td>REL*</td>
<td>NUM00/NUM01/NUM02</td>
<td>RELBB/RELBB/RELBB</td>
</tr>
<tr>
<td>A/B/C</td>
<td>A/B/C</td>
<td>V*1</td>
<td>N[7]*MOD</td>
<td>VA1/VA1/VC1</td>
<td>N7AMOD/NBBMOD/N9CMOD</td>
</tr>
<tr>
<td>1/2/3</td>
<td>A/A/A</td>
<td>NUM</td>
<td>MOD*[C]</td>
<td>NUM/NUM/NUM</td>
<td>MODAC/MODAD/MODAE</td>
</tr>
</tbody>
</table>

Figure 2: Masks used to copy or rename files

FLUTILIN ASSEMBLE

**********************************************************************
****FLTUTILIN get file list in ****
****
**********************************************************************

FLUTILIN CSECT
USING *,12
LR 11,14
PACK DOUBLE(8),32(5,1)
CVB Ø,DDOUBLE
LTR Ø,Ø
BZ RET
MVC XDCB+8(8),8(1)
MVC XDCB+16(8),16(1)
MVC XDCB+24(2),24(1)
LA 1Ø,XDCB
USING FSCBD,1Ø
MVI FSCBFV,C'F'
ST Ø,FSCBANIT
LR 1,Ø
MH 0,=H'10'
MH 1,=H'80'
ST 1,FSCBSIZE
LR 2,Ø
DMSFREE DWORDS=(Ø)
ST 1,FSCBBUFF
MVC FSCBCOMM(8),=CL8'DMSXFLRD'
LA Ø,EXTPLIST
LA 1,XDCB
INSTALL EXEC

/**************************** ******************************************************/
/****                         ***        ***        ***/  
/**** INSTALL generate FLUTIL MODULE ***        ***        ***/  
/****                         ***        ***        ***/  
/**************************** ******************************************************/

HI = '1DF8'X
LO = '1DF0'X
CLRSCRN
DO 11
   SAY
END

MESSAGE = 'user request'

SAY': Start FLUTILIN MODULE generation - reply Y or N

HI TIME(L)LO

PULL REPLY

IF REPLY ¬= 'Y' THEN

SIGNAL ERROR

SET CMSTYPE HT

STATE FLUTILIN MODULE A

SAVE_RC = RC

SET CMSTYPE RT

IF SAVE_RC = Ø THEN

DO

SAY ':— FLUTILIN MODULE found on disk A

HI TIME(L)LO

SAY ':— Replace FLUTILIN MODULE A - reply Y or N

HI TIME(L)LO

PULL REPLY

IF REPLY ¬= 'Y' THEN

SIGNAL ERROR

END

SET CMSTYPE HT

SIGNAL ON ERROR

MESSAGE = 'error when assemble' FLUTILIN

ASSEMBLE FLUTILIN

ERASE FLUTILIN LISTING A

MESSAGE = 'error when load' FLUTILIN

LOAD FLUTILIN '()' NOMAP NOLIBE

MESSAGE = 'error when genmod' FLUTILIN

GENMOD

ERASE FLUTILIN TEXT A

SIGNAL OFF ERROR

SET CMSTYPE RT

SAY ':— FLUTILIN MODULE generated successfully

HI TIME(L)LO

SET CMSTYPE HT

STATE FLUTIL HELP A

SAVE_RC = RC

SET CMSTYPE RT

MESSAGE = 'user request'

IF SAVE_RC = Ø THEN

DO

SAY ':— FLUTIL HELP found on disk A

HI TIME(L)LO

SAY ':— Replace FLUTIL HELP A - reply Y or N

HI TIME(L)LO

PULL REPLY

IF REPLY ¬= 'Y' THEN

SIGNAL ERROR

END

BUF.1 = '— PFKEYs'

BUF.2 = '    2,3,6,9 & 12 - exit'

BUF.3 = '           4 & 5 - top & bottom'

BUF.4 = '           7 & 8 - up & down'

BUF.5 = '         10 & 11 - sort by name & sort by type'
BUF.6 = '— prefix macro'
BUF.7 = ' C & CC - copy'
BUF.8 = ' D & DD - erase'
BUF.9 = ' R & RR - rename'
BUF.10 = '— examples of parameters if * is replacement for N -'
BUF.11 = ' FN FT masks FM result FN FT FM'
BUF.12 = ' ——————————————————— ———————————————————'
BUF.13 = ' 1* 1 A 1N 1 A'
BUF.14 = ' * *2 A N N2 A'
BUF.15 = '[1| * A 1 N A, 2 N A,..,9 N A'
BUF.16 = ' 1 [1B2| A 1 1B2 A, 1 1B3 A,..,1 9Z9 A'
BUF.17 = ' 1[A]*B 1*2 A 1AAB 1N2 A, 1BAB 1N2 A,..,1ZAB 1N2 A'
BUF.18 = ' ';
BUF.19 = 'Note: One counter may be processed for filename - it may be'
BUF.20 = ' declared in FN mask or in FT mask only. Prefixes and'
BUF.21 = ' postfixes may be combined without limitations.'
EXECIO 21 DISKW FLUTIL HELP A '(STE BUF.'
SAY '— FLUTIL HELP generated successfully'HI TIME(L)LO
EXIT
ERROR:
SET CMSTYPE RT
SAY '— FLUTILIN MODULE not generated due to' MESSAGE HI TIME(L)LO

FLUTILQQ XEDIT

/*********************************************************************/
/***                                                  ***         ***/
/*** FLUTIQQ     FILELIST util qq                     ***         ***/
/***                                                  ***         ***/
/*********************************************************************/
QUIT
LRECL 1Ø8
RECFM V
TOP
REST
SOS PFØ2

FLUTILHL XEDIT

/*********************************************************************/
/***                                                  ***         ***/
/*** FLUTILHL    FILELIST util help                    ***         ***/
/***                                                  ***         ***/
/*********************************************************************/

DO I = 1 TO 12
PF || I ONLY QQUIT
END
ENT ONLY QQUIT
CMD OFF
CURL ON 3
CURS SCR 24 8Ø
MSGL OFF
NUM OFF
PRE NULL
SCAL OFF
SER OFF
STAY ON
TOFEOF OFF
RESER 1 H '- Help -' COPIES(' ', 35)
    '*** Filelist Utility ***'
RESER 2 H COPIES(' ', 47)
    '***** Ver 1.Ø (C) DG''99 *****'
':1

FLUTILPR XEDIT

/**************************************************************************/ 
/**                                                  **/ 
/*** FLUTILPR          filelist util profile          **/ 
/**                                                  **/ 
***************************************************************************/
LINEND OFF
ENT IGNORE CURS HOME
CASE MIXED
CMDLINE BOT
CURLINE ON M
FULL ON
HEX ON
MSGLINE OFF
MSGM OFF
NONDISP ''
NUMB ON
NULLS ON
PREF NULL
TOFEOF OFF
SCALE OFF
SERIAL OFF
SHADOW OFF
STAY ON
ZONE 1 '*'
PRE SYN C FLUTILCR
PRE SYN CC FLUTILCR
FLUTILIN XEDIT

/********************************************************************/
/***                                                  ***         ***/
/*** FLUTILIN         FILELIST util user input        ***         ***/
/***                                                  ***         ***/
********************************************************************/

CTL '+' PROTECT
CTL '-' NOPROTECT H
CTL '|' ESCAPE
PULL LINE_NO OPER .
IF LEFT(OPER, 1) = 'C' THEN
  DO
    OPER = 'COPYFILE'
    TXT = 'Copied'
    ERR_TXT = 'Bad RC after copyfile -'
  END
ELSE
  DO
    OPER = 'RENAME'
    TXT = 'Renamed'
    ERR_TXT = 'Bad RC after rename -'
  END
N_TO_PROC = QUEUED()
DO I = 1 TO N_TO_PROC
  PULL ORIG_ID.I.1 ORIG_ID.I.2 VOL .
END
DO WHILE J < 4
  CMD OFF
  IF LEFT(OPER, 1) = 'C' THEN

RESER 24 N 'Enter target FN|__________|+FT|__________|+FM|_|+'.
    | 'Replace|_|+''
ELSE
RESER 24 N 'Enter target FN|__________|+FT|__________|+'
CURS SCR 24 17
READ N N
RESER 24 OFF
CMD BOT
IF LEFT(OPER, 1) = 'R' THEN
QUEUE VOL
DO J = Ø TO QUEUED() - 1
    PULL ID.J
END
END
IF J < 5 THEN
REP = ''
ELSE
REP = '(REP'
DO I = 1 TO 2
    J = INDEX(ID.I, '_') - 1
    IF J < Ø THEN
        J = 1Ø
    CHECK_NAME.I = SUBSTR(ID.I, 1, J)
END
NEW_VOL = ID.3
CHAR = 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
NUM  = XRANGE('Ø', '9')
IF VERIFY(CHECK_NAME.1 || CHECK_NAME.2,
    CHAR || NUM || '*[|') ¬= Ø THEN
DO
    MSGM ON
    EMSG '— Invalid chars in filename'
    MSGM OFF
    EXIT
END
COUNT_SEQ = Ø
DO ID = 1 TO 2
    J = Ø
    START_S = 1
    DO I = 1 TO LENGTH(CHECK_NAME.ID)
        IF SUBSTR(CHECK_NAME.ID, I, 1) = '*' THEN
            DO
                IF I - START_S > Ø THEN
                    DO
                        J = J + 1
                        NEW.ID.J = SUBSTR(CHECK_NAME.ID, START_S, I - START_S)
                    END
                J = J + 1
                NEW.ID.J = '*'
            END
        END
    END
DO
    J = J + 1
    NEW.ID.J = '*'
END

ELSE
  DO
    DO I = 1 TO 26
      CHAR.I = SUBSTR(CHAR, I, 1)
    END
    DO I = 0 TO 9
      NUM.I = SUBSTR(NUM, I + 1, 1)
    END
    STR = NEW.COUNT_ID.COUNT_SEQ
    DO I = 1 TO LENGTH(STR)
      GEN.I = SUBSTR(STR, I, 1)
      TYP.I = LEFT(DATATYPE(GEN.I), 1)
      IF TYP.I = 'N' THEN
        DO J = 0 TO 9
          IF GEN.I = NUM.J THEN
            DO
              GEN.I = J
              LEAVE
            END
          END
        ELSE
          DO J = 1 TO 26
            IF GEN.I = CHAR.J THEN
              DO
                GEN.I = J
                LEAVE
              END
            END
          END
        END
      END
    END
  END
END
CL ':'22
DO I = 1 TO N_TO_PROC
  IF I > 1 THEN
    DO J = LENGTH(STR) TO 1 BY -1
      IF J = LENGTH(STR) THEN
        DO
          START = 0
          GEN.J = GEN.J + 1
          IF TYP.J = 'N' THEN
            IF GEN.J > 9 THEN
              DO
                GEN.J = 0
                START = 1
              END
            IF TYP.J = 'C' THEN
              IF GEN.J > 26 THEN
                DO
                  GEN.J = 1
                  START = 1
                END
              END
            END
          END
        END
      END
    END
  END
END
ELSE
DO
  IF START = 1 THEN
  GEN.J = GEN.J + 1
  START = Ø
  IF TYP.J = 'N' THEN
    IF GEN.J > 9 THEN
      DO
        GEN.J = Ø
        START = 1
      END
    IF TYP.J = 'C' THEN
      IF GEN.J > 26 THEN
        DO
          GEN.J = 1
          START = 1
        END
      END
    END
  END
END

COUNT = ''
DO K = 1 TO LENGTH(STR)
  L = GEN.K
  IF TYP.K = 'N' THEN
    COUNT = COUNT || NUM.L
  ELSE
    COUNT = COUNT || CHAR.L
  END
END
NEW.COUNT_ID.COUNT_SEQ = COUNT
DO ID = 1 TO 2
  NEW_ID.ID = ''
  IF J_PARSE.ID = Ø THEN
    NEW_ID.ID = ORIG_ID.I.ID
  ELSE
    DO J = 1 TO J_PARSE.ID
      IF NEW.ID.J = '*' THEN
        NEW_ID.ID = NEW_ID.ID || ORIG_ID.I.ID
      ELSE
        NEW_ID.ID = NEW_ID.ID || NEW.ID.J
      END
    END
  END
END
GEN_ID = LEFT(NEW_ID.1, 8) LEFT(NEW_ID.2, 8) NEW_VOL
':LINE_NO + I - 1
ADDRESS CMS OPER ORIG_ID.I.1 ORIG_ID.I.2 VOL GEN_ID REP
IF RC = Ø THEN
  CR CENTRE(TXT 'as' GEN_ID || REP 'at' TIME(), 52, '-')
ELSE
  CR CENTRE(ERR_TXT RC 'at' TIME(), 52, '-')
END
FLUTILD XEDIT

 /********************************************************************
 /** FLUTILD FILELIST util delete */
 /********************************************************************

 ARG . LINE_NO N
 PARSE SOURCE . . . CMD
 EXT'/LI'
 CL ':'22
 IF CMD = 'D' THEN
   DO
     ': LINE_NO
     STACK 1 1 2Ø
     PULL F_N
     ERASE F_N
     IF RC = Ø THEN
       CR CENTRE('Deleted at' TIME(), 52, '-')
     ELSE
       CR CENTRE('Bad RC after delete' RC 'at' TIME(), 52, '-')
   END
 ELSE
   IF CMD = 'DD' THEN
     DO
       EXT'/PEND DD'
       ':: LINE_NO
       IF PENDING.Ø = Ø THEN
         PEND BLOCK DD
       ELSE
         PEND OFF
         EXT'/LI/PEND DD' LINE_NO + 1
       ':: PENDING.1
       PEND OFF
     DO LINE = LINE_NO TO PENDING.1
     ':: LINE
     STACK 1 1 2Ø
     PULL F_N
     ERASE F_N
     IF RC = Ø THEN
       CR CENTRE('Deleted at' TIME(), 52, '-')
     ELSE
       CR CENTRE('Bad RC after delete' RC 'at' TIME(), 52, '-')
 Americana XEDIT

FLICTULCR XEDIT

/*****************************/
/* *** FLUTILCR FILELIST util copy & rename ***
/* *** FLUTILCR FILELIST util copy & rename ***
/*****************************/

ARG . . LINE_NO N .
PARSE SOURCE . . . . CMD .
EXT'/LI'
IF LEFT(CMD, 1) = 'C' THEN
TXT = 'copied'
ELSE
TXT = 'renamed'
IF CMD = 'C' |
CMD = 'R' THEN
DO
':LINE_NO
STACK 1 1 2Ø
PUSH LINE_NO CMD
PUSH FLUTILIN
CL ':.22
CR CENTRE('To be' TXT 'at' TIME(), 52, '-')
END
ELSE
IF CMD = 'CC' |
CMD = 'RR' THEN
DO
EXT'/PEND' CMD
':LINE_NO
IF PENDING.Ø = Ø THEN
PEND BLOCK CMD
ELSE
DO
PEND OFF
EXT'/LI/PEND' CMD LINE_NO + 1
':PENDING.1
PEND OFF
CL ':.22
DO LINE = LINE_NO TO PENDING.1
':LINE
STACK 1 1 2Ø
FLUTIL XEDIT

 /***************************************************************************
 /****                                                     **********
 /**** FLUTIL               FILELIST util                  **********
 /****                                                     **********
 /***************************************************************************
 /****          SIZE 0017  VER 1.0 MOD 000                  **********
 /*************************************************************************/

 PRES
 LRECL 80
 RECFM F
 SET CMSTYPE HT
 ADDRESS CMS ERASE FLISTIN FILELIST A
 EXT '/SIZ/FN/FT/FM'
 XEDIT FLISTIN FILELIST A '(' W 80 PROF FLUTILPR
 ADDRESS CMS FLUTILIN FNAME.1 FTYPE.1 FMODE.1 RIGHT(SIZE.1, 5, '0')
 TOP

FLUTIL PREPARATION

The INSTALL EXEC, FLUTIL, generates executable code and the FLUTIL help file.

The FILELIST utility may then be started by typing FLUTIL in the command line of the FILELIST screen.

Please note that this will work only for the read/write A mini-disk, so, if the behaviour of FLUTIL is odd, you should check your A disk status.

Dobrin Goranov
Information Services Co (Bulgaria)  © Dobrin Goranov 1999
A full screen console interface – part 16

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.

CSCTMR ASSEMBLE

This module performs no function at present; however, it should eventually support time-based events. It already contains some date/time conversion routines.

This module is not part of the CSC Service Program. Under VM/ESA Version 2, it can be installed as a CP Exit to allow some DSMs that use the CP MSG as a command interface to be controlled by CSC. This is because of the way SCIF and the *MSG System Service were implemented. Check CP Programming Services (*MSG), the CSCMGX comments, and possibly APAR VM49400 for details on this VM/ESA feature.

```
TITLE 'CSCTMR - CSC Date and Time Management Routines'
CSCTMR START X'01B438'
PRINT NOGEN
CSCHDR                        Time Management

* * Process Time Based Events
*
*
USING TMRSECT,R6              TMR (Event) Block
SPACE

* * Validate Time Based Event (TMR) entry
*
*
Input R1 Addresses entry to validate
* Output R1 Addresses entry validated
* If entry is valid it is added to Events list and a zero cc is returned
* If entry is not valid a non-zero cc is returned
```
* * 
LR R6,R1 Address TMR entry
TM TMROPT1,TMRNAME Was NAME specified?
BZ TMR5ØØ No, some bad news
TM TMROPT1,TMROCMD Is COMMAND missing?
BZ TMR6ØØ Yes, more bad news
TM TMROPT2,TMRODATE Date specified
BZ TMR1ØØ No, check something else
LA R1,TMRDATE Address DATE
BAS R14,ASTRO Convert to astronomical
BAS R14,CALENDAR Convert back to calendar
C R1,TMRDATE Is it the same?
BE TMR1ØØ Yes, that was a good date
LA R1,TMRDATE Address DATE
B TMR8ØØ No, that was a bad date

SPACE

TMR1ØØ TM TMROPT2,TMROFROM Test FROM option
BZ TMR2ØØ
LA R1,TMRFROM
BAS R14,ASTRO
BAS R14,CALENDAR
C R1,TMRFROM
BE TMR2ØØ
LA R1,TMRFROM
B TMR8ØØ

SPACE

TMR2ØØ TM TMROPT2,TMRTO Test TO option
BZ TMR3ØØ
LA R1,TMRTO
BAS R14,ASTRO
BAS R14,CALENDAR
C R1,TMRTO
BE TMR3ØØ
LA R1,TMRTO
B TMR8ØØ

SPACE

TMR3ØØ TM TMROPT2,TMROTIME Test TIME option
BZ TMR31Ø
LA R1,TMRTIME
BAS R14,TIME
BNZ TMR81Ø Not valid, display error message

TMR31Ø TM TMROPT2,TMROFRST Test FIRST option
BZ TMR32Ø
LA R1,TMRFIRST
BAS R14,TIME
BNZ TMR81Ø

TMR32Ø TM TMROPT2,TMROFRST Test LAST option
BZ TMR33Ø
LA R1,TMRLAST
BAS R14,TIME
BNZ TMR810
TMR33Ø TM TMROPT2,TMROINT Test INTERVAL option
B2 TMR36Ø
LA R1,TMRINT
BAS R14,TIMEMMSS Validate minutes/seconds only
BNZ TMR810
TMR36Ø L R1,TMRPTR
ST R1,TMRFWD
ST R6,TMRPTR
SR R15,R15 Generate zero cc
TMR9ØØ LR R1,R6 Copy entry address to R1
LTR R15,R15 Generate appropriate cc
BACK SPACE
TMR5ØØ MSG Ø8ØØ Missing NAME option
B TMR9ØØ SPACE
TMR6ØØ MSG Ø8Ø1 Missing COMMAND option
B TMR9ØØ SPACE
TMR8ØØ LH RØ,.Ø(,R1) Load year
CVD RØ,TMRCONV Convert to decimal
OI TMRCONV+7,X'ØF' Remove sign
UNPK BADDATE,TMRCONV Convert to character
SR RØ,RØ Required by next IC
IC RØ,2(.R1) Load month
CVD RØ,TMRCONV
OI TMRCONV+7,X'ØF'
UNPK BADMM,TMRCONV
IC RØ,3(.R1) Load days
CVD RØ,TMRCONV
OI TMRCONV+7,X'ØF'
UNPK BADDD,TMRCONV
LA R2,BADDAT Address date for message
MSG Ø8Ø2 Invalid date
B TMR9ØØ SPACE
TMR81Ø LH RØ,.Ø(,R1) Load hours
CVD RØ,TMRCONV Convert to decimal
OI TMRCONV+7,X'ØF' Remove sign
UNPK BADMIN,TMRCONV Convert to character
SR RØ,RØ Required by next IC
IC RØ,2(.R1) Load minutes
CVD RØ,TMRCONV
OI TMRCONV+7,X'ØF'
UNPK BADSEC,TMRCONV
IC RØ,3(.R1) Load seconds
CVD RØ,TMRCONV
OI TMRCONV+7,X'ØF'
UNPK BADSEC,TMRCONV

LA R2,BADTIME Address time for message
LA R1,3

TMR82Ø
CLI Ø(R2),C'Ø' Remove leading zeros
BNE TMR83Ø
LA R2,l(.R2)
BCT R1,TMR82Ø Maximum of three zeros

TMR83Ø
MSG Ø8Ø3 Invalid time
B TMR9ØØ

SPACE 3

* Convert next date (TMRNDATE) and next time (TMRNTIME) to TOD format
* Output TMRCC contains date/time
*
TODBUILD EQU * Convert date/time to TOD format
LA RØ,DIAGØØØØ Address work area for DIAG ØØØØ
LA R1,L'DIAGØØØØ Length of work area
DIAG RØ,R1,X'ØØØØ' Do it to get current zone value
IH RØ,TMRNDYY Load year (yyyy)
S RØ,YEARZERO Subtract TOD clock origin (19ØØ)
LR R2,RØ Copy to work register
SRDL RØ,32 Expand to double register pair
M RØ,Y2D Multiply by 365
SR R3,R3 Required by next IC
IC R3,TMRNDMM Load month (mm)
C R3,TWO Before February?
BH TOD1ØØ

BCTR R2,Ø Yes, decrement one

TOD1ØØ SRL R2,2 Divide year by four
AR R1,R2 Add days for leap years
BCTR R3,Ø Number of full months
SLL R3,2 Prepare to index table
L R2,M2D(R3) Get total of days
AR R1,R2 Accumulate
SR R2,R2 Required by next IC
IC R2,TMRNDDD Load day of the month (dd)
BCTR R2,Ø Decrement
AR R1,R2 Accumulate -> R1 = days
M RØ,D2H Convert days to hours
IC R2,TMRNTHH Load hour (hh)
AR R1,R2 Accumulate -> R1 = hours
M RØ,H2M Convert hours to minutes
IC R2,TMRNTMM Load minutes (mm)
AR R1,R2 Accumulate -> R1 = minutes
M RØ,M2M Convert to micro seconds
IC R2,TMRNTSS Load seconds (ss)
S R2,TMRZONE Adjust current zone
SRDL R2,32 Expand to register pair
M R2,S2M Convert to micro seconds
**ALR** R1,R3  
**BNO** TOD2ØØ  
**LA** R2,1(,R2)  
**AR** R0,R2  
**SLD** R0,12  
**STM** R0,R1,TMRCC  
**BR** R14  
**SPACE 3**

* Convert calendar date yyyy/mm/dd into astronomical date
* Input R1 addresses calendar date (TMR format)
* Output R1 contains astronomical date
* Comments  - formula is taken from:
  * 'Calculus astronomiques a l'usage des amateurs'
  * Jean Meeus, Societe Astronomique de France
  * Paris, 1986. - page 18
* Note  - Adapted from VM UPDATE issue 25
  * September 1988
* 

**ASTRO** EQU *  
**LH** R4,0(,R1)  
**SR** R3,R3  
**IC** R3,2(,R1)  
**SR** R2,R2  
**IC** R2,3(,R1)  
**C** R3,TWO  
**BH** AST1ØØ  
**BCTR** R4,0  
**LA** R3,12(,R3)  
**LR** R1,R4  
**M** R0,K1  
**D** R0,HUNDRED  
**AR** R1,R2  
**LA** R3,1(,R3)  
**M** R2,K2  
**D** R2,TRUNC  
**AR** R1,R3  
**A** R1,K3  
**LR** R3,R4  
**SR** R2,R2  
**D** R2,HUNDRED  
**SR** R1,R3  
**SRL** R3,2  
**AR** R1,R3  
**BR** R14  
**SPACE 3**

* Convert calendar to astronomic
  * yyyy = yyyy - 1
  * mm = mm + 12
  * astro = trunc(yyyy * 365.25) +
  * dd +
  * trunc((mm + 1) * 30.6001) +
  * 1720997 -
  * trunc(yyyy / 100) +
  * trunc(yyyy / 400)
* Convert astronomical date into calendar date yyyy/mm/dd
* 
* Input R1 contains astronomical date
* Output R1 contains calendar date (TMR format)
* 
* Comments - formula is taken from:
* 'Calculus astronomicus a l'usage des amateurs'
* Jean Meeus, Societe Astronomique de France
* Paris, 1986. - page 18
* 
* Note - Adapted from VM UPDATE issue 25
* September 1988
* 
CALENDAR EQU * Convert astronomic to calendar
LR  R2,R1                  k = trunc(astro -
M  RØ,HUNDRED            1867216.25) /
S  R1, K4                     36524.25
D  RØ,K5                     36524.25
LA  R2,1(R1,R2)                a = astro + 1 + k -
SRL  R1,2                         trunc(k / 4) +
SR  R2,R1                     1524
A  R2,K6                         Save R2 = a
LR  R1,R2                     b = trunc((a - 122.1) / 365.25)
M  RØ,HUNDRED                   
S  R1,K7
D  RØ,K1
LR  R3,R1                         Save R3 = b
M  RØ,K1                        c = trunc(365.25 * b)
D  RØ,HUNDRED                   
LR  R4,R1                         Save R4 = c
LR  R1,R2                         d = trunc((a - c) / 30.6001)
SR  R1,R4
M  RØ,TRUNC                     
D  RØ,K2
LR  R5,R1                         Save R5 = d
M  RØ,K2                        dd = a - c - trunc(30.6001 * d)
D  RØ,TRUNC
SR  R2,R1
SR  R2,R4                         Save R2 = dd
BCTR  R5,0                        Save R5 = mm = d - 1
S  R3,K8                         Save R3 = yyyy = b - 4715
C  R5,TWELVE                     If mm > 12 Then
BNH  CAL1ØØ
S  R5,TWELVE                     mm = mm - 12
CAL1ØØ  C  R5, TWELVE             If mm > 2 Then
BNH  CAL2ØØ
BCTR  R3,0                      yyyy = yyyy - 1
CAL2ØØ  LR  RØ,R2                  Start with dd
SRDL  RØ,8                        Shift dd to R1
LR  RØ,R5
SRDL R0,8 Shift mm to R1
LR R0, R3
SRDL R0,16 Calendar day in R1 (yyyymmdd)
BR R14
SPACE 3

* Validate time
*
* Input R1 Addresses time (TMR format)
* Output A zero cc is returned if time is valid
* Otherwise a non-zero cc is returned
* TIMEHHSS is invoked to validate Minutes and Seconds only
*
* TIME EQU * Validate time
LH R0,0(.R1)
C R0,MAXHH
BH TIME900
TIMEMMSS CLI 2(R1),MAXMM
BH TIME900
CLI 3(R1),MAXSS
BH TIME900
CR R14,R14
TIME900 BR R14
SPACE 3
TMRCOVN DS D
DIAG0000 DS CL40 Buffer for DIAG X'0000'
TMRZONE EQU DIAG0000+32,4 System zone
SPACE
YEARZERO DC F'1900'
Y2D DC F'365'
D2H DC F'24'
H2M DC F'60'
M2M DC F'60000000'
S2M DC F'1000000'
M2D DC F'000,031,059,090,120,151,181,212,243,273,304,334'
SPACE
K1 DC F'36525' Ask the astronomers please...
K2 DC F'306001' Constants used by ASTRO
K3 DC F'1720997' and CALENDAR
K4 DC F'186721625'
K5 DC F'3652425'
K6 DC F'1524'
K7 DC F'122190'
K8 DC F'4715'
TWELVE DC F'12'
HUNDRED DC F'100'
TRUNC DC F'10000'
SPACE
MAXHH DC F'23'
MAXMM EQU 59
MAXSS EQU 59
SPACE
BADDATE DS CL4
   DC C'/'
BADMM DS CL2
   DC C'/'
BADDY DS CL2
   DC C'/'
SPACE
BADTIME DS CL5
   DC C':'
BADMIN DS CL2
   DC C':'
BADSEC DS CL2
   DC C'
   SPACE 3
   CSCDATA
   CSCDS TMR
   REGEQU
END

CSCMGX ASSEMBLE

TITLE 'CSCMGX - CSC CP Exit 121Ø'
MACRO
&LABEL CSCMDLAT &EPNAME
&LABEL MDLATHDR &EPNAME
MDLATENT CSCMGX,MODATTR=(PAG,MP,DYN),CPXLOAD=YES
MDLATTLR
MEND
SPACE 3

*———————————————————————————————————————————————————————————————————— *
*                                                                     *
*   CSCMGX                                                            *
*                                                                     *
* Entry Point Name - CSCMGXCK                                         *
*                                                                     *
* Descriptive name - CP Exit 121Ø                                     *
*                                                                     *
* Function         - Change CP MESSAGE into CP SMSG                  *
*                                                                     *
* Register use     - RØ  - CP exit number                            *
*                    R1  - Address of the parameter list               *
*                    R2  - Address of CP exit call request block        *
*                    R3  - Scratch                                   *
*                    - Message control flags                         *
*                    R4  - Scratch                                   *
* Message type (version)
R5 - Scratch
* Address of receiving user-id
R6 - Scratch
R7 - Scratch
R8 - Scratch
R9 - Scratch
R10 - Scratch
R11 - Dispatched VMDBK address
* - Destination VMDBK address
R12 - Base register
R13 - Save Area address
R14 - Work register, linkage
R15 - Work register, linkage

* Exit normal
- R15 = Ø
* Exit error
- None
* Function
- A message will be changed into a special message *(CP SMSG)* when all the following conditions are met:

- The CP MESSAGE command was issued.
- The message target user-id is different from the originating user-id
- is running disconnected
- has a secondary user defined
- has a valid path to *MSG or *MSGALL
- has issued a CP SET MSG IUCV command
- is authorized to receive Special Messages

* General comments
- Use the following CP commands to activate

```
CPXLOAD CSCMGX TEXT * MP NOCONTROL PERMANENT
ASSOCIATE EXIT 121Ø EPNAME CSCMGXCK
ENABLE EXIT 121Ø
```

```
PRINT NOGEN
SPACE
HPCMPID COMPID=CSC Define component ID
COPY HCOPTNS
CSCMGX HCPPROLG ATTR=(PAGEABLE,REENTERABLE,INDIRECTCALL),
COPYRID=' Copyright CSC Inc.',
COPYR=(1997),
BASE=(R12)
SPACE 3
HCPEXTRN HCPFNDUS Locate VMDBK for given userid
```
SPACE 3
COPY HCPPFXPG Host Prefix Page
COPY HCPVMDBK Virtual Machine Definition Block
COPY HCPSAVBK Savearea Block
COPY HCPEQUAT General equates
COPY HCPEQXIT CP Exit equates
COPY HCPPLXIT CP Exit PLIST definitions
SPACE 3
HCPUSING PFXPG,R0
HCPUSING X12I0,R1
HCPUSING VMDBK,R11
HCPUSING SAVBK,R13
SPACE 3
CSCMGXCK HCPENTER CALL,SAVE=DYNAMIC Define entry point
C R0,EXT1210 Is it the right exit?
BNE MGX900 No, forget it
L R3,X12I0CFG Address message control flags
L R4,X12I0MVR Check message type (version)
TM Ø(R4),EXTMSG Is it a CP MESSAGE?
BZ MGX900 No, forget it
L R5,X12I0RID Address receiving user-id
CLC VMDUSER,Ø(R5) Sending a message to yourself?
BE MGX900 Yes, forget
LA R0,L'VMDUSER Length of receiving user-id
L R1,X12I0RID Address of receiving user-id
HCPCALL HCPFNDUS Locate VMD block
BNZ MGX900 It is gone, forget it
LR R11,R1 Check receiving user-id
TM VMDOSTAT,VMDDISC Is it running disconnected?
BZ MGX900 No, forget it
CLC VMDSECU,BLANKS Any secondary user-id defined?
BE MGX900 No, forget it
TM VMDMSSFL,VMDMSSVP+VMDMSAVP Connected to *MSG or *MSGALL?
BZ MGX900 No, forget it
TM VMDMIUCV,VMDMSGIU Is MSG set to IUCV?
BZ MGX900 No, forget it
TM VMDMLVL,VMDSPMSG Can we receive SMSGS?
BZ MGX900 No, forget it
TM VMDMIUCV,VMDMSSGI Is SMSG also set to IUCV?
B0 MGX800 Yes, finally some good news
TM VMDVMCF,VMDVAUTH Are we authorized for VMCF
BZ MGX900 No, too bad...
MGX800 MVI Ø(R3),X'ØØ' Reset all message flags
MVI Ø(R4),EXTMSMG Change MESSAGE to SMSG
MGX900 SR R15,R15 Set return code zero
ST R15,SAVER15 HCPEXIT EP=(CSCMGXCK),SETCC=NO
SPACE 3
DS Ø0
BLANKS DC C' ' A few blanks and spaces
ON-LINE HELP
To access the CSC on-line help, press PF01 from a CSC session or enter HELP CSC for a list of commands.

CSC HELPMENU
.cm VM Software Services
.cm

[]CSC Tool[%
A file may be selected for viewing by placing the cursor under any character of the file wanted and pressing the ENTER key or the PF1 key. A MENU file is indicated when a name is preceded by an asterisk (*). A TASK file is indicated when a name is preceded by a colon (:). For a description of the HELP operands and options, type HELP HELP.

*CSCCFG
*CSCSVP
*CSCUSR
*NEWS
:CSC
CSC_SVP
CSC_USR
CSCSVP
CSCUSR

CSC HELPTASK
.cm VM Software Services
.cm

[]CSC Tool[%
Move the cursor to the task that you want, then press the ENTER key
or the PF1 key.

NEWS menu List the New additions to CHECK

CSC CSCUSR Display user screen
CSCCFG menu Creating a configuration file

CSC HELPABBR

<table>
<thead>
<tr>
<th>Command</th>
<th>Description</th>
<th>Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>BACKWARD</td>
<td>Backward</td>
<td>1</td>
</tr>
<tr>
<td>BACKWARD</td>
<td>BWD</td>
<td>3</td>
</tr>
<tr>
<td>BOTTOM</td>
<td>BOTtom</td>
<td>3</td>
</tr>
<tr>
<td>CLEAR</td>
<td>Clear</td>
<td>1</td>
</tr>
<tr>
<td>CONNECT</td>
<td>Connect</td>
<td>2</td>
</tr>
<tr>
<td>CURRENT</td>
<td>CURrent</td>
<td>3</td>
</tr>
<tr>
<td>DISCONN</td>
<td>DISconnect</td>
<td>2</td>
</tr>
<tr>
<td>DLOCATE</td>
<td>DLocate</td>
<td>2</td>
</tr>
<tr>
<td>DLOCATE</td>
<td>DOWNLocate</td>
<td>5</td>
</tr>
<tr>
<td>DMATCH</td>
<td>DMatch</td>
<td>2</td>
</tr>
<tr>
<td>DMATCH</td>
<td>DOWNMatch</td>
<td>5</td>
</tr>
<tr>
<td>DOWN</td>
<td>Down</td>
<td>1</td>
</tr>
<tr>
<td>DOWN</td>
<td>Next</td>
<td>1</td>
</tr>
<tr>
<td>EXCLUDE</td>
<td>Exclude</td>
<td>1</td>
</tr>
<tr>
<td>FORWARD</td>
<td>Forward</td>
<td>1</td>
</tr>
<tr>
<td>FORWARD</td>
<td>FWD</td>
<td>3</td>
</tr>
<tr>
<td>GO</td>
<td>Go</td>
<td>1</td>
</tr>
<tr>
<td>HELP</td>
<td>Help</td>
<td>1</td>
</tr>
<tr>
<td>INCLUDE</td>
<td>Include</td>
<td>1</td>
</tr>
<tr>
<td>LOCATE</td>
<td>Locate</td>
<td>1</td>
</tr>
<tr>
<td>LOCATE</td>
<td>/</td>
<td>1</td>
</tr>
<tr>
<td>MATCH</td>
<td>Match</td>
<td>1</td>
</tr>
<tr>
<td>MATCH</td>
<td>\</td>
<td>1</td>
</tr>
<tr>
<td>PRINT</td>
<td>Print</td>
<td>1</td>
</tr>
<tr>
<td>REDSPLY</td>
<td>&amp;</td>
<td>1</td>
</tr>
<tr>
<td>RELEASE</td>
<td>Release</td>
<td>1</td>
</tr>
<tr>
<td>REPEAT</td>
<td>=</td>
<td>1</td>
</tr>
<tr>
<td>RETRIEVE</td>
<td>?</td>
<td>1</td>
</tr>
<tr>
<td>SHIFT</td>
<td>SHift</td>
<td>2</td>
</tr>
<tr>
<td>SWITCH</td>
<td>Swap</td>
<td>1</td>
</tr>
<tr>
<td>SWITCH</td>
<td>Switch</td>
<td>1</td>
</tr>
<tr>
<td>TOP</td>
<td>Top</td>
<td>1</td>
</tr>
<tr>
<td>UP</td>
<td>Up</td>
<td>1</td>
</tr>
<tr>
<td>WRITE</td>
<td>Write</td>
<td>1</td>
</tr>
</tbody>
</table>

CSCUSR HELPMENU

.cm VM Software Services
.cm
.mt CSC
A file may be selected for viewing by placing the cursor under any character of the file wanted and pressing the ENTER key or the PF1 key. A MENU file is indicated when a name is preceded by an asterisk (*). A TASK file is indicated when a name is preceded by a colon (:). For a description of the HELP operands and options, type HELP HELP.

&
/?
=
Backward
BOTTtom
BWD
Clear
CMS
COntect
CURrent
Disconnect
DLocate
DMatch
Down
DOWNLocate
DOWNMatch
END
Exclude
Forward
FWD
Go
Help
Include
Locate
Match
Next
OP
Print
Release
\SHift
Swap
Switch
Top
Up
Write

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

Fernando Duarte  
Analyst (Canada)  
© F Duarte 1999
IBM has announced a new family of OSA-Express open systems adapters for 100MBps Fast Ethernet and 155MBps ATM that will have LPAR support, virtual IP addressing for VM/ESA and OS/390, IP multicast support for OS/390, and RMF enhancements. Another feature will be auto-negotiation between 10/100MBps, shared, switched, half, or full duplex operation and IPX support through Novell Network Services for OS/390 Release 1.

OSA-Express 155 ATM features include 155MBps over single mode and multimode fibre, ATM Forum-compliant LAN Emulation, SNA/APPN load balancing through Token Ring LANE, native ATM and APPN for OS/390, and native ATM and 'classic' IP via RFC 1577 in VM/ESA and OS/390. The System/390 OSA support facility in VM/ESA and OS/390 includes Windows 95 and NT support, plus a GUI or REXX EXEC interface. Running on G5 and G6 servers, a maximum of 12 OSA-Express can be installed on any one box.

For further information contact your local IBM representative.

* * *

VM users can benefit from Web390, Information Builders’ HTTP/HTTPS server, which transforms an IBM System/390 into a World Wide Web application and data server capable of supporting Internet and intranet activities of new and existing System/390 users.

Web390 provides a secure gateway to existing 3270-based applications running under VM/ESA, MVS/TSO, or CICS, giving Web browser users immediate access to the legacy applications and data. Users have a choice of screen presentation modes, to customize the look and feel of the application, and can also be given protected access to all datasets allocated in the System/390 environment.

Application developers can write CGI programs in traditional System/390 languages to accept and process input from Web browsers. Existing data access code, written in COBOL, PL/I, or Assembler, can be reused to publish operational data directly on the Web. The Web390 server supports all of the popular Web file formats (MIME types), including HTML, VRML, GIF, and JPEG graphic files, as well as MPEG animation files, JavaScript, and Java applets. All Web-based content is hosted in standard mini-disk files or SFS directories on VM, and standard PDS, PDSE, and flat files in the MVS environment.

Web390’s OpenCGI supports the use of CGI scripts written in REXX, server-side JavaScripts, or 3GL languages such as C++, Assembler, PL/I, and COBOL for accessing native System/390 data.

For further information contact:
Information Builders, 1250 Broadway, 30th Floor, New York, NY 10001, USA.
Tel: (212) 736 4433.
Information Builders (UK), Wembley Point, Harrow Road, Wembley, Middlesex, HA9 6DE, UK.
Tel: (0181) 982 4700.

* * *