190

MVS update

July 2002

In this issue

MVS Update

Published by

Xephon
27-35 London Road
Newbury
Berkshire RG14 1JL
England
Telephone: 01635 38342
From USA: 01144 1635 38342
E-mail: trevore@xephon.com

North American office

Xephon
PO Box 350100
Westminster, CO 80035-0100
USA
Telephone: 303 410 9344

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1999 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

MVS Update on-line

Code from *MVS Update*, and complete issues in Acrobat PDF format, can be downloaded from our Web site at http://www.xephon .com/mvs; you will need to supply a word from the printed issue.

Editor

Trevor Eddolls

Contributions

When Xephon is given copyright, articles published in *CICS Update* are paid for at the rate of £170 ($260) per 1000 words and £100 ($160) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of £50 ($80) per 100 lines. In addition, there is a flat fee of £30 ($50) per article. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon. com/nfc.

On the way to a mainframe Google

The mainframe community can surely give some great ideas to the Web development community – like reliability, hierarchical thinking, simplicity of thinking; on the other hand, it can accept ideas like user-friendliness, integrating all possible resources, and – above all – setting unlimited goals.

Willie van Tilburg presented a simple but good command-line tool in Issue 198 of *MVS Update*, January 1999. The command XF (a REXX function) scans a PDS for a string, and shows the members (where the string has been found) in a popup-window with the possible selection of V(iew) or E(dit).

It gave me some ideas for a general search engine – something like Google on the Web.

We have basically two problems, namely:

- What to search in.
- How to present the hits.

I don't want to aim too high with the scope of searchable targets; I think things like DB2 or IMS DBs are out of the question. We can focus on datasets with PS/PO/VSAM organization, and use generic name specifications. It can be a most important question with queries like:

```
==> seek mystring userid.ispf.*
```

This will find all occurrences of 'mystring' in 'userid.ispf.cntl', 'userid.ispf.data', etc. A generic search for datasets can be a most unsatisfying job, but there are jewels like CATSRCH from Mark Zelden and others to lighten it. VSAM datasets can be 'flattened' into PSs before the searching.

The presentation of the hits is a most important topic. I propose – as a first step – an extension of XF from van Tilburg on this. Also presented is a technique to include a panel inside the REXX procedure (no more PANELLIB necessary).

The hits are presented like this:

```
command ===> xf cntl.freigabe vsam w        scroll === Row 1 to 12 of 12
member     found

 #REHDM1  //* INPUT FILE:  SEQ/VSAM/DB2/IMS DATEN + //* INPUT FILE:

 BREØ2Ø3Ø //*+BREØ2Ø3Ø JOBB ==> LADEN ANWENDUNGS-DATEI IN TEMPORARY V

 BREØ2Ø5Ø //*+BREØ2Ø5Ø JOBB ==> LADEN PERSONEN-DATEI IN TEMPORARY VSA

 BREØ5Ø14 //EINGABE  DD DSN=SYSR.NDVSYS.VSAM.KVSDD,

 BREØ5Ø4Ø //*+BREØ5Ø4Ø JOBB ==> LADEN SEQL. DATEI (ALLE SD-S) IN VSAM
 BREØ5Ø5Ø //*+BREØ5Ø5Ø JOBB ==> LADEN SEQL. DATEI (SD-TO-PRG) IN VSAM
 BREØ526Ø //*+BREØ526Ø JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
 BREØ527Ø //*+BREØ527Ø JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
 BREØ528Ø //*+BREØ528Ø JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
 BREØ5282 //*+BREØ5282 JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
 BREØ529Ø //*+BREØ529Ø JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
 BREØ53ØØ //*+BREØ53ØØ JOBB ==> VSAM-DATEI MIT ROCHADE-INTERNE DSNAME
**************************** Bottom of data *****************************


THESE ARE THE MEMBERS WITH >VSAM<
```

As you can see, the hits are presented with a '+' to indicate multiple occurrences of the search string in the target. It's important to know that it has been created with standard components like ISP, REXX, and SRCHFOR (= 3.14).

Let me list some possible improvements:

- To include a macro to point to the searched string when viewing/ editing the selected member (but personally I don't like the need for a further external component).
- To enable users to see all the hits for a member by scrolling (it's an everlasting problem – no scrolling in Table Services, see Doug Nadel's Web site http://www.sillysot.com/mvs/ for a solution).
- To enable the use of a high-level qualifier instead of dataset name (see above).
- To sort the hits by date/time of the last change, etc.
- A further (but interesting) possibility could be the searching in ALL DATASETS in the target dataset(s); that means, searching for a string in all datasets defined in the 1.Parameter (it must, of course, be all JCL).

- To use regular expressions (it’s not impossible: there’s a GREP for the host by courtesy of http://www.dignus.com/).

XF REXX

```
/******************************** REXX *******************************/
/*  TSO COMMAND  XF                                                   */
/*  Description :                                                     */
/*  Find members in a PDS with search argument like in ISPF 3.14      */
/*  The members will be shown in a selection panel with the 'hits'    */
/*  ie the text found in the member. You can select these by          */
/*  'E(dit)' or x (any other selection = View; it can be easily       */
/*  re-programmed)                                                    */
/*  The Parameter DD (=DSN) can be defined with wildcard(s) = *,      */
/*  asterisk.                                                         */
/*  ----------+--------------------------------------------+--------  */
/*  Parameter |                Description                 | Default  */
/*  ----------+--------------------------------------------+--------  */
/*  parameter | parameter-Description                      |          */
/*  ----------+--------------------------------------------+--------  */
/*  DD        | dataset-name with/without apostrophes      | must-be  */
/*            | '?' = this HELP                            |          */
/*  SE1       | search-argument                            | must-be  */
/*  W         | if = 'W' then search word                  | no word  */
/*  ----------+--------------------------------------------+--------  */
/*  Warnings                                                          */
/*  1) Please change the message #PRSC001 into your local LMSG        */
/*********************************************************************/
arg dd se1 w
"ISPEXEC VGET ##NAME SHARED"
if ##name <> 'XF' /* I am now a TSO ISPEXEC Command */
then
do
  ##name = 'XF'
  "ISPEXEC VPUT ##NAME SHARED"
  ##se1  = se1
  "ISPEXEC VPUT ##SE1 SHARED"
end
else /* I am now an EDIT macro */
do
  "ISPEXEC VGET ##SE1 SHARED"
  "ISREDIT MACRO PROCESS"
  "ISREDIT HILITE FIND"
  "ISREDIT F "##se1" ALL"
  "ISPEXEC VERASE ##NAME SHARED"
  return
end
"ISPEXEC VERASE ##NAME SHARED"
/*                       Help wanted ?                              */
if dd = '?'
```

```
then
do
  do i = 3 to sourceline() ,
     until (substr(sourceline(i),15,9) = 'changed :')
    say substr(sourceline(i),3,68)
  end
  exit Ø
end
/* Inits & check Parameter                                            */
if substr(dd,1,1) ¬= "'"
then do
  dd = "'" || userid() || '.' || dd || "'"
end
call msg(off)
e=listdsi(dd)
if sysreason > Ø & pos('*',dd) = Ø then
   do
   e=listdsi(se1)
   if sysreason > Ø then
      do
        lmsg = 'no valid dataset and only 1 search argument allowed'
        "ISPEXEC setmsg msg(#prscØØ1)"
        exit
      end
      else
        do
          se2 = se1
          se1 = dd
          dd  = se2
        end
   end
dsname=substr(dd,2,length(dd)-2)
if pos('*',dsname) > Ø
then do
  call CATSRCM dsname
  do i = 1 to dsns.Ø
    parse var dsns.i prefix '.' postfix
    if prefix = 'DSNS' & datatype(postfix) = 'NUM'
    then
      nop
    else
      dsname = dsns.i
  end
end
if se1 = ''                           /* no search argument = EXIT */
then do
  lmsg = 'use a search argument'
  "ISPEXEC SETMSG MSG(#PRSCØØ1)"
  exit
end
/* Mainline                                                           */
```

```
                                   /* allocate the nessecary datasets */
                                   /*for ispf standard search program */
address TSO
'FREE FI(NEWDD,OUTDD,SYSIN)'
"ALLOC FI(NEWDD) SHR DA('"DSNAME"')"
'ALLOC FI(OUTDD) NEW DSORG(PS) REC(F B) LR(133) BLK(13300)' ,
     'SPACE(2,2) TRACKS DA(XF.LIJST)'
'ALLOC FI(SYSIN) DELETE DSORG(PS) REC(F B) LR(80) BLK(3120)' ,
     'SPACE(1,2) TRACKS'
cmd = 'SRCHFOR' "'"se1"'"
if w ¬= ''
then
  cmd = cmd || ',w'
queue cmd
'EXECIO 1 DISKW SYSIN (FINIS'
                                             /* issue the ispf search */
address ISPEXEC
'SELECT PGM(ISRSUPC) PARM(SRCHCMP,ANYC,NOSEQ)'
             /* free the datasets and read the results into a buffer */
address TSO
'FREE FI(NEWDD,OUTDD,SYSIN)'
'ALLOC FI(OUTDD) SHR DA(XF.LIJST) DELETE'
'EXECIO * DISKR OUTDD (FINIS'
'FREE FI(OUTDD)'
               /* read the buffer and put valid members into a table */
n       = 0
member. = ''
foundt. = ''
do queued()
   pull regel
   if pos('STRING(S) FOUND',regel) > 0
   then do
     if substr(regel,3,1) = ' ' then iterate
     if substr(regel,3,11) = 'LINES-FOUND' then iterate
     if substr(regel,3,11) = 'MEMBER-SEAR' then iterate
     if substr(regel,3,11) = 'PROCESS OPT' then iterate
     if substr(regel,3,11) = 'THE FOLLOWI' then iterate
     if n > 0
     then
       foundt.n = founds
     n        = n+1
     member.n = substr(regel,2,9)
     founds   = ''
  end
  else do
    parse var regel line# text
    if datatype(line#) = 'NUM' & ,
       pos('COMPARE UTILITY',regel) = 0
    then do
      text = strip(text)
      if length(founds) < 1000
      then
        if founds = ''
        then
          founds = text
        else
          founds = founds || ' + ' || text
    end
  end
end
foundt.n = founds
                                  /* check if there are members found */
if member.1='' then do
   lmsg = 'there are no members with' se1
   "ISPEXEC SETMSG MSG(#PRSC001)"
   exit
end
lmsg = 'SearchArgument>'se1'<'
"ISPEXEC SETMSG MSG(#PRSC001)"
/* put the table into a ispf table */
address ISPEXEC
"TBCREATE MEMSEL NAMES(MEMBER FOUND) NOWRITE REPLACE"
do x=1 to 99999
   if member.x=''
   then
     leave
   member = strip(member.x)
   found  = foundt.x
   'tbadd memsel'
end
                                            /* display the memberlist */
'TBTOP MEMSEL'
zwinttl = 'Select Hits from 'dsname         /* Title of t.POPUP-Window*/
"ADDPOP ROW(1) COLUMN(9)"
call READ_PANEL
"TBDISPL MEMSEL PANEL(XFPANEL)"
/* select line command E or B,S...(these are equal for View)        */
PANEL_ACTION:
if reply='END'
then
  exit
if ztdsels=0
then
  "TBDISPL MEMSEL"
if ztdsels=1 then do
  "CONTROL DISPLAY SAVE"
  select
    when t = 'E' then
      "EDIT DATASET('"DSNAME"("MEMBER")')"
    when t = 'S' then do
      ##name = 'XF'
      "ISPEXEC VPUT ##NAME SHARED"
      "EDIT DATASET('"dsname"("member")') MACRO(XF)"
    end
    otherwise
      "VIEW DATASET('"DSNAME"("MEMBER")')"
  end /*EndSelect*/
  "CONTROL DISPLAY RESTORE"
  "TBDISPL MEMSEL"
end
if ztdsels > 1 then
  do until ztdsels=1
     member=strip(member)
     "CONTROL DISPLAY SAVE"
      select
        when t = 'E' then
          "EDIT DATASET('"dsname"("member")') MACRO(XF)"
        when t = 'S' then do
          ##name = 'XF'
          "ISPEXEC VPUT ##NAME SHARED"
          "EDIT DATASET('"dsname"("member")') MACRO(XF)"
        end
        otherwise
          "VIEW DATASET('"dsname"("member")')"
      end /*EndSelect*/
     "CONTROL DISPLAY RESTORE"
     "TBDISPL MEMSEL"
  end
signal PANEL_ACTION
address TSO
'FREE F($UPDPAN)'
address ISPEXEC
'LIBDEF ISPPLIB '
exit Ø
/* END of mainline                                                   */
/* internal functions                                                */
/* read built-in panel                                               */
READ_PANEL:
  unitname = 'VIO'    /* change this if allocations fail  */
  address TSO
  'ALLOC NEW DEL F($UPDPAN) DSO(PO) DIR(1) SP(3,3) TRACK
        REUSE RECFM(F B) BLKSIZE(Ø) LRECL(8Ø) UNIT('UNITNAME')'
  a = 1
  do until substr(line,1,7)='/*panel'
    line = sourceline(a)
    a=a+1
  end
  parse var line . panelname .
  address ISPEXEC
  'LMINIT DATAID(TMPPNL) ENQ(EXCLU) DDNAME($UPDPAN)'
  'LMOPEN DATAID('TMPPNL') OPTION(OUTPUT)'
  do until substr(line,1,4)=')end'
    line = sourceline(a)
    'LMPUT DATAID(&TMPPNL) MODE(INVAR) DATALOC(LINE) DATALEN(80)'
    a=a+1
  end
  'LMMADD DATAID(&TMPPNL) MEMBER('PANELNAME')'
  'LMFREE DATAID(&TMPPNL)'
  'LIBDEF ISPPLIB LIBRARY ID($UPDPAN) STACK'
return
/* search catalog for datasets with wildcards                         */
CATSRCM: procedure expose dsns.
parse upper arg key cat            /*                                */
if key = '' then do                /*                                */
  say 'enter data set name filter' /*                                */
  pull key                         /*                                */
end                                /*                                */
count = 0                          /* total entries found            */
modrsnrc = substr(' ',1,4)         /*   clear module/return/reason  */
csifiltk = substr(key,1,44)        /*   move filter key into list   */
if cat <> '' then                  /*                                */
  csicatnm = substr(cat,1,44)      /*   use specified catalog       */
else                               /*                                */
  csicatnm = substr(' ',1,44)      /*   clear catalog name          */
csiresnm = substr(' ',1,44)        /*   clear resume name           */
csidtyps = substr(' ',1,16)        /*   clear entry types           */
csicldi  = substr('Y',1,1)         /*   indicate data and index     */
csiresum = substr(' ',1,1)         /*   clear resume flag           */
if cat <> '' then                  /*                                */
  csis1cat = substr('Y',1,1)       /* search only 1 cat              */
else                               /*                                */
  csis1cat = substr(' ',1,1)       /* search > 1 catalogs            */
csiresrv = substr(' ',1,1)         /*   clear reserve character     */
csinumen = '0001'x                 /*   init number of fields       */
csifld1    = substr('VOLSER',1,8)  /*   init field 1 for volsers    */
 /*  build the selection criteria fields part of parameter list      */
csiopts  = csicldi || csiresum || csis1cat || csiresrv
csifield = csifiltk || csicatnm || csiresnm || csidtyps || csiopts
csifield = csifield || csinumen || csifld1
 /*  initialize and build work are output part of parameter list     */
/* worklen = 1024  */
/* dwork = '00000400'x || copies('00'x,worklen-4) */
worklen = 131072  /* 128k */      /* apar ow39593 */
dwork = '00020000'x || copies('00'x,worklen-4)   /* apar ow39593 */
 /*  initialize work variables                                       */
resume = 'Y'
prevname = ''                    /* no previous name            @01a*/
catnamet = substr(' ',1,44)
dnamet = substr(' ',1,44)
 /*  set up loop for resume (if a resume is ncessary)                */
do while resume = 'Y'
 /*  issue link to catalog generic filter interface                  */
 address LINKPGM 'IGGCSI00  MODRSNRC  CSIFIELD  DWORK'
 resume = substr(csifield,150,1)    /* get resume flag for next loop */
```

```
usedlen = c2d(substr(dwork,9,4))   /* get amount of work area used  */
pos1=15                            /* starting position             */
/*  process data returned in work area                              */
do while pos1 < usedlen            /* do until all data is processed*/
  if substr(dwork,pos1+1,1) = 'Ø'  /* if catalog, print catalog head*/
   then do
        catname=substr(dwork,pos1+2,44)
        pos1 = pos1 + 5Ø
        end
  dname = substr(dwork,pos1+2,44)  /* get entry name                */
/*  assign entry type name                                          */
  if substr(dwork,pos1+1,1) = 'C' then dtype = 'CLUSTER '
   else
     if substr(dwork,pos1+1,1) = 'D' then dtype = 'DATA    '
    else
     if substr(dwork,pos1+1,1) = 'I' then dtype = 'INDEX   '
    else
     if substr(dwork,pos1+1,1) = 'A' then dtype = 'NONVSAM '
    else
     if substr(dwork,pos1+1,1) = 'H' then dtype = 'GDS     '
    else
     if substr(dwork,pos1+1,1) = 'B' then dtype = 'GDG     '
    else
     if substr(dwork,pos1+1,1) = 'R' then dtype = 'PATH    '
    else
     if substr(dwork,pos1+1,1) = 'G' then dtype = 'AIX     '
    else
     if substr(dwork,pos1+1,1) = 'X' then dtype = 'ALIAS   '
    else
     if substr(dwork,pos1+1,1) = 'U' then dtype = 'UCAT    '
    else do   /* no entries in the catalog or unknown type */
              /* unknown type can be caused by partial     */
              /* catalog record such as an interupted      */
              /* delete or update-extend (idc11441i)       */
              /* this used to cause a loop in this code    */
     if substr(dwork,pos1+1,1) = 'Ø' then iterate  /* cat entry     */
     pos1 = pos1 + 46  /* unknown dtype   */
     pos1 = pos1 + c2d(substr(dwork,pos1,2)) /* next entry*/
     iterate
    end   /* else do */
/* moved this section of code from above so catalog name would      */
/* not print during generic hlq search if no entries were found.    */
        if catname <> catnamet then /* if resume name may already be*/
         do                         /*    printed                   */
          if found = 'true' then  queue '     '
          queue 'catalog ' catname  /* if not, print it             */
          queue '   '
          catnamet = catname
          found = 'true'
         end
/* check for error in entry returned                              */
```

```
   csieflag = substr(dwork,pos1+Ø,1) /* entry flag information*/
   if bitand(csieflag,'4Ø'x) = '4Ø'x then do
     pos1 = pos1 + 5Ø           /* header length */
     pos1 = pos1 + c2d(substr(dwork,pos1,2))
     queue 'error processing the following entry:'
     queue 'dsn  =' dname
     queue 'cat  =' catname
     queue 'type =' dtype
     iterate    /* go to next entry */
   end
 /*  have name and type, get volser info                            */
    count = count + 1  /* total entries found  */
    pos1 = pos1 + 46
    numvol = c2d(substr(dwork,pos1+4,2))/6 /* how many volsers ?     */
    pos2 = pos1+6                   /* position on data              */
    do i=1 to 3                     /* only clear 3 volser fields    */
      volser.i = substr(' ',1,6)
    end
    do i = 1 to numvol              /* move volsers to output fields */
      volser.i = substr(dwork,pos2,6)
      pos2 = pos2 + 6
    end
    queue copies(' ',8) dtype dname volser.1 volser.2 volser.3
 /*   get position of next entry                                    */
    pos1 = pos1 + c2d(substr(dwork,pos1,2))
  end
 if resume = 'Y' &,/* if we've tried this entry @Ø1a*/
  prevname = dname then             /* twice, we've got to quit  @Ø1a*/
   do                                                          /*@Ø1a*/
     queue strip(dname) 'cannot be processed with the work area size '
     queue  'provided - you must increase the work area and retry'
     leave                                                     /*@Ø1a*/
   end                                                         /*@Ø1a*/
 prevname = dname                   /* save for next iteration   @Ø1a*/
end
queue ''  /* null to end stack */
i = Ø
do queued()
 parse pull line
 i = i + 1
 if pos('VSAM',line) > Ø
 then do
   parse var line . dsn .
   dsns.i = dsn
 end
end
dsns.Ø = i
return
/* end external procs                                                */
/* here the attached panel                                           */
/*panel xfpanel
```

```
)attr default(%+_)
! type(output) intens(high) caps(on) just(left)
S type(output) intens(high) caps(on) just(left) color(red)
$ type(input) intens(low) caps(on) just(asis)
)body window(76,19) cmd(zcmd)
 %command ===> _zcmd                           %scroll ===>_amt +
 +
 in member  ...text found
)model
$t!z       Sz
+
)init
.zvars ='(member found)'
&amt = page
)reinit
&t=''
&zcmd=''
refresh(zcmd)
)proc
&reply=.resp
)end
*/
```

Gabor Markon
Systems Engineer
HVB Systems (Germany)

HSM recovery panel

If you use DFSMSHSM to back up/recover your files, you’ll find this little ISPF application very useful. It is part of a larger application I developed for HSM users.

Use it to recover a dataset from HSM back-up DASD or TAPEs, choosing between several back-up generations. You can restore a dataset, overriding the existing one or creating a copy on another volume.

Our environment is MVS/ESA or OS/390 until Release 2.10 with active DFHSM or DFSMShsm; TSO/ISPF/DM until Release 4.5.

The command to use is RECUPERA (it is a CLIST that calls ISPF services). Note: obviously, it won’t work if DFHSM isn’t running!

RECUPERA

Put the RECUPERA CLIST code in a member in a SYSPROC concatenated library (member=RECUPERA), eg ISP.UISPCLIB.

It can be used in front of the dataset name in ISPF 3.4, or directly by coding:

```
TSO RECUPERA ‘IBMUSER.MY.DATASET’ (for example)
```

This will display a panel that you can fill as you wish. It cannot be used from TSO READY (outside ISPF). So, first enter ISPF. This code works well in OS/390 2.6 and 2.10. To run in MVS/ESA, first check the input list in dataset TEMPREC. Because of new information in OS/390, the listing records may be slightly different (eg the back-up date columns may be three or four digits in length).

```
PROC 1 DATASET
/*-----------------------------------------------------------------*/
/*                                                                 */
/* RECUPERA:  RECOVER BACKED UP DATASET FROM PANEL NAPRECOV        */
/*            INPUT = DATASET (DSNAME TO BE RECOVERED).            */
/*            CLIST ASSUMES THAT PROFILE PREFIX IS ACTIVATED       */
/*            FOR YOUR TSO USERID. TO CHECK IT, ENTER:             */
/*            COMMAND ===> TSO PROFILE LIST                        */
/*            AND VERIFY ACTIVE PREFIX. OTHERWISE, SET:            */
/*            COMMAND ===> TSO PROFILE PREFIX(MYPREF)              */
/*            WHERE MYPREF IS YOUR TSO USERID.                     */
/*-----------------------------------------------------------------*/
 CONTROL NOMSG NOLIST NOFLUSH END(ENDO) NOCONLIST NOPROMPT
 DELETE  TEMPREC                              /* DEL TEMP IF EXISTS  */
 HLIST DATASET(&DATASET) BCDS ODS(TEMPREC)    /* OBTAIN BACKUP LIST  */
 ALLOC F(AL) DA(TEMPREC) SHR REU INPUT        /* ALLOC HSM LIST      */
 OPENFILE AL INPUT                            /* OPEN LIST           */
 ERROR DO                                     /* ERROR ROUTINE       */
  IF &LASTCC = 400 THEN GOTO FINEFILE         /* EOF GO TO FINEFILE  */
  RETURN                                      /*                     */
 ENDO                                         /* ERROR ROUTINE END   */
 CERCA: +
 GETFILE AL                                   /* GET A REC FROM LIST */
/*-----------------------------------------------------------------*/
/*                                                                 */
/* THIS NOTE APPLIES TO LINES 35-46.                               */
/*THE FOLLOWING CODE IS UPDATED FOR OS/390 2.6  DFHSM OUTPUT LISTING.*/
/* TO WORK CORRECTLY IN PREVIOUS VERSIONS, CHECK THE OUTPUT LISTING  */
/* IN THE 'TEMPREC' DATASET WITH CMD "COLS>" TO SEE THE POSITION OF: */
/*  BACKUP DATE      (NOW FROM COL 68 TO 75)                       */
/*  BACKUP VOLUME    (NOW FROM COL 52 TO 57)                       */
/*  GENERATION NUMBER (ONLY LAST TWO DIGITS, FROM COL 94 TO 95)       */
/* AND WHERE DIFFERENT, CHANGE THE COL NUMBERS AS NECESSARY.          */
/*--------------------------------------------------------------------*/
/*                                                                    */
/* LINE 35-38. CHECK / POSITION (BACKUP DATE) AT INPUT COLS 7Ø & 73  */
/*--------------------------------------------------------------------*/
SET BARR1=&STR(&SUBSTR(7Ø,&AL)
SET BARR2=&STR(&SUBSTR(73,&AL)
    IF &STR(&BARR1) EQ &STR(/) AND +
    &STR(&BARR2) EQ &STR(/) THEN DO
/*--------------------------------------------------------------------*/
/*                                                                    */
/* LINE 39-45: OBTAIN INFORMATION FROM TEMPREC.                       */
/*--------------------------------------------------------------------*/
SET &DATACOP=&SUBSTR(68:75,&AL)
SET &GENNUM =&SUBSTR(94:95,&AL)
SET &BACKVL =&STR(&SUBSTR(52:57,&AL)
SET TROV&GENNUM=&STR+
    ( &GENNUM   &BACKVL   &DATACOP)
IF &GENNUM=Ø THEN SET DTC=&SUBSTR(68:69,&AL)+
   &SUBSTR(71:72,&AL)&SUBSTR(74:75,&AL)
ENDO
GOTO CERCA
/*--------------------------------------------------------------------*/
/*                                                                    */
/* AFTER READING INPUT FILE "TEMPREC", IF NO BACK-UP FOUND, EXIT      */
/* WITH CODE(12). ELSE, DISPLAY THE NAPRECO PANEL FILLED WITH INFO    */
/* TAKEN FROM INPUT.                                                  */
/*--------------------------------------------------------------------*/
 FINEFILE: +
 ERROR OFF
 IF &STR(&DATACOP)=   THEN DO                /* IF NO COPIES, THEN  */
 WRITE *** NO BACKUPS FOR &DATASET ***
    EXIT CODE(12)                            /* EXIT WITH RC 12     */
                      ENDO                   /*                     */
 CLOSFILE AL                                 /* CLOSE INPUT DATASET */
 FREE F(AL)                                  /* FREE DDNAME         */
 DISL: +
 SET RIMPIAZZ = NO                           /* SET REPLACE TO NO   */
             SET &LL = &LENGTH(&DATASET)     /* SET DATASET LENGTH  */
 SET &NDATASET=&STR(&SUBSTR(2:&LL-1,&STR(&DATASET)))
 SET &NDATASET=&STR(&NDATASET..AT&DTC)
 ISPEXEC DISPLAY PANEL(NAPRECO)              /* DISPLAY PANEL       */
 IF &LASTCC = 8 THEN EXIT CODE(Ø)            /* IF PF3 PRESSED, END */
 IF &NDATASET EQ  THEN SET &NEWDS=
 ELSE SET &NEWDS=&STR(NEWNAME('&NDATASET'))
 IF &RIMPIAZZ EQ NO THEN SET &REP=
 ELSE SET &REP=&STR(REPLACE)
 IF &VOLUMER EQ  THEN DO                     /* IF NO VOLUME PASSED */
 SET &UNIT=
 SET &VOL=
                     ENDO
 ELSE DO                                         /* ELSE (VOL PASSED)   */
 SET &UNIT=&STR(U(SYSDA))
 SET &VOL=&STR(TOVOL(&VOLUMER))
    ENDO
 ISPEXEC SETMSG MSG(NAMØ35)                      /* SET ISPF MESSAGE    */
 HRECOV &DATASET GEN(&BACKGEN) &REP &NEWDS &UNIT &VOL
 EXIT CODE(Ø)                                    /* EXIT CODE ZERO      */
```

NAPRECO

This ISPF PANEL code must be added to an ISPPLIB concatenated dataset (member=NAPRECO), eg ISP.UISPPLIB.

```
)ATTR DEFAULT(%+_)
 % TYPE(TEXT) COLOR(PINK) HILITE(BLINK)
 # TYPE(TEXT) COLOR(GREEN) HILITE(BLINK)
 ! TYPE(TEXT) COLOR(GREEN)
 $ TYPE(TEXT) COLOR(PINK) HILITE(USCORE)
 & TYPE(TEXT) COLOR(TURQ)
 £ TYPE(TEXT) COLOR(GREEN)
)BODY
$                  &HSM DATASET RECOVERY$           &&ZJDATE%&ZTIME$
+
+
+ Specify recovery option for
+ dataset:£&DATASET
+  &Generation+required%===>_Z +(from Ø to 12)
+ Do you want to replace existing dataset with copy? ===>_Z  +(YES/NO)
+ If&NO+you must rename recovered dataset (without ''):
+   New dataset name%===>_NDATASET
+ To restore on a specified volume:
+   VOLUME  %===>_Z     +    (target volume)
+
+ backup copies found: u
      & GEN  VOLUME  BACKUP DATE   & GEN  VOLUME  BACKUP DATE
      £&TROVØØ                     £&TROVØ7
      £&TROVØ1                     £&TROVØ8
      £&TROVØ2                     £&TROVØ9
      £&TROVØ3                     £&TROV1Ø
      £&TROVØ4                     £&TROV11
      £&TROVØ5                     £&TROV12
      £&TROVØ6                     £&TROV13
+
+Press%Enter+to recovery,%PF3+to exit
)INIT
  .ZVARS = '(BACKGEN RIMPIAZZ VOLUMER)'
  IF (&BACKGEN = '' )      &BACKGEN = 'Ø'
  IF (&VOLUMER = '' )      &VOLUMER = ''
  IF (&RIMPIAZZ = '' )     &RIMPIAZZ = 'NO'

)PROC
  VER (&BACKGEN,RANGE,Ø,13,MSG=NAMØ33)
  VER (&RIMPIAZZ,NB,LIST,YES,NO,MSG=NAMØ34)
  IF (&RIMPIAZZ = 'NO' )
     VER (&NDATASET,NB,DSNAME)
)END
```

ISPF MESSAGE CODE

The ISPF MESSAGE CODE panel must be added to an ISPMLIB concatenated dataset (member=NAM03), eg ISP.UISPMLIB:

```
NAMØ33      'GENERATION INVALID' .ALARM=YES
'SPECIFY A NUMBER IN THE RANGE Ø-13'
NAMØ34      'SPECIFY YES OR NO' .ALARM=YES
'SPECIFY EITHER YES OR NO'
NAMØ35      'RECOVERY IN PROGRESS' .ALARM=NO
'WAIT ARC1ØØØI MSG AT END OF &DATASET RECOVERY'
```

Alberto Mungai
Senior Systems Programmer (Italy)

Interfacing Assembler programs with IBM C

Many of the exit points and APIs (Application Programming Interfaces) provided with OS/390 and its associated services are designed to be used most easily with Assembler programs. The capabilities provided within Assembler programs are limited only by the skills of the coder and the security access level that can be gained. This makes the use of Assembler code for the exits and APIs an obvious choice. That said, the ease of performing certain programming tasks could be greatly improved by using higher-level programming languages. Many of the tools and applications developed today are written in multiple programming languages where the selection of the programming language can be linked more to the task at hand than to a forced use of a particular language for all components.

Superficially, this makes logical sense; however, there are typically some idiosyncrasies in moving back and forth between language environments. This article describes the requirements for interfacing Assembler programs with IBM C coded subroutines and also using Assembler subroutine calls from IBM C main programs. Both scenarios will be demonstrated by simple, but expandable, examples.

OPTION 1 – ASSEMBLER TO C

The first situation we will consider is the use of C subroutines called from Assembler programs. One example of where this could be useful is in creating an application that will make use of the TCP/IP network. There are many C functions written for this purpose and, although there are equivalent Assembler APIs, most of the example documentation available for coding a TCP/IP network application uses coded C examples. You can leverage this example code much more easily if it is used as demonstrated. The example in this article is nowhere near as sophisticated as that just mentioned, but it will show the basics for setting up a main Assembler program calling a C subroutine.

To create the proper calling environment, I recommend that the Assembler routine be created as a Language Environment (LE)-conforming Assembler routine. This provides one main benefit – the LE environment is created once and maintained across calls.

This provides significant performance benefits if the Assembler program makes multiple calls to different C subroutines.

IBM provides a suite of Assembler macros that are used for this purpose. The CEEENTRY and CEETERM macro pair can be used to create and terminate the LE environment. Other macros include CEECAA, CEEDSA, and CEEPPA. These macros are all documented in the *OS/390 Language Environment for OS/390 & VM Programming Guide*.

The ASM2C program provided with this article shows an Assembler program that creates an LE-conforming Assembler routine that is set up to run as the main enclave in the environment. It accepts an incoming parameter character string and then issues a standard Assembler CALL macro to invoke the GETLEN C subroutine. GETLEN accepts two parameters:

- The address of the character string that was passed to the original program.

- The address of a data area used to return the length of the character string.

Obviously, more sophisticated and practical examples could be used, but I simply wanted to demonstrate the requirements for the Assembler calling routine and the target C subroutine while also showing how flexible the options are.

For our example, if the parameter value passed to the ASM2C program was PARM='ABCDEFG', the GETLEN subroutine will write two records to the SYSPRINT output DD as follows:

- Parameter data is: ABCDEFG
- Parameter length is: 7.

Upon return from the GETLEN subroutine, the ASM2C program will issue a WTO indicating the length of the parameter value as returned from GETLEN.

Once the basic requirements for calling C subroutines from Assembler programs have been mastered, it is not difficult to see the possibilities that are now available.

OPTION 2 – C TO ASSEMBLER

Once you have started to use C subroutines from your Assembler programs it will not be long before you want to do the reverse. This can be extremely beneficial, especially if you need to perform some authorized function or if there is some other operation that you are having difficulty performing in C. A practical use I have needed to deploy is the ability to sleep() in a non-POSIX capable C program (not to mention needing sleep() granularity finer than one second).

In this particular case, I have coded my own MySleep() C function that accepts two integer parameters – the number of seconds and the number of hundredths of a second that the program is to be dormant. The MySleep() function performs some data validation and builds a character string parameter value in the format HHMMSSTT that gets passed to the ASMWAIT Assembler routine. The ASMWAIT Assembler routine will perform an STIMERM wait for the specified length of time.

The program examples provided include a C2ASM main() C program that calls the MySleep() C subroutine. The MySleep() subroutine makes the appropriate call to the ASMWAIT Assembler routine. A set-up/destroy macro pair, EDCPRLG and EDCEPIL, is used in the ASMWAIT program to properly manage the incoming and outgoing environment. These macros ensure that an Assembler routine can properly participate in a C/C++ established environment.

Recommendations

It is probably good practice to create your C programs to be re-entrant and to allow for long routine names (RENT and LONGNAME compiler options). Either of these requires you to run the object code created by the compiler to also be processed by a prelink step. As of OS/390 2.10, an example C compile PROC can be located in CBC.SCBCPRC(EDCC) and an example prelink PROC can be found in CEE.SCEEPROC(EDCPL). When you have multiple C object modules that need to be combined into a single load module, you will also need to use the prelink step. Other C/C++ compile options I make frequent use of are the NOMAR option (program code can extend beyond column 72 for RECFM=FB source files) and SSCOMM option (slash slash (//) format comments are valid). See the *OS/390 C/C++ User's Guide* for information on all available compiler options.

USING C++

The comments up to now have been primarily directed to using the C/C++ compiler in C compile mode. There are some other considerations if you want to invoke the C/C++ compiler in C++ compile mode. First of all, you will require an additional CPARM value of CXX. This will cause the C/C++ compiler to be activated in C++ mode. Also, compiler options like RENT and SSCOMM should be removed because they are not valid in C++ mode.

You can determine the compiler mode by reviewing the output from the SYSCPRT DD statement of your compile output. A standard C compile will show an indicator like:

```
5647AØ1 V2 R1Ø OS/39Ø C
```

in the upper left-hand corner of each output page. If you are performing a C++ compile you will see something like:

```
     5647AØ1 V2 R1Ø OS/39Ø C++
```

in the upper left-hand corner of each output page.

You can locate a sample C++ procedure in CBC.SCBCPRC(CBCC).

For C++ compiled programs that require a prelink step, you will need to include the CEE.SCEECPP dataset in the SYSLIB DD statement of the prelink step (see the JCL example for a compile and prelink later in this article).

CONCLUSION

This article was designed to discuss Assembler to C and C to Assembler invocation set-up considerations and was not meant to be an in-depth discussion on using C/C++ (that's another article). As your experience with C increases, the options available to you will expand as well. The ability to move back and forth between programming language environments is a powerful mechanism that can be employed to create robust applications and systems programming interfaces.

JCL SAMPLES

Sample JCL to compile and prelink GETLEN

As mentioned earlier, I recommend using the C prelink program (EDCPRLK) for all of your C programs even if it is not a necessary requirement for a specific C program. Using the C prelink step allows you to create re-entrant object code and allows for the use of long routine names (greater than eight characters). It also allows you to integrate multiple independently-created C code object modules into a single object module that can be used in a linkedit step. Following is the JCL that can be used to compile and prelink the GETLEN C program provided with this article:

```
//*
//EDCCPL   PROC  INFILE=,                < INPUT ... REQUIRED
//  CREGSIZ='48M',                       < COMPILER REGION SIZE
//  CRUN=,                               < COMPILER RUNTIME OPTIONS
//  CPARM=,                              < COMPILER OPTIONS
//  CPARM2=,                             < COMPILER OPTIONS
//  CPARM3=,                             < COMPILER OPTIONS
//  LIBPRFX='CEE',                       < PREFIX FOR LIBRARY DSN
//  LNGPRFX='CBC',                       < PREFIX FOR LANGUAGE DSN
//  CLANG='EDCMSGE', < NOT USED IN THIS RELEASE. KEPT FOR COMPATIBILITY
//  DCB80='(RECFM=FB,LRECL=80,BLKSIZE=3200)',      <DCB FOR LRECL 80
//  DCB3200='(RECFM=FB,LRECL=3200,BLKSIZE=12800)', <DCB FOR LRECL 3200
// OBJFILE=,
//  TUNIT='SYSDA'                          < UNIT FOR TEMPORARY FILES
//*
//*-------------------------------------------------------------------
//*  COMPILE STEP:
//*-------------------------------------------------------------------
//COMPILE EXEC PGM=CBCDRVR,REGION=&CREGSIZ,
//    PARM=('&CRUN/&CPARM &CPARM2 &CPARM3')
//STEPLIB  DD  DSNAME=&LIBPRFX..SCEERUN,DISP=SHR
//         DD  DSNAME=&LNGPRFX..SCBCCMP,DISP=SHR
//SYSMSGS  DD  DUMMY,DSN=&LNGPRFX..SCBC3MSG(&CLANG),DISP=SHR
//SYSIN    DD  DSNAME=&INFILE,DISP=SHR
//SYSLIB   DD  DSNAME=&LIBPRFX..SCEEH.H,DISP=SHR
//         DD  DSNAME=&LIBPRFX..SCEEH.SYS.H,DISP=SHR
//SYSLIN   DD  DSNAME=&OBJFILE,DISP=SHR
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
//SYSCPRT  DD  SYSOUT=*
//SYSUT1   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB80
//SYSUT4   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB80
//SYSUT5   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT6   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT7   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT8   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),DCB=&DCB3200
//SYSUT9   DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=VB,LRECL=137,BLKSIZE=882)
//SYSUT10  DD  SYSOUT=*
//SYSUT14  DD  UNIT=&TUNIT.,SPACE=(32000,(30,30)),
//             DCB=(RECFM=FB,LRECL=3200,BLKSIZE=12800)
//        PEND
//*
//STEP1   EXEC EDCCPL,CPARM=LIST,CPARM2='RENT,NOSEARCH',
// CPARM3='NOMAR,NOOPT,LANGLVL(EXTENDED),SOURCE,LONGNAME,SSCOMM',
//         INFILE=c.source.code(GETLEN),
//         OBJFILE=object.code.pds(GETLENO)
//*
//*-------------------------------------------------------------------
//* PRE-LINKEDIT STEP:
//*-------------------------------------------------------------------
//PLKED1  EXEC PGM=EDCPRLK,PARM='UPCASE',
//    REGION=2048K
//SYSMSGS  DD  DSNAME=CEE.SCEEMSGP(EDCPMSGE),DISP=SHR
//SYSLIB   DD  DUMMY
//* USE THE FOLLOWING SYSLIB STATEMENT FOR C++
//*SYSLIB   DD  DSN=CEE.SCEECPP,DISP=SHR
//SYSOBJ   DD  DSN=object.code.pds,DISP=SHR
//SYSMOD   DD  DSN=object.code.pds(GETLEN),DISP=SHR
//SYSOUT   DD  SYSOUT=*
```

```
//SYSPRINT DD  SYSOUT=*
//SYSIN    DD  *
  INCLUDE SYSOBJ(GETLENO)
```

LINKEDIT JCL to create ASM2C load module

Once the GETLEN object module has been created and the ASM2C Assembler module has been assembled, use the following JCL to create the ASM2C load module:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT   DD    DSN=object.code.pds,DISP=SHR
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD    DSN=laod.library,DISP=SHR
//SYSLIN   DD    *
   INCLUDE OBJECT(ASM2C)
   INCLUDE OBJECT(GETLEN)
   ENTRY   ASM2C
   NAME    ASM2C(R)
```

Compile and prelink C2ASM

To compile and prelink the supplied C2ASM C program, use the sample compile and prelink JCL provided earlier. Substitute all occurrences of GETLEN with C2ASM and all occurrences of GETLENO with C2ASMO. Running the compile and prelink steps will create a C main program object module.

LINKEDIT JCL to create C2ASM load module

Once the C2ASM object module has been created and the ASMWAIT Assembler module has been assembled, use the following JCL to create the C2ASM load module:

```
//IEWL     EXEC  PGM=HEWLHØ96,PARM='XREF,LIST,MAP,RENT'
//SYSPRINT DD    SYSOUT=*
//SYSUT1   DD    UNIT=SYSDA,SPACE=(CYL,(2,1))
//OBJECT   DD    DSN=object.code.pds,DISP=SHR
//SYSLIB   DD    DSN=CEE.SCEELKED,DISP=SHR
//SYSLMOD  DD    DSN=laod.library,DISP=SHR
//SYSLIN   DD    *
   INCLUDE OBJECT(C2ASM)
   INCLUDE OBJECT(ASMWAIT)
   ENTRY   CEESTART
   NAME    C2ASM(R)
```

ASM2C ASM

The ASCII source file for the ASM2C program referenced in the article:

```
***********************************************************************
*   The ASM2C program can be used to call subroutines written in C.   *
*                                                                     *
*   The following macros are required to set up and take down the    *
*   environment that makes this operation possible:                   *
*      CEEENTRY                                                       *
*      CEETERM                                                        *
*      CEEPPA                                                         *
*      CEEDSA                                                         *
*      CEECAA                                                         *
*                                                                     *
*   The following restrictions are in effect for the CEEENTRY macro:  *
*      With MAIN=YES:                                                 *
*         BASE= can be any register R3-R11 (R11 is the default)       *
*         R12 is to the address of the CEECAA                         *
*         R13 is to the address of the CEEDSA                         *
*         PARMREG= the parameter list address (R1 is the default)     *
***********************************************************************
ASM2C    CEEENTRY PPA=MAINPPA,     ** Label of CEEPPA mapping macro  **X
               AUTO=WORKSIZE,      ** Size of DSA & local work area  **X
               MAIN=YES,           ** This rtn is main rtn in enclave**X
               EXECOPS=NO,         ** No runtime options in parms    **X
               PARMREG=R1,         ** R1 is the default parm reg     **X
               BASE=R11,           ** R11 is the default base reg    **X
               PLIST=HOST          ** Standard JCL PARM= parm list   **
***********************************************************************
         USING WORKAREA,R13        Set addressability to temp storage
***********************************************************************
         ST    R1,PARMADDR         Save incoming parm address
         LTR   R1,R1               Any parameter?
         BZ    RETURNØ4            No - set return code and exit
         L     R3,Ø(,R1)           Get parameter address
         N     R3,=X'7FFFFFFF'     Turn off x'8Ø' bit
         LTR   R3,R3               Any parameter?
         BZ    RETURNØ4            No - set return code and exit
         CLC   Ø(2,R3),=H'Ø'       Any parameter data?
         BE    RETURNØ4            No - set return code and exit
         XR    R15,R15             Clear R15
         ICM   R15,B'ØØ11',Ø(R3)   Capture parm length
         BCTR  R15,Ø               Reduce by one for EX
         EX    R15,PARMMVC         Copy the parm data
***********************************************************************
         CALL  GETLEN,             ** GETLEN is the C subroutine     **X
               (PARMDATA,          ** Parameter data area            **X
               PARMLEN),           ** Return length field            **X
               VL,MF=(E,CALLLST)
***********************************************************************
         L     R15,PARMLEN         Get parameter length
         CVD   R15,DBL1            Convert to decimal
         L     R15,DBL1+4          Load significant portion
         SRL   R15,4               Dump the 'sign'
         ST    R15,DBL2            Save the length
         UNPK  DBL1(9),DBL2(5)     Unpack the value
         NC    DBL1(8),=8X'ØF'     Clear high order nibbles
         TR    DBL1(8),=C'Ø123456789' Make the value readable
         MVC   WTO1WRK(WTO1LN),WTO1LST Copy WTO model
         MVC   WTO1WRK+32(4),DBL1+4 Copy readable parm length
         WTO   MF=(E,WTO1WRK)      Issue WTO
***********************************************************************
RETURN   DS    ØH
         L     R5,RETCODE          Load return code
         CEETERM  RC=(R5),MF=(E,CEETERMW)
RETURNØØ DS    ØH
         MVC   RETCODE(4),=F'Ø'    Set return code value
         B     RETURN              Return
RETURNØ4 DS    ØH
         MVC   RETCODE(4),=F'4'    Set return code value
         B     RETURN              Return
***********************************************************************
PARMMVC  MVC   PARMDATA(*-*),2(R3) Copy parm data
***********************************************************************
WTO1LST  WTO   'ASM2C - Parameter length is xxxx     ',                X
               ROUTCDE=(1),DESC=(6),MF=L
WTO1LN   EQU   *-WTO1LST
***********************************************************************
*
MAINPPA  CEEPPA                   Constants describing the code block
* ================================================================== *
*        The Workarea and DSA                                        *
* ================================================================== *
WORKAREA DSECT
         ORG    *+CEEDSASZ         Leave space for the DSA fixed part
CALLLST  CALL  ,(Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø,Ø),VL,MF=L
CEETERMW CEETERM MF=L
*
RETCODE  DS     F                  Return code
PARMADDR DS     F                  Parameter address
PARMDATA DS     CL256              Parameter data save area
PARMLEN  DS     F                  Parameter length returned by GETLEN
WTO1WRK  DS     ØD,CL(WTO1LN)      WTO work area
DBL1     DS     2D                 A work area
DBL2     DS     2D                 A work area
         DS     ØD
WORKSIZE EQU    *-WORKAREA
***********************************************************************
         CEEDSA                    Mapping of the Dynamic Save Area
         CEECAA                    Mapping of the Common Anchor Area
*
RØ       EQU   Ø
```

```
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

ASMWAIT ASM

The ASCII source file for the ASMWAIT program referenced in the article:

```
**********************************************************************
*     INPUT: PARMØ    - ADDRESS OF INCOMING PARMS                   *
*            PARM1    - ADDRESS OF THE WAIT TIME VALUE (8 BYTES CHAR)  *
*            PARM2    - ADDRESS OF WAIT TIME SECONDS   (4 BYTES BIN)   *
*            PARM3    - ADDRESS OF WAIT TIME HUNDRETHS (4 BYTES BIN)   *
*                                                                    *
*    OUTPUT: RETURN CODE DESIGNATION                                 *
*            R15 -    Ø WAIT HAS BEEN SUCCESSFUL                     *
*                    -8 INCORRECT NUMBER OF PARMS DETECTED            *
*                    -9 INCOMING PARM PROBLEM                         *
*                                                                    *
*   CB STRUCTURE USED:                                               *
*            LOCAL WORKAREA DSECT                                    *
**********************************************************************
ASMWAIT  CSECT
ASMWAIT  AMODE 31
ASMWAIT  RMODE ANY
         EDCPRLG BASEREG=R12,DSALEN=WORKLEN
         USING WAITWORK,R13             ADDRESSABILITY TO TEMP STORAGE
**********************************************************************
*   R1 CONTAINS THE ADDRESS OF THE PASSED PARM ADDRESS.  THREE PARMS  *
*   HAVE BEEN PASSED (FOR EXAMPLE PURPOSES).  THE THIRD PARM ADDRESS  *
*   SHOULD HAVE THE X'8Ø' HIGH ORDER BIT SET INDICATING THAT IT IS    *
*   THE LAST PARM ADDRESS (STANDARD CALL WITH VL OPTION PROTOCOL).    *
*   CHAIN THROUGH THE PARM ADDRESSES AND SAVE THEM.  RETURN -8 IF     *
*   THE NUMBER OF PARMS DETECTED IS INCORRECT.                        *
**********************************************************************
         ST    R1,PARMØ                 SAVE INCOMING PARM ADDRESS
         LTR   R1,R1                     PARMS OK?
         BZ    RETNEGØ9                  NO - RETURN -9
         LR    R9,R1                     COPY PARM ADDRESS
         L     R2,Ø(,R9)                 GET BUFFER ADDRESS
         ST    R2,PARM1                  SAVE PARM ADDRESS
         L     R2,4(,R9)                 GET BUFFER ADDRESS
         ST    R2,PARM2                  SAVE PARM ADDRESS
         L     R2,8(,R9)                 GET BUFFER ADDRESS
         ST    R2,PARM3                  SAVE PARM ADDRESS
         TM    PARM3,X'8Ø'               IS THIS THE LAST PARM?
         BNO   RETNEGØ8                  NO - RETURN -8
         NI    PARM3,X'7F'               SANITIZE THE X'8Ø' BIT
***********************************************************************
*   R2 CONTAINS THE ADDRESS OF THE RETURN WAIT TIME VALUE.            *
***********************************************************************
         L     R2,PARM1                  COPY PARM ADDRESS
         L     R5,PARM1                  COPY PARM ADDRESS
         LA    R15,8                     SET LOOP COUNT
DIGITCHK EQU   *
         CLI   Ø(R5),C'Ø'                A DECIMAL DIGIT?
         BL    RETNEGØ9                  NO - RETURN ERROR
         CLI   Ø(R5),C'9'                A DECIMAL DIGIT?
         BH    RETNEGØ9                  NO - RETURN ERROR
         LA    R5,1(,R5)                 POINT TO NEXT BYTE
         BCT   R15,DIGITCHK              CHECK NEXT BYTE
HOURCHK  EQU   *
         CLC   Ø(2,R2),=C'23'            HOUR VALUE OK?
         BH    RETNEGØ9                  NO - RETURN ERROR
MINCHK   EQU   *
         CLC   2(2,R2),=C'59'            MINUTE VALUE OK?
         BH    RETNEGØ9                  NO - RETURN ERROR
SECCHK   EQU   *
         CLC   4(2,R2),=C'59'            SECOND VALUE OK?
         BH    RETNEGØ9                  NO - RETURN ERROR
***********************************************************************
*   IF WE GET HERE THE PASSED TIMER VALUE IS SYNTACTICALLY CORRECT.   *
*   BUILD THE NECESSARY STIMERM COMPONENTS.                           *
***********************************************************************
         MVC   TIMERWK1(TIMERLN1),TIMERLS1 MOVE IN MODEL STIMERM
         STIMERM SET,                                                  X
               ID=TIMERID1,                                            X
               WAIT=YES,                                               X
               DINTVL=(R2),                                            X
               MF=(E,TIMERWK1)
***********************************************************************
RETURNOK EQU   *
***********************************************************************
         MVC   RETCODE(4),=F'Ø'          SET RETURN CODE TO Ø
***********************************************************************
         B     RETURN                    RETURN
RETNEGØ8 EQU   *
```

```
         MVC   RETCODE(4),=F'-8'         SET RETURN CODE TO -8
         B     RETURN                    RETURN
RETNEGØ9 EQU   *
         MVC   RETCODE(4),=F'-9'         SET RETURN CODE TO -9
         B     RETURN                    RETURN
RETURN   EQU   *
         L     R15,RETCODE               LOAD RETURN CODE
         EDCEPIL
***********************************************************************
TIMERLS1 STIMERM SET,MF=L
TIMERLN1 EQU   *-TIMERLS1
***********************************************************************
         LTORG
***********************************************************************
WAITWORK EDCDSAD
PARMØ    DS    F                         INCOMING PARM ADDRESS
PARM1    DS    F                         ADDR OF WAIT TIME - HHMMSSHH
PARM2    DS    F                         ADDR OF WAIT TIME SECONDS
PARM3    DS    F                         ADDR OF WAIT TIME HUNDREDTHS
RETCODE  DS    F                         RETURN CODE
TIMERWK1 DS    ØD,CL(TIMERLN1)           STIMERM WORK AREA
TIMERID1 DS    D                         STIMERM ID
DBL1     DS    2D                        DBL WORD WORK AREA
DBL2     DS    2D                        DBL WORD WORK AREA
WORKLEN  EQU   *-WAITWORK
RØ       EQU   Ø
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R1Ø      EQU   1Ø
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
         END
```

C2ASM.C

The ASCII source file for the C2ASM program referenced in the article:

```
/*  This C program establishes a main() program environment that
 *  calls a MySleep() C subroutine (included with this program
 *  listing).  The MySleep() function accepts two parameters - the
 *  number of seconds and the number of fractions of a second (in
 *  hundredths) that the program is to wait.
 *  The MySleep() function does some data validation and then
 *  calls the ASMWAIT Assembler routine.  The ASMWAIT routine expects
 *  three incoming parameters (only one is really used, but this
 *  was done to demonstrate additional flexibility).  ASMWAIT does
 *  some additional data validation and if everything looks good, it
 *  will issue an STIMERM wait for the specified length of time.
 *  MySleep() provides some benefit over the traditional C sleep()
 *  function in that you can wait for fractions of a second (that is
 *  not available with the sleep() function).                        */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*  Indicate to the compiler that standard OS linkage will be used.  */
#ifdef __cplusplus
 extern "OS" int ASMWAIT(char*, int*, int*);
#else
 #pragma linkage (ASMWAIT, OS)
#endif
#define  MVS
int MySleep(int sec, int hund)
{
/*  Do some data validation, build the wait time value, and call
 *  the ASMWAIT Assembler function.                                  */
 char numtable[201]     = "00010203040506070809"
                          "10111213141516171819"
                          "20212223242526272829"
                          "30313233343536373839"
                          "40414243444546474849"
                          "50515253545556575859"
                          "60616263646566676869"
                          "70717273747576777879"
                          "80818283848586878889"
                          "90919293949596979899";
#define DAY_LIMIT         86400   // Number of seconds in a day
#define HUNDRETHS_LIMIT   100     // Number of hundredths of a second
#define LIMIT_ERROR      -1       // More than 86,399 seconds specified
#define HUNDRETHS_ERROR  -2       // More than 99 hundredths specified
 int hours, minutes, seconds, hundreths;
 int offset;
 int i;
 char timebuff[9];
 div_t modval;
 seconds = sec;
 hundreths = hund;
/*  Make sure the timebuff storage is sanitized.  */
 for (i=0; i<10; i++)
 {
    timebuff[i] = 0;
 }
/*  Check if number of seconds greater than one day. */
 if (seconds >= DAY_LIMIT)
 {
    return(LIMIT_ERROR);
 }
/*  Check if number of hundredths of a second greater than 100.  */
 if (hundreths >= HUNDRETHS_LIMIT)
 {
    return(HUNDRETHS_ERROR);
 }
/*  Calculate number of hours specified.  */
 modval = div(seconds,3600);
 hours = modval.quot;
/*  Calculate number of minutes specified.  */
 seconds = modval.rem;
 modval = div(seconds,60);
 minutes = modval.quot;
/*  Calculate number of seconds specified.  */
 seconds = modval.rem;
/*  Create a character string that looks like 'HHMMSSHH'.  */
 offset = hours * 2;
 strncpy(timebuff,numtable+offset,2);
 offset = minutes * 2;
 strncat(timebuff,numtable+offset,2);
 offset = seconds * 2;
 strncat(timebuff,numtable+offset,2);
 offset = hundreths * 2;
 strncat(timebuff,numtable+offset,2);
 printf("Wait time is %s\n",timebuff);
/*  Go wait the specified length of time.  */
 i = ASMWAIT(timebuff,&seconds,&hundreths);
 printf("ASMWAIT() rc=%d\n",i);
 return(i);
}
main()
{
 int  i;
 i = MySleep(10,10);    /*  Wait for 10.10 seconds.  */
 printf("Done waiting\n");
 return(0);
}
```

GETLEN.C

The ASCII source file for the GETLEN program referenced in the article:

```
/*  This is a C program subroutine that is invoked from a calling
 *  Assembler program that has established the main() enclave.  The
 *  GETLEN subroutine will be invoked via a standard Assembler
 *  CALL macro from the calling program.
 *  This subroutine expects two incoming parameters:
 *     PARM1:  is the address of a character string (null delimited)
 *     PARM2:  is the address of a fullword used to return the
 *             length of the passed character string                */
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
/*  Indicate to the compiler that standard OS linkage will be used.  */
#ifdef __cplusplus
 extern "OS" int GETLEN(char*, int*);
#else
 #pragma linkage (GETLEN,OS)
#endif
int GETLEN(char *parmdata, int *parmlen)
{
int i;
/*  Determine the length of the character string and update the
 *  return length field with the appropriate value.               */
i = strlen(parmdata);
*parmlen = i;
printf("Parameter data is:  %s\n",parmdata);
printf("Parameter length is:  %d\n",*parmlen);
return Ø;
}
```

Systems Programmer (Canada) © Xephon 2002

Mainframe spreadsheet

We regularly receive standard reports from our branch offices in printable format. We needed to carry out some additional calculation on the incoming data in order to merge the reports and produce some global figures and trends. Our end users tried to do that using some of the standard PC spreadsheet programs. Transferring data in both directions and the conversion of characters caused different problems in practice. These were the reasons for writing a utility that performs calculations on table data on the OS/390 platform. Exploitation of this utility gave us following benefits:

- We avoid transferring data from host to PC and *vice versa*.
- Calculations on data in standard tables are moved into production, after the initial preparation and testing of control statements.

- We can process very large tables.

PROGRAM DESCRIPTION

The CALCTAB utility program is written in the PL/I language to enable users to calculate inside the existing tables. Additionally, the calculation program has the ability to check existing data.

Users write control statements to describe the input data and to specify the calculation rules. They put them in a sequential dataset assigned to the SYSIN DDname. The program processes the input table, calculates data from the table according to specified statements, and prints the results into the same table. The following example will demonstrate calculating with CALCTAB on one simple table.

Input table

```
                                                                          Row No.
                Sales report for 18.03.02.                                  001
                                                                            002
   cipher value date  number of p. price/piece*        value   percent     003
 ==================================================================        004
   124401  28.03.02.          257    11,123.45                              005
   363656  28.03.02.           72     3,235.78                              006
   111503  01.04.02.            8     1,247.99                              007
   156865  03.04.02.           92     7,456.72                              008
   167855  04.04.02.            2     2,257.37                              009
  ==================================================================       010
   TOTAL:                                                                   011
* Column price/piece has numeric with decimal point (.) and separator(,) 012
```

Parameters in SYSIN dataset

```
 TRACE;
 COLUMNS   A=(25:33,0)  B=(35:46,2) C=(48:60,2,',','.') D=(64:70,2);
 ROWS      (5:11);
 C = B * A;
 A11 = SUM(A5:A9);
 C11 = SUM(C5:C9);
 D5:D11 = C5:C11   / C11 * 100 ;
```

Resulting table

```
                                                                          Row No.
                Sales report for 18.03.02.                                  001
                                                                            002
   cipher value date  number of p. price/piece*        value   percent     003
```

```
  ================================================================   ØØ4
   1244Ø1  28.Ø3.Ø2.          257    11,123.45  2.858.726,65     75.38   ØØ5
   363656  28.Ø3.Ø2.           72     3,235.78    232.976,16      6.14   ØØ6
   1115Ø3  Ø1.Ø4.Ø2.            8     1,247.99      9.983,92      Ø.26   ØØ7
   156865  Ø3.Ø4.Ø2.           92     7,456.72    686.Ø18,24     18.Ø9   ØØ8
   167855  Ø4.Ø4.Ø2.            2     2,257.37      4.514,74      Ø.12   ØØ9
  ================================================================   Ø1Ø
   TOTAL:                     431               3.792.219,71    1ØØ.ØØ   Ø11
* Column price/piece has numeric with decimal point (.) and separator(,) Ø12
```

CONTROL STATEMENT SYNTAX

Statements are executed in the specified order. The end of the statements is marked by a semicolon. You can type the statements in upper or lower case; the program converts everything to capitals. There are three types of control statements:

1 Comments, eg:

  *text of comment

2 Descriptive statements:

  – TRACE

  – CHECK

  – COLUMNS

  – ROWS.

3 Executive statements (formulas).

You can specify a formula that performs mathematical operations in a format similar to a standard PC spreadsheet program. A formula is an equation with elements to be calculated, which are separated by calculation operators. Each element can be a cell, a range of cells, a function, or a constant.

Comments

All rows in control datasets that have ‘*’ in the first position are considered as comments and the program ignores their content.

Descriptive statements

Descriptive statements must be specified in the proper order. TRACE and CHECK are optional and must be at the beginning, followed by the COLUMNS and ROWS, which are mandatory.

When a user specifies a TRACE statement, the program prints a detailed report about statement execution in the output list. This statement is optional and can be used for better understanding the effects of the executive statements.

Use CHECK to check figures that already exist in the table, comparing them with just-calculated data. If a difference exists, an error message is written, accompanied by the checked row and calculated row.

If you omit the CHECK statement, the program calculates the data as specified by the statements. When the calculation is finished, data is stored in predefined places.

COLUMNS is a mandatory statement and has the following syntax:

```
COLUMNS Name1=(SP1:EP1,D1,'P1','S1')
        Name2=(SP2:EP2,D2,'P2','S2')
           ...
        Namen=(SPn:EPn,Dn,'Pn','Sn');
```

COLUMNS defines the position of the columns inside the table.

Abbreviations used are:

- NAMEi – alphabetic string up to eight bytes long, first and last character must not be numeric. A function name may not be used as a column name.
- SPi – start position of *i*th column
- EPi – end position of *i*th column
- Di – number of decimal numbers to describe precision of calculated data in column *i*. If you omit it, the default value is 0.
- Pi – optional parameter that represents a decimal point – it can be only a comma or a full stop (period).
- Si – optional parameter that means class separator – it can be only a comma or a full stop (period), and must be different from the previous decimal point.

Input data must be in the range from SP to EP, and output data will be justified to the right margin EP.

Example:

```
          ----+----1----+----2----+----3----+----4----+----5----+----6---
     table row :
          optional text1 |    1Ø|  opt text 2   |  6Ø.ØØ  |  234.157,7Ø|
          optional text3 |    15|  opt text 4   |  72.ØØ  |   15.242,2Ø|
```

Statement for defining column position:

```
      COLUMNS   PRICE=(18:23) AMOUNT=(41:49) TOTAL=(51:62,2,',','.');
```

ROWS is a mandatory statement, the last from the descriptive group. You can use ROWS to specify rows that take part in a calculation. The format of the ROWS statement looks like:

```
     ROWS (SR1:ER1) (SR2:ER2) ... ;
```

where SR*i* stands for the starting row and ER*i* stands for the ending row.

If you have numeric rows intercepted by plenty of delimiters and you don't want to specify exactly how many numeric rows, you can specify the first and last numeric row of the table in a single bracket in the ROWS statement. The program will ignore all rows that have no data in all the specified column positions. You can check the output report from CALCTAB to see which rows are eliminated from the calculation.

Executive statements

Some typical examples of formulas are:

```
          TOTAL   = PRICE * AMOUNT;
          PERCENT = TOTAL / TOTAL11 * 1ØØ ;
          A19 = SUM(A5:A18:2);
          B24 = RANGE(B4:B22:2);
          D5:D11 = C5:C11 / C11 * 1ØØ ;
```

In the descriptive statement COLUMNS, you have already named the columns of the table. Reference to a row is by its relative record number in the editor. Following this naming convention, we use A19 to refer to the cell at the intersection of column A and row 19. To refer to the range of cells, we have to start with the cell in the upper-left corner of the range, a colon, and then the reference to the cell in the lower-right corner of the range, as in the last three examples.

Calculation operators that separate references to the cells or range of cells are:

- + – addition

- - – subtraction
- * – multiplication
- / – division
- ** – raise to power.

The following functions can be applied to the references to cells or range of cells:

- SUM – sum cells in specified range.
- MIN – minimum value inside the specified range.
- MAX – maximum value inside the specified range.
- AVRG – average value in the specified range of cells.
- RANGE – difference between maximum and minimum value inside the specified range.

These functions are chosen to meet our customers' needs. We encourage new users of CALCTAB to add new functions to the source code using existing ones as a model.

All functions can have one additional parameter inside the bracket delimited by a colon from the reference to cell range. This parameter represents row step for picking values from specified ranges. It can be very useful for tables that have rows with data intercepted with delimiters.

An example is:

```
A11 = SUM(A2:A8:2);
```

This means that you want to sum only even rows of column A in the range from 2 to 8 and place the result in row 11.

Arithmetic operations are performed from left to right according to the order of operator precedence. You can use parentheses to change the order of operations.

Examples:

```
    A= B + C * D / ( E - F ) ** 2 ;
    FF= 2 * SUM(AA:EE);
```

PROGRAM CALCTAB

```
CALCTAB: PROCEDURE OPTIONS(MAIN);
/*************************** DATASETS ********************************/
DCL TABLE    FILE RECORD SEQL,
    WORK     FILE RECORD SEQL KEYED ENV(VSAM),
    SYSIN    FILE RECORD SEQL INPUT,
    SYSPRINT FILE STREAM OUTPUT ENV(FB RECSIZE(133) BLKSIZE(399Ø));
/*************************** RECORDS *********************************/
DCL RECORD_TABLE CHAR(32765) VAR;
/*-------- Record for executive statement for row or column --------*/
DCL 1 STMT  BASED(P_STMT),
     2 NEXT       PTR INIT(NULL),        /*POINTS TO NEXT STATEMENT  */
     2 COL_NO1    BIN FIXED INIT(Ø),     /*ORDINAL NUMBER OF COLUMN 1*/
     2 COL_NO2    BIN FIXED INIT(Ø),     /*ORDINAL NUMBER OF COLUMN 2*/
     2 ROW_NO1    BIN FIXED(31) INIT(Ø), /*ORDINAL NUMBER FROM ROW   */
     2 ROW_NO2    BIN FIXED(31) INIT(Ø), /*ORDINAL NUMBER TO   ROW   */
     2 ROW_STEP   BIN FIXED(15),         /*ORDINAL NUMBER TO STEP    */
     2 P_ARG      PTR INIT(NULL);        /*POINTS TO TREE OF STATEMENT
                                           EXECUTION*/
/*-- Different types of records that are pointed by stmt.P_ARG ----*/
DCL 1 ARG_OPER BASED(P_ARG),
     2 NEXT       PTR INIT(NULL),
     2 TYPE       CHAR(1) INIT(' ');
DCL 1 ARG       BASED(P_ARG),
     2 NEXT       PTR INIT(NULL),
     2 TYPE       CHAR(1) INIT(' '),
     2 COL_NO1    BIN FIXED INIT(Ø),
     2 COL_NO2    BIN FIXED INIT(Ø),
     2 ROW_NO1    BIN FIXED(31) INIT(Ø),
     2 ROW_NO2    BIN FIXED(31) INIT(Ø),
     2 ROW_STEP   BIN FIXED(15) INIT(1);
DCL 1 ARG_CONST   BASED(P_ARG),
     2 NEXT       PTR INIT(NULL),
     2 TYPE       CHAR(1) INIT(' '),
     2 NUMBER     BIN FLOAT(53) INIT(Ø);
/*------ Work record for calculating data in one row ----------------*/
DCL 1 WORK_RECORD BASED(P_WR),
     2 NEXT          PTR INIT(NULL),
     2 ROW_NO        BIN FIXED(31),          /*ORDINAL NUMBER OF ROW*/
     2 COLS_NUM      BIN FIXED,              /*NUMBER OF COLUMNS */
     2 DATA(COLS_NO  REFER(COLS_NUM)) BIN FLOAT(53) INIT((COLS_NO)Ø),
                                         /*IS THERE ANY CHANGED DATA?*/
     2 IND_CHANGE(COLS_NO REFER(COLS_NUM)) BIT INIT((COLS_NO)'Ø'B);
/*------ Work record for get column and row ordinal no --------------*/
DCL 1 W_STMT,
      2 TYPE    CHAR(1),
      2 COL_NO1 BIN FIXED,
      2 COL_NO2 BIN FIXED,
      2 ROW_NO1 BIN FIXED(31),
      2 ROW_NO2 BIN FIXED(31),
      2 ROW_STEP BIN FIXED(15);
```

```
/*-- Index of table row ---------------------------------------------*/
DCL INDEX_WORK_RECORD(NO_TABLE_RECORDS) PTR CTL
    INIT((NO_TABLE_RECORDS) NULL);
/*-- Columns definition ---------------------------------------------*/
DCL 1 COLS(COLS_NO) CTL,
     2 COL_NAME        CHAR(8) VAR,
     2 FROM_POSITION   BIN FIXED,
     2 TO_POSITION     BIN FIXED,
     2 NO_OF_DECIMALS  BIN FIXED,
     2 DECIMAL_POINT   CHAR(1),
     2 SEPARATOR       CHAR(1);
/*-- Rows definition ----------------------------------------------*/
DCL 1 ROWS(ROWS_NO) CTL,
     2 FROM_POSITION BIN FIXED(31),
     2 TO_POSITION   BIN FIXED(31);
/************************** WORK VARIABLES *************************/
DCL (Q, P_ARG, PPSR) PTR;
DCL (P_STMT, P_STMT_ROOT INIT(NULL), PP_STMT) PTR;
DCL (P_WR, P_WR_ROOT INIT(NULL), PP_WR) PTR;
DCL (COLS_NO INIT(-1), C) BIN FIXED,
    (ROWS_NO INIT(-1), NO_TABLE_RECORDS) BIN FIXED(31);
DCL NOT_EOF_TABLE BIT INIT('1'B);
DCL (IND_CHECK INIT('Ø'B), IND_TRACE INIT('Ø'B)) BIT;
DCL CH(-1:32765)   CHAR(1)   BASED(Q),
    LEN_REC_STMT   BIN FIXED BASED(Q);
DCL NO_CELL_IN_RESULT BIN FIXED(31);
DCL W  PIC'ZZZZZZ9';
/************************** BUILTIN FUNCTIONS **********************/
DCL (ADDR, NULL, SUBSTR, INDEX, TRANSLATE, ABS, SIGN, ONCODE,
     MIN, LENGTH, VERIFY, REPEAT, ANY, ALL, PLIRETC) BUILTIN;
/************************** PROCEDURES *****************************/
ON ERROR SNAP SYSTEM;
ON ENDFILE(TABLE)  NOT_EOF_TABLE = 'Ø'B;
/*-- Lexical analysis of statements and preparation for calculation */
CALL GET_STATEMENTS;
/*-- Copying from table to work dataset ----------------------------*/
Q = ADDR(RECORD_TABLE);
CALL COPY_TABLE_IN_WORK_BUFF;
/*-- Calculate data in table --------------------------------------*/
CALL CALCULATE;
IF IND_CHECK
/*-- CHECK data calculated earlier in table -----------------------*/
THEN CALL CHECK_DATA_IN_TABLE;
/*-- Copying from work dataset to table ----------------------------*/
ELSE CALL WRITE_COUNT_DATA_INTO_TABLE;
/*-- Free work buff ------------------------------------------------*/
DO P_WR = P_WR_ROOT WHILE(P_WR ¬= NULL);
   PP_WR = WORK_RECORD.NEXT;
   FREE WORK_RECORD;
   P_WR = PP_WR;
END;
/************************** WORK PROCEDURES ************************/
```

```
/* PROCEDURE DOES LEXICAL ANALYSES OF STATEMENTS AND ACCORDING TO    */
/* THEM CONVERTS EXECUTIVE STATEMENTS FROM INFIX TO PREFIX POLISH    */
/* NOTATION                                                          */
/*******************************************************************/
GET_STATEMENTS: PROCEDURE;
DCL  RECORD_STMT CHAR(512) VAR,
     OPER CHAR(1),
     (I, J, K, L, BZ INIT(Ø)) BIN FIXED,
     PP PTR,
     (NOT_END_STMT, NOT_EOF_SYSIN INIT('1'B)) BIT AUTO;
ON ENDFILE(SYSIN)  NOT_EOF_SYSIN='Ø'B;
/*-- Get descriptive statements and analyze them ------------------*/
PUT SKIP EDIT('=== GET STATEMENTS ',REPEAT('=',53)) (A);
Q=ADDR(RECORD_STMT);
I = #NEXT_NONBLANK#(LEN_REC_STMT+1);
IF SUBSTR(RECORD_STMT,I,5) = 'CHECK' |
   SUBSTR(RECORD_STMT,I,5) = 'TRACE'
THEN DO J = 1 TO 2;
        IF SUBSTR(RECORD_STMT,I,5) = 'CHECK'
        THEN DO;
             IND_CHECK = '1'B;
             I = #NEXT_STMT#(I+5);
             END;
        IF SUBSTR(RECORD_STMT,I,5) = 'TRACE'
        THEN DO;
             IND_TRACE = '1'B;
             I = #NEXT_STMT#(I+5);
             END;
     END;
/*-- Get column definitions --------------------------------------*/
CALL #GET_COLUMNS#;
/*-- Get row definitions ------------------------------------------*/
CALL #GET_ROWS#;
/*-- Get executive statements and analyse them --------------------*/
CALL #GET_ARITMETIC_STMTS#;
/*---------------- Print internal definitions ---------------------*/
IF IND_TRACE
THEN DO;
     PUT SKIP EDIT(' ','=== TRACE INFORMATION ',REPEAT('=',48))
                  (A,SKIP, 2 A);
     PUT SKIP DATA(IND_CHECK,IND_TRACE);
     PUT SKIP EDIT('-- COLUMNS DEFINITION -------------------') (A);
     DO I = 1 TO COLS_NO;
        PUT SKIP DATA(COLS(I));
     END;
     PUT SKIP EDIT('-- ROWS DEFINITION -----------------------') (A);
     DO I = 1 TO ROWS_NO;
        PUT SKIP DATA(ROWS(I));
     END;
     DO PP_STMT = P_STMT_ROOT REPEAT(PP_STMT->STMT.NEXT)
                  WHILE(PP_STMT ¬= NULL);
        CALL PRINT_STMT(PP_STMT);
     END;
     END;
/*******************************************************************/
/* PROCEDURE FINDS NEXT NONBLANK SYMBOL IN STATEMENTS              */
/* AND SKIP COMMENTS                                               */
/*******************************************************************/
#NEXT_NONBLANK#: PROCEDURE(J) RETURNS(BIN FIXED);
  DCL J BIN FIXED;
  DO UNTIL(¬ NOT_EOF_SYSIN | J <= LEN_REC_STMT);
     IF J > LEN_REC_STMT & NOT_EOF_SYSIN
     THEN DO;
          READ FILE(SYSIN) INTO(RECORD_STMT);
          LEN_REC_STMT= MIN(72,LENGTH(RECORD_STMT));
          IF NOT_EOF_SYSIN
          THEN PUT SKIP EDIT(RECORD_STMT) (A);
          J=1;
          END;
     DO J=J TO LEN_REC_STMT WHILE(CH(J)=' ');
     END;
     IF J=1 & CH(J) = '*' /* SKIP COMMENT */
     THEN J=LEN_REC_STMT+1;
/*-- Converting lowercase into uppercase in statements ----------*/
     RECORD_STMT = TRANSLATE(RECORD_STMT,'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
                                         'abcdefghijklmnopqrstuvwxyz');
  END;
  NOT_END_STMT = (NOT_EOF_SYSIN & CH(J) ¬= '5E'X);
RETURN(J);
END #NEXT_NONBLANK#;
/*******************************************************************/
/* PROCEDURE INFORMS ABOUT SYNTAX ERROR                            */
/*******************************************************************/
#SYNTAX_ERROR#: PROCEDURE(MSG);
 DCL MSG CHAR(*);
/*-- Position of error is in variable i plus asa character --------*/
 PUT SKIP EDIT('*',' <-- POSITION',I) (X(I-1),A,A,A);
 PUT SKIP EDIT('*** ERROR ',MSG) (A);
 IF BZ>Ø
 THEN PUT SKIP EDIT(' RIGHT PARENTHESIS IS MISSING ',BZ) (A);
 IF BZ<Ø
 THEN PUT SKIP EDIT(' ????? OF RIGHT PARENTHESIS  ',-BZ) (A);
 CALL PLIRETC(12);
 EXIT;
END #SYNTAX_ERROR#;
/*******************************************************************/
/* PROCEDURE FINDS BEGINNING OF THE NEXT STATEMENT                 */
/*******************************************************************/
#NEXT_STMT#: PROCEDURE(J) RETURNS(BIN FIXED);
  DCL J BIN FIXED;
  I = #NEXT_NONBLANK#(J);
  IF NOT_END_STMT
  THEN CALL #SYNTAX_ERROR#('EXPECTED '||'5E'X);
  I = #NEXT_NONBLANK#(I+1);
RETURN(I);
END #NEXT_STMT#;
/*********************************************************************/
/* PROCEDURE FINDS EXPECTED CHARACTER                                */
/*********************************************************************/
#FIND_CHAR#: PROCEDURE(J,C,IND_MANDATORY,MSG) RETURNS(BIN FIXED);
  DCL (I,J) BIN FIXED;
  DCL C CHAR(1);
  DCL IND_MANDATORY BIT;
  DCL MSG CHAR(*);
  DO I=J TO LEN_REC_STMT WHILE( CH(I) ¬= '5E'X &
            VERIFY(CH(I),'+-*/(),.=:''') ¬= Ø);
  END;
  IF IND_MANDATORY & CH(I) ¬= C
  THEN CALL #SYNTAX_ERROR#('EXPECTED "'||C||'" '||MSG);
 RETURN(I);
END #FIND_CHAR#;
/*********************************************************************/
/* PROCEDURE GET NUMERICS                                            */
/*********************************************************************/
#GET_INT_NUMBER#:PROCEDURE(STR,IND_POSITIV,DESCRIPTION_TEXT)
                  RETURNS(BIN FIXED(31));
  DCL STR CHAR(*);
  DCL IND_POSITIV BIT;
  DCL DESCRIPTION_TEXT CHAR(*);
  DCL N      BIN FIXED(31);
  IF VERIFY(STR,'Ø123456789 ')¬=Ø | LENGTH(STR) = Ø
  THEN DO;
       PUT SKIP EDIT('>',STR,'<') (A);
       CALL #SYNTAX_ERROR#
            ('EXPECTED NUMERICS FOR DEFINITION '||DESCRIPTION_TEXT);
       END;
  GET STRING(STR) LIST(N);
  IF IND_POSITIV & N <= Ø
  THEN CALL #SYNTAX_ERROR#
          (DESCRIPTION_TEXT||' MUST BE GREATER THAN Ø');
 RETURN(N);
END #GET_INT_NUMBER#;
/*********************************************************************/
/* PROCEDURE GETS COLUMNS DEFINITIONS                                */
/*********************************************************************/
#GET_COLUMNS#: PROCEDURE;
DCL L BIN FIXED;
DCL (P_W_COLS, P_W_COLS_ROOT INIT(NULL), PPW) PTR;
DCL PP PTR INIT(NULL);
/*-- Work record for reading of row and column numbers -----------*/
DCL 1 W_COLS BASED(P_W_COLS),
     2 NEXT       PTR INIT(NULL),
     2 COL_NAME        CHAR(8) VAR,
     2 FROM_POSITION   BIN FIXED,
      2 TO_POSITION     BIN FIXED,
      2 NO_OF_DECIMALS  BIN FIXED INIT(Ø),
      2 DECIMAL_POINT   CHAR(1) INIT('.'),
      2 SEPARATOR       CHAR(1) INIT(' ');
If SUBSTR(RECORD_STMT,I,7) ¬= 'COLUMNS'
THEN CALL #SYNTAX_ERROR#('EXPECTED "COLUMNS" DEFINITIONS');
I = #NEXT_NONBLANK#(I+7);
PPW = ADDR(P_W_COLS_ROOT);
DO COLS_NO = 1 BY 1 WHILE(NOT_EOF_SYSIN & CH(I) ¬= '5E'X);
   ALLOC W_COLS;
   PPW->W_COLS.NEXT = P_W_COLS;
   PPW = P_W_COLS;
   W = COLS_NO;
/*-- Get Col Name ---------------------------------------------------*/
   IF CH(I) >= 'Ø' & CH(I) <= '9'
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME BEGINS WITH NUMERIC !');
   J = #FIND_CHAR#(I,'=','1'B,'AFTER COLUMN NAME');
   DO L = J-1 TO I BY -1 WHILE(CH(L) = ' ');
   END;
   IF CH(L) >= 'Ø' & CH(L) <= '9'
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME ENDS WITH NUMERIC !');
   L = L - I + 1;
   IF L = Ø
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME IS NOT DEFINED');
   IF L > 8
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME EXCEEDS 8 CHARACTERS '||
                            SUBSTR(RECORD_STMT,I,L));
   IF #CHECK_FUNCTION_NAME#(SUBSTR(RECORD_STMT,I,L)) ¬= ' '
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME IS EQUAL AS FUNCTION NAME');
   W_COLS.COL_NAME = SUBSTR(RECORD_STMT,I,L);
   I = #NEXT_NONBLANK#(J + 1);
   IF CH(I) ¬= '('
   THEN CALL #SYNTAX_ERROR#('EXPECTED "(" FOR DEFINED COLUMNS
   '||W);
/*-- Get START position of column definition ---------------------*/
   I = I + 1;
   J=#FIND_CHAR#(I,':','1'B,'BETWEEN START & END POSITION OF COLUMN');
   W_COLS.FROM_POSITION=#GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                                '1'B, 'START POSITION OF COLUMNS '||W);
   IF (PP ¬= NULL)
   THEN IF PP->W_COLS.TO_POSITION > W_COLS.FROM_POSITION
        THEN CALL #SYNTAX_ERROR#(
             'PREVIOUS END POSITION IS GREATHER THEN START POSITION');
/*-- Get END position of column definition -----------------------*/
   I = J + 1;
   J = #FIND_CHAR#(I,',','Ø'B,'');
   W_COLS.TO_POSITION=#GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                                '1'B, 'END POSITION OF COLUMNS '||W);
   IF W_COLS.FROM_POSITION > W_COLS.TO_POSITION
   THEN CALL #SYNTAX_ERROR#('END POS. IS LESS THEN START POSITION');
/*-- Get NO OF DECIMALS IN this column ---------------------------*/
```

```
   IF CH(J) = ','
   THEN DO;
        I = J + 1;
        J = #FIND_CHAR#(I,',','Ø'B,'');
        W_COLS.NO_OF_DECIMALS = #GET_INT_NUMBER#
           (SUBSTR(RECORD_STMT,I,J-I), 'Ø'B,'DECIMALS OF COLUMNS '||W);
        IF W_COLS.NO_OF_DECIMALS >=
          (W_COLS.TO_POSITION - W_COLS.FROM_POSITION)
        THEN CALL #SYNTAX_ERROR#
           ('DECIMALS IS LARGEER THEN COLUMN LENGTH');
        I = J;
        IF CH(I)= ','
        THEN DO;
             W_COLS.DECIMAL_POINT = #GET_CONST#
                                    ('DECIMAL POINT CHARACTER');
             I = #FIND_CHAR#(I,'''','Ø'B,'');
             IF CH(I)= ','
             THEN W_COLS.SEPARATOR = #GET_CONST#
                                    ('SEPARATOR CHARACTER');
             IF W_COLS.DECIMAL_POINT = W_COLS.SEPARATOR
             THEN CALL #SYNTAX_ERROR#
                    ('DECIMAL POINT AND SEPARATOR MUST BE DIFFERENT');
             END;
        J = #FIND_CHAR#(I,')','1'B,'AT END OF COLUMN DEFINITION');
        END;
   I = #NEXT_NONBLANK#(J+1);
   PP = P_W_COLS;
END;
COLS_NO = COLS_NO - 1;
ALLOC COLS;
DO C = 1 TO COLS_NO;
   P_W_COLS = P_W_COLS_ROOT;
   COLS(C) = W_COLS, BY NAME;
   P_W_COLS_ROOT = W_COLS.NEXT;
   FREE W_COLS;
END;
I = #NEXT_STMT#(I);
END #GET_COLUMNS#;
#GET_CONST#:PROCEDURE(MSG) RETURNS(CHAR(1));
DCL MSG CHAR(*);
DCL CH1 CHAR(1);
I = #FIND_CHAR#(I+1,'''','1'B,'FOR '||MSG);
I = I + 1;
CH1 = CH(I);
IF CH(I) ¬= '.' & CH(I) ¬= ','
THEN CALL #SYNTAX_ERROR#('DECIMALS POINT MUST BE "." OR ","');
I = I + 1;
IF CH(I) ¬= ''''
THEN CALL #SYNTAX_ERROR#('MISSING "''" AT END OF '||MSG);
I = #NEXT_NONBLANK#(I+1);
RETURN(CH1);
```

```
END #GET_CONST#;
/*******************************************************************/
/* PROCEDURE GETS ROWS DEFINITIONS                                 */
/*******************************************************************/
#GET_ROWS#: PROCEDURE;
DCL (J,K) BIN FIXED;
DCL (P_W_ROWS, P_W_ROWS_ROOT INIT(NULL), PPW) PTR;
/*-- Work record for reading of row and column numbers -----------*/
DCL 1 W_ROWS BASED(P_W_ROWS),
     2 NEXT          PTR INIT(NULL),
     2 FROM_POSITION BIN FIXED(31),
     2 TO_POSITION   BIN FIXED(31);
If SUBSTR(RECORD_STMT,I,4) ¬= 'ROWS'
THEN CALL #SYNTAX_ERROR#('EXPECTED "ROWS" DEFINITIONS');
I = #NEXT_NONBLANK#(I+4);
PPW = ADDR(P_W_ROWS_ROOT);
DO ROWS_NO = 1 BY 1 WHILE(NOT_EOF_SYSIN & CH(I) ¬= '5E'X);
   ALLOC W_ROWS;
   PPW->W_ROWS.NEXT = P_W_ROWS;
   PPW = P_W_ROWS;
   W = ROWS_NO;
   IF CH(I) ¬= '('
   THEN CALL #SYNTAX_ERROR#('EXPECTED "(" FOR COLUMN DEFINITION '||W);
/*--- Get START position of rows definition -----------------------*/
   I = I + 1;
   J = #FIND_CHAR#(I,':','1'B,'BETWEEN START & END POSITION OF ROW');
   W_ROWS.FROM_POSITION=#GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                             '1'B, 'START POSITION OF ROW '||W);
/*-- Get END position of row definition --------------------------*/
   I = J + 1;
   J = #FIND_CHAR#(I,')','1'B,'AT END OF ROW DEFINITION');
   W_ROWS.TO_POSITION = #GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                             '1'B, 'END POSITION OF ROWS '||W);
   IF W_ROWS.FROM_POSITION > W_ROWS.TO_POSITION
   THEN CALL #SYNTAX_ERROR#('END POSITION IS LESS THEN STARTED!');
   I = #NEXT_NONBLANK#(J+1);
END;
ROWS_NO = ROWS_NO - 1;
ALLOC ROWS;
DO K = 1 TO ROWS_NO;
   P_W_ROWS = P_W_ROWS_ROOT;
   ROWS(K) = W_ROWS, BY NAME;
   P_W_ROWS_ROOT = W_ROWS.NEXT;
   FREE W_ROWS;
END;
I = #NEXT_STMT#(I);
END #GET_ROWS#;
/*******************************************************************/
/*  PROCEDURE CHECKS SYNTAX AND GETS DEFNS OF EXECUTIVE STATEMENTS   */
/*******************************************************************/
#GET_ARITMETIC_STMTS#: PROCEDURE;
```

```
PP_STMT = ADDR(P_STMT_ROOT);
DO WHILE(NOT_EOF_SYSIN);
/*-- Get Col Name --------------------------------------------------*/
   ALLOC STMT;
   PP_STMT->STMT.NEXT = P_STMT;
   PP_STMT = P_STMT;
   CALL #GET_INTERVAL#('IN ARITHMETIC STATEMENT');
   STMT = W_STMT, BY NAME;
   NO_CELL_IN_RESULT = (STMT.COL_NO2 - STMT.COL_NO1 + 1) *
                       (STMT.ROW_NO2 - STMT.ROW_NO1 + 1);
   IF CH(I) ¬= '='
   THEN CALL #SYNTAX_ERROR#('EXPECTED "=" IN ARITHMETIC STATEMENT');
   ELSE I = #NEXT_NONBLANK#(I+1);
   STMT.P_ARG = #PROC_E#(NULL);
   I = #NEXT_STMT#(I);
END;
END #GET_ARITMETIC_STMTS#;
/*******************************************************************/
/* PROCEDURE GETS INTERVAL ROW AND/OR COL IN ARITHMETIC STATEMENT    */
/*******************************************************************/
#GET_INTERVAL#: PROCEDURE(MSG);
DCL MSG CHAR(*);
W_STMT.TYPE = ' ';
J = #CHECK_COL_NAME#('1'B);
W_STMT.COL_NO1 = J;
I = I + LENGTH(COLS(J).COL_NAME);
/*-- Get Row No 1 --------------------------------------------------*/
J = #FIND_CHAR#(I,':','Ø'B,'');
IF CH(I) >= 'Ø' & CH(I) <= '9'
THEN W_STMT.ROW_NO1 = #GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                                       '1'B, 'ROW NO1 '||MSG);
ELSE W_STMT.ROW_NO1 = Ø;
IF W_STMT.ROW_NO1 > ROWS(ROWS_NO).TO_POSITION
THEN CALL #SYNTAX_ERROR#('ROW IS NOT IN ROWS DEFINITION !');
/*-- Get Row No 2 if exist ----------------------------------------*/
IF CH(J) = ':'
THEN DO;
     I = #NEXT_NONBLANK#(J+1);
     C = #CHECK_COL_NAME#('1'B);
     W_STMT.COL_NO2 = C;
     I = I + LENGTH(COLS(C).COL_NAME);
     J = #FIND_CHAR#(I,' ','Ø'B,'');
     IF CH(I) >= 'Ø' & CH(I) <= '9'
     THEN W_STMT.ROW_NO2 = #GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                                            '1'B, 'ROW NO2 '||MSG);
     ELSE W_STMT.ROW_NO2 = Ø;
     END;
ELSE DO;
     W_STMT.COL_NO2 = W_STMT.COL_NO1;
     W_STMT.ROW_NO2 = Ø;
     END;
```

```
IF W_STMT.ROW_NO2 > ROWS(ROWS_NO).TO_POSITION
THEN CALL #SYNTAX_ERROR#('ROW IS NOT IN ROWS DEFINITION !');
/*-- Get Step if exist -------------------------------------------*/
IF CH(J) = ':'
THEN DO;
     I = J + 1;
     J = #FIND_CHAR#(I,')','1'B,'AT END OF DEFINITION ARGUMENTS');
     W_STMT.ROW_STEP = #GET_INT_NUMBER#(SUBSTR(RECORD_STMT,I,J-I),
                                   '1'B, 'STEP '||MSG);
     END;
ELSE W_STMT.ROW_STEP = 1;
IF W_STMT.ROW_NO1 > Ø & W_STMT.ROW_NO2 = Ø
THEN W_STMT.ROW_NO2 = W_STMT.ROW_NO1;
I = #NEXT_NONBLANK#(J);
END #GET_INTERVAL#;
/*******************************************************************/
/*PROCEDURE CHECKS COLUMN NAME DEFINITION AND RETURNS COLUMN ORDER NO*/
/*******************************************************************/
#CHECK_COL_NAME#: PROCEDURE(IND_MANDATORY) RETURNS(BIN FIXED);
DCl IND_MANDATORY BIT;
DCL (J,L) BIN FIXED;
   DO J = 1 TO COLS_NO
          UNTIL( COLS(J).COL_NAME = SUBSTR(RECORD_STMT,I,L));
      L = LENGTH(COLS(J).COL_NAME);
   END;
   IF IND_MANDATORY & J > COLS_NO
   THEN CALL #SYNTAX_ERROR#('COLUMN NAME DOES NOT EXIST !');
RETURN(J);
END #CHECK_COL_NAME#;
/*******************************************************************/
/*PROCEDURE CHECKS FUNCTION NAME AND RETURNS INTERNAL FUNCTION NUMBER*/
/*******************************************************************/
#CHECK_FUNCTION_NAME#: PROCEDURE(FUNCTION_NAME) RETURNS(CHAR(1));
DCl FUNCTION_NAME CHAR(*);
DCl FUNCTION_INTERNAL_IDENT CHAR(1);
SELECT(FUNCTION_NAME);
WHEN('SUM')    FUNCTION_INTERNAL_IDENT = '1';
WHEN('MUL')    FUNCTION_INTERNAL_IDENT = '2';
WHEN('MIN')    FUNCTION_INTERNAL_IDENT = '3';
WHEN('MAX')    FUNCTION_INTERNAL_IDENT = '4';
WHEN('AVRG')   FUNCTION_INTERNAL_IDENT = '5';
WHEN('RANGE')  FUNCTION_INTERNAL_IDENT = '6';
OTHERWISE      FUNCTION_INTERNAL_IDENT = ' ';
END; /* SELECT */
RETURN(FUNCTION_INTERNAL_IDENT);
END #CHECK_FUNCTION_NAME#;
/*******************************************************************/
/* PROCEDURE CHECKS SYNTAX RIGHT FROM EQUAL SIGN AND BUILDS INTERNAL */
/* CHAIN OF STRUCTURES WHICH REPRESENT ARITHMETIC STATEMENT IN PREFIX*/
/* NOTATION.                                                         */
/*******************************************************************/
```

```
 #PROC_E#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
 DCL (P_FIRST_EL, P_LAST_EL) PTR;
 DCL SIGN CHAR(1) INIT(' ');
 /*-- SIGN OF ARGUMENT ---------------------------------------------*/
 IF CH(I) = '+' | CH(I) = '-'
 THEN DO;
      SIGN = CH(I);
      I = I + 1;
      END;
 P_FIRST_EL = #PROC_T#(P_LAST_EL);
 IF SIGN = '-'
 THEN DO;
      ALLOC ARG_OPER;
      ARG_OPER.NEXT = P_FIRST_EL;
      ARG_OPER.TYPE = 'm';
      P_FIRST_EL = P_ARG;
      END;
 IF NOT_END_STMT & (OPER='+' | OPER='-')
 THEN P_FIRST_EL = #E_LIST#(P_FIRST_EL,P_LAST_EL);
 RETURN(P_FIRST_EL);
 END #PROC_E#;
 #E_LIST#: PROCEDURE(P_FIRST_EL, P_LAST_EL) RECURSIVE RETURNS(PTR);
 DCL (P_FIRST_EL, P_LAST_EL) PTR;
*/
 ALLOC ARG_OPER;
 ARG_OPER.TYPE=OPER;
 ARG_OPER.NEXT=P_FIRST_EL;
 P_FIRST_EL=P_ARG;
 P_LAST_EL->ARG_OPER.NEXT = #PROC_T#(P_LAST_EL);
 IF NOT_END_STMT & ( OPER='+' | OPER='-')
 THEN P_FIRST_EL = #E_LIST#(P_FIRST_EL,P_LAST_EL);
 RETURN(P_FIRST_EL);
 END #E_LIST#;
 #PROC_T#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
 DCL (P_FIRST_EL, P_LAST_EL) PTR;
 P_FIRST_EL = #PROC_F#(P_LAST_EL);
 IF NOT_END_STMT & (OPER='*' | OPER='/')
 THEN P_FIRST_EL = #T_LIST#(P_FIRST_EL,P_LAST_EL);
 RETURN(P_FIRST_EL);
 END #PROC_T#;
 #T_LIST#: PROCEDURE(P_FIRST_EL,P_LAST_EL) RECURSIVE RETURNS(PTR);
 DCL (P_FIRST_EL, P_LAST_EL) PTR;
 ALLOC ARG_OPER;
 ARG_OPER.TYPE=OPER;
 ARG_OPER.NEXT=P_FIRST_EL;
 P_FIRST_EL=P_ARG;
 P_LAST_EL->ARG_OPER.NEXT=#PROC_F#(P_LAST_EL);
 IF NOT_END_STMT & (OPER='*' | OPER='/')
 THEN P_FIRST_EL = #T_LIST#(P_FIRST_EL,P_LAST_EL);
 RETURN(P_FIRST_EL);
 END #T_LIST#;
```

```
#PROC_F#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
DCL (P_FIRST_EL, P_LAST_EL) PTR;
P_FIRST_EL = #PROC_P#(P_LAST_EL);
IF NOT_END_STMT & OPER='P'
THEN P_FIRST_EL = #F_LIST#(P_FIRST_EL,P_LAST_EL);
RETURN(P_FIRST_EL);
END #PROC_F#;
#F_LIST#: PROCEDURE(P_FIRST_EL,P_LAST_EL) RECURSIVE RETURNS(PTR);
DCL (P_FIRST_EL, P_LAST_EL) PTR;
ALLOC ARG_OPER;
ARG_OPER.TYPE = OPER;
ARG_OPER.NEXT = P_FIRST_EL;
P_FIRST_EL = P_ARG;
P_LAST_EL->ARG_OPER.NEXT = #PROC_P#(P_LAST_EL);
IF NOT_END_STMT & OPER='P'
THEN P_FIRST_EL = #F_LIST#(P_FIRST_EL,P_LAST_EL);
RETURN(P_FIRST_EL);
END #F_LIST#;
#PROC_P#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
DCL (P_FIRST_EL, P_LAST_EL) PTR;
OPER = ' ';
I = #NEXT_NONBLANK#(I);
IF NOT_END_STMT
THEN IF CH(I)='('
     THEN DO;
          BZ=BZ+1;
          I = #NEXT_NONBLANK#(I+1);
          P_FIRST_EL = #PROC_E#(P_LAST_EL);
          IF CH(I)=')'
          THEN DO;
               BZ=BZ-1;
               I = #NEXT_NONBLANK#(I+1);
               END;
          ELSE CALL #SYNTAX_ERROR#('EXECEPTED ")"');
          END;
     ELSE P_FIRST_EL = #PROC_ELEMENT#(P_LAST_EL);
ELSE P_FIRST_EL = NULL;
IF INDEX('+-/*',CH(I)) > Ø
THEN IF CH(I) = '*' & CH(I+1) = '*'
     THEN DO;
          OPER = 'P';
          I = #NEXT_NONBLANK#(I+2);
          END;
     ELSE DO;
          OPER = CH(I);
          I = #NEXT_NONBLANK#(I+1);
          END;
RETURN(P_FIRST_EL);
END #PROC_P#;
#PROC_ELEMENT#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
DCL (P_FIRST_EL, P_LAST_EL) PTR;
DCL C BIN FIXED;
I = #NEXT_NONBLANK#(I);
C = #CHECK_COL_NAME#('Ø'B);
IF C <= COLS_NO
THEN DO;
     J = I + LENGTH(COLS(C).COL_NAME);
     IF VERIFY(CH(J),' Ø123456789+-/*)') = Ø | CH(J) = '5E'X
     THEN DO;
          ALLOC ARG;
          P_FIRST_EL,P_LAST_EL = P_ARG;
          CALL #GET_INTERVAL#(' ');
          ARG = W_STMT, BY NAME;
          J=(ARG.COL_NO2 - ARG.COL_NO1 + 1) *
            (ARG.ROW_NO2 - ARG.ROW_NO1 + 1) /
             ARG.ROW_STEP;
          IF j = 1  & ARG.ROW_NO1 > Ø
          THEN ARG.TYPE = '#';
          IF j > 1 & J ¬= NO_CELL_IN_RESULT
          THEN CALL #SYNTAX_ERROR# ('RANGE OF CELL IN ARGUMENT'||
                    ' IS DIFFERENT THAN IN RESULT');
          END;
     ELSE P_FIRST_EL = #PROC_ELEMENT_FUNCTION#(P_LAST_EL);
     END;
ELSE IF VERIFY(CH(I),'Ø123456789.') = Ø
     THEN DO; /*-- Get Constant ----------------------------------*/
          ALLOC ARG_CONST;
          P_FIRST_EL,P_LAST_EL = P_ARG;
          ARG_CONST.TYPE = '"';
          DO J=I+1 TO LEN_REC_STMT
             WHILE(INDEX('Ø123456789.',CH(J)) > Ø);
          END;
          GET STRING(SUBSTR(RECORD_STMT,I,J-I))
               LIST(ARG_CONST.NUMBER);
          I = #NEXT_NONBLANK#(J);
          END;
     ELSE P_FIRST_EL = #PROC_ELEMENT_FUNCTION#(P_LAST_EL);
RETURN(P_FIRST_EL);
END #PROC_ELEMENT#;
#PROC_ELEMENT_FUNCTION#: PROCEDURE(P_LAST_EL) RECURSIVE RETURNS(PTR);
DCL (P_FIRST_EL, P_LAST_EL) PTR;
DCL FUNCTION_NAME CHAR(8);
DCL FUNCTION_TYPE CHAR(1);
J = #FIND_CHAR#(I+1,'(','Ø'B,'');
/*-- Get Function Name -----------------------------------------*/
FUNCTION_NAME = SUBSTR(RECORD_STMT,I,J-I);
FUNCTION_TYPE = #CHECK_FUNCTION_NAME#(FUNCTION_NAME);
IF FUNCTION_TYPE = ' '
THEN CALL #SYNTAX_ERROR#('UNKNOWN FUNCTION OR COLUMN NAME "'||
                         FUNCTION_NAME!!'"');
J = #FIND_CHAR#(J,'(','1'B,'FOR DEFINED ARGUMENT OF FUNCTION');
I = #NEXT_NONBLANK#(J+1);
ALLOC ARG_OPER;
ARG_OPER.TYPE = FUNCTION_TYPE;
P_FIRST_EL = P_ARG;
ALLOC ARG;
P_FIRST_EL->ARG_OPER.NEXT = P_ARG;
P_LAST_EL = P_ARG;
/*-- Get Argument 1 ---------------------------------------------------*/
CALL #GET_INTERVAL#('IN ARG OF FUNCTIONS '||FUNCTION_NAME);
ARG = W_STMT, BY NAME;
IF CH(I) ¬= ')'
THEN CALL #SYNTAX_ERROR#('EXPECTED ")"');
I = #NEXT_NONBLANK#(J+1);
RETURN(P_FIRST_EL);
END #PROC_ELEMENT_FUNCTION#;
END GET_STATEMENTS;
/**********************************************************************/
/* PROCEDURE COPIES TABLE INTO WORK DATASET                           */
/**********************************************************************/
COPY_TABLE_IN_WORK_BUFF: PROCEDURE;
DCL (K, NO INIT(Ø), INDEX_ROW) BIN FIXED(31);
DCL (J, FR_POS, TO_POS) BIN FIXED;
DCL IND_IS_NUMERIC(COLS_NO) BIT;
DCL PP  PTR;
PUT SKIP EDIT(' ','=== COPYING TABLE IN WORK DATASET ',REPEAT('=',36))
             (A,SKIP,2 A);
OPEN  FILE(TABLE) INPUT;
ALLOC WORK_RECORD;
P_WR_ROOT = P_WR;
/*-- Copy data in work area ------------------------------------*/
READ FILE(TABLE) INTO(RECORD_TABLE);
INDEX_ROW = 1;
DO K=1 BY 1 WHILE(NOT_EOF_TABLE) ;
   IF INDEX_ROW <= ROWS_NO
   THEN DO;
        IF K >= ROWS(INDEX_ROW).FROM_POSITION &
           K <= ROWS(INDEX_ROW).TO_POSITION
        THEN DO;
             WORK_RECORD.ROW_NO = K;
             IND_IS_NUMERIC(*) = 'Ø'B;
             DO C = 1 TO COLS_NO;
                FR_POS = COLS(C).FROM_POSITION;
                TO_POS = COLS(C).TO_POSITION;
                IF IS_CELL_NUMERIC(C,(FR_POS),(TO_POS))
                THEN DO;
                     IND_IS_NUMERIC(C) = '1'B;
                     WORK_RECORD.DATA(C) = GET_CELL_NUMERIC
                       (SUBSTR(RECORD_TABLE,FR_POS,TO_POS-FR_POS+1),C);
                     END;
                ELSE WORK_RECORD.DATA(C)=Ø;
             END;
             IF ANY(IND_IS_NUMERIC)
```

```
                THEN DO;
                     NO = NO + 1;
                     /*-- alloc work record for new data -----------*/
                     PP_WR = P_WR;
                     ALLOC WORK_RECORD;
                     PP_WR->WORK_RECORD.NEXT = P_WR;
                     END;
                IF ¬ ALL(IND_IS_NUMERIC)
                THEN DO;
                     PUT SKIP EDIT('### WARNING: IN ROW ',K,
                         ' IN DECLARE ROWS INTERVAL: ROWS(',
                         ROWS(INDEX_ROW).FROM_POSITION,' :',
                         ROWS(INDEX_ROW).TO_POSITION,')',
                         ' IN FOLLOWING COLUMNS NO NUMERICS DATA!')
                         (A,F(7),A,F(3),A,F(3),A,SKIP,X(12),A);
                     PUT SKIP EDIT('Table data:','>',RECORD_TABLE,'<')
                         (A,SKIP,3(A));
                     CALL MARK_FIELD(IND_IS_NUMERIC);
                     END;
                END;
           ELSE DO;
                IF K > ROWS(INDEX_ROW).TO_POSITION
                THEN INDEX_ROW = INDEX_ROW + 1;
                END;
           END;
      READ FILE(TABLE) INTO(RECORD_TABLE);
   END;
   PP_WR->WORK_RECORD.NEXT = NULL;
   FREE WORK_RECORD;
   /*-- building index of work records ------------------------------*/
   NO_TABLE_RECORDS = K - 1;
   IF INDEX_ROW <= ROWS_NO
   THEN IF NO_TABLE_RECORDS < ROWS(INDEX_ROW).TO_POSITION
        THEN DO;
             ROWS(INDEX_ROW).TO_POSITION = NO_TABLE_RECORDS;
             ROWS_NO = INDEX_ROW;
             END;
   ALLOC INDEX_WORK_RECORD;
   DO P_WR = P_WR_ROOT REPEAT(WORK_RECORD.NEXT) WHILE(P_WR ¬= NULL);
     INDEX_WORK_RECORD(WORK_RECORD.ROW_NO) = P_WR;
   END;
   IF NO > Ø
   THEN PUT SKIP EDIT
   ('### GET ',NO,' NUMERICAL TABLE ROWS IN WORK AREAS')
                     (A, F(7),A);
   ELSE CALL EXECUTE_ERROR('IN DEFINED COLUMNS NO NUMERICS DATA');
   CLOSE FILE(TABLE);
   END COPY_TABLE_IN_WORK_BUFF;
   /*****************************************************************/
   /* Procedure marks specified fields                              */
   /*****************************************************************/
```

```
MARK_FIELD: PROCEDURE(IND);
DCL IND(*) BIT;
DCL L BIN FIXED;
RECORD_TABLE = REPEAT(' ',LENGTH(RECORD_TABLE));
DO C = 1 TO COLS_NO;
   IF ¬ IND(C)
   THEN DO;
        L = COLS(C).TO_POSITION - COLS(C).FROM_POSITION + 1;
        SUBSTR(RECORD_TABLE,COLS(C).FROM_POSITION,L) = REPEAT('*',L);
        END;
END;
PUT SKIP EDIT(RECORD_TABLE) (X(1),A);
PUT SKIP EDIT('MARK COLS:') (X(3),A);
DO C = 1 TO COLS_NO;
   IF ¬ IND(C)
   THEN PUT EDIT(COLS(C).COL_NAME,'/') (X(1),A);
END;
END MARK_FIELD;
/**********************************************************************/
/* Procedure recognizes field with data in table                      */
/**********************************************************************/
IS_CELL_NUMERIC: PROCEDURE(C,FP,LP) RETURNS(BIT);
DCL (C, FP, LP, L) BIN FIXED;
DCl VALID_CHAR CHAR(11) INIT('Ø123456789Ø');
DCL IND BIT;
DO FP = FP TO LP-1 WHILE(CH(FP) = ' ');
END;
IF CH(FP) = '+' | CH(FP) = '-'
THEN FP = FP + 1;
IF FP ¬= LP
THEN DO;
     DO LP = LP TO FP+1  BY -1 WHILE(CH(LP) = ' ');
     END;
     L = COLS(C).NO_OF_DECIMALS;
     IND = VERIFY(SUBSTR(RECORD_TABLE,LP-L+1,L),VALID_CHAR) = Ø;
     LP = LP - L;
     IF CH(LP) = '.' | CH(LP) = ','
     THEN DO;
          IF CH(LP) = ','
          THEN SUBSTR(VALID_CHAR,11,1) = '.';
          ELSE SUBSTR(VALID_CHAR,11,1) = ',';
          LP = LP - 1;
          END;
     IND = IND & VERIFY(SUBSTR(RECORD_TABLE,FP,LP-FP+1),VALID_CHAR)=Ø;
     END;
ELSE IND = '1'B;
RETURN (IND);
END IS_CELL_NUMERIC;
/**********************************************************************/
/* Procedure get field with data in table                             */
/**********************************************************************/
```

```
GET_CELL_NUMERIC: PROCEDURE(NUMERIC_CELL,C) RETURNS(BIN FLOAT(53));
DCL NUMERIC_CELL CHAR(*);
DCL CH(32767) CHAR(1) BASED(ADDR(NUMERIC_CELL));
DCL (LP, C, J,K) BIN FIXED;
DCL N BIN FLOAT(53);
ON CONVERSION BEGIN;
              PUT SKIP EDIT('>',NUMERIC_CELL,'<') (A);
              CALL EXECUTE_ERROR('DATA NOT NUMERIC');
              END;
/*--------------------------------------------------------------------*/
/* CONVERTS DATA FROM FORMAT WITH SEPARATED GROUP OF 3 DIGIT INTO    */
/* STANDARD NUMERICS FORMAT                                           */
/*--------------------------------------------------------------------*/
IF INDEX(NUMERIC_CELL,',')>Ø
THEN DO;
     DO LP = LENGTH(NUMERIC_CELL) TO 1 BY -1
                   WHILE(CH(LP)=' ');
     END;
     DO K=LP TO 1 BY -1 WHILE(CH(K) >= 'Ø' & CH(K) <= '9');
     END;
     IF CH(K) = ','
     THEN CH(K) = '.';
     K = K - 3;
     DO J = 1 TO K;
        IF CH(J) = '.' | CH(J) = ','
        THEN DO;
             SUBSTR(NUMERIC_CELL,J,LP-J+1) =
             SUBSTR(NUMERIC_CELL,J+1,LP-J);
             END;
     END;
     END;
GET STRING(NUMERIC_CELL||' Ø') LIST(N);
RETURN (N);
END GET_CELL_NUMERIC;
/*********************************************************************/
/* PROCEDURE WRITES ERROR IN EXECUTION                               */
/*********************************************************************/
EXECUTE_ERROR: PROCEDURE(MSG);
 DCl IND_ROW BIT;
 DCl J BIN FIXED;
 DCL MSG CHAR(*);
 IF P_WR ¬= NULL
 THEN DO;
      PUT SKIP EDIT('*** ERROR IN ROW ',WORK_RECORD.ROW_NO,' AND COL ',
                    COLS(C).COL_NAME) (A,F(7),A,A);
      DO J=1 TO COLS_NO;
         PUT SKIP EDIT(COLS(J).COL_NAME,' =',
                       WORK_RECORD.DATA(J)) (X(5),3 A);
      END;
      END;
 IF IND_TRACE & P_STMT ¬= NULL
 THEN CALL PRINT_STMT(P_STMT);
 PUT SKIP EDIT('*** ERROR ',MSG) (A);
 STOP;
END EXECUTE_ERROR;
/*******************************************************************/
/* PROCEDURE CHECKS IF ROW ORDER IS VALID                          */
/*******************************************************************/
CHECK_VALIDATE_ROW: PROCEDURE RETURNS(BIT);
DCL IND_OK BIT INIT('1'B);
DCL R BIN FIXED(31);
DCL PPS PTR;
IF STMT.ROW_NO1 = Ø
THEN DO;
     STMT.ROW_NO1 = ROWS(1).FROM_POSITION;
     IF STMT.ROW_NO2 = Ø
     THEN STMT.ROW_NO2=MIN(ROWS(ROWS_NO).TO_POSITION,NO_TABLE_RECORDS);
     END;
ELSE IF STMT.ROW_NO2 = Ø
     THEN STMT.ROW_NO2 = STMT.ROW_NO1;
R = STMT.ROW_NO1;
IND_OK = (R <= NO_TABLE_RECORDS);
DO PPS = STMT.P_ARG REPEAT(PPS->ARG.NEXT)
                        WHILE(PPS ¬= NULL & IND_OK);
   IF PPS->ARG_OPER.TYPE = ' ' |
      PPS->ARG_OPER.TYPE = 'C'
   THEN DO;
        R = PPS->ARG.ROW_NO1;
        IF R <= NO_TABLE_RECORDS
        THEN R = PPS->ARG.ROW_NO2;
        END;
   IND_OK = (R <= NO_TABLE_RECORDS);
END;
IF ¬IND_OK
THEN DO;
     PUT SKIP EDIT('*** ROW ',R,' DOES NOT EXIST IN TABLE') (A);
     IF IND_TRACE
     THEN CALL PRINT_STMT(P_STMT);
     CALL PLIRETC(8);
     END;
RETURN (IND_OK);
END CHECK_VALIDATE_ROW;
/*******************************************************************/
/* PROCEDURE CALCULATES ROWS AND COLS                              */
/*******************************************************************/
CALCULATE: PROCEDURE;
DCL N BIN FIXED INIT(Ø);
DCl R BIN FIXED(31);
ON OVERFLOW  CALL EXECUTE_ERROR('DATA OVERFLOW');
ON UNDERFLOW CALL EXECUTE_ERROR('DATA UNDERFLOW');
PUT SKIP EDIT(' ','=== CALCULATE ROWS AND COLUMNS ',REPEAT('=',39))
             (A,SKIP,2 A);
```

```
DO P_STMT = P_STMT_ROOT REPEAT(STMT.NEXT) WHILE(P_STMT ¬= NULL);
   N = N + 1;
   IF IND_TRACE
   THEN PUT SKIP EDIT(' ','-- CALCULATE STMT NO:',N,REPEAT('-',34))
                (A,SKIP,3 A,X(1),A);
   IF CHECK_VALIDATE_ROW()
   THEN CALL CALCULATE_CELLS(STMT.COL_NO1, STMT.COL_NO2,
             STMT.ROW_NO1, STMT.ROW_NO2, STMT.ROW_STEP, STMT.P_ARG);
   ELSE PUT SKIP EDIT('*** STATEMENT NO:',N,' IGNORED') (A,F(3),A);
END;
/*******************************************************************/
/* PROCEDURE CALCULATES COLUMNS FOR EACH REPEATABLE ROW_NO          */
/*******************************************************************/
CALCULATE_CELLS: PROCEDURE(COL1, COL2, ROW1, ROW2, ROW_STEP, P_ARG);
DCL (COL1, COL2, C, ROW_STEP) BIN FIXED;
DCL (ROW1, ROW2, R, N INIT(Ø)) BIN FIXED(31);
DCL (P_ARG, PP_WR) PTR;
DCl REZ BIN FLOAT(53);
IF IND_TRACE
THEN PUT SKIP EDIT('--> CALCULATE CELLS FROM ',COLS(COL1).COL_NAME,
                             ROW1,' TO ',COLS(COL2).COL_NAME,
                             ROW2) (A,A,F(7));
DO R = ROW1 TO ROW2 BY ROW_STEP;
   PP_WR = INDEX_WORK_RECORD(R);
   IF PP_WR ¬= NULL
   THEN DO;
        DO C = COL1 TO COL2;
           REZ = CALCULATE_CELL(P_ARG,C - COL1,ROW1,R - ROW1);
           IF PP_WR->WORK_RECORD.DATA(C) ¬= REZ | REZ = Ø
           THEN DO;
                PP_WR->WORK_RECORD.IND_CHANGE(C) = '1'B;
                PP_WR->WORK_RECORD.DATA(C) = REZ;
                END;
        END;
        N = N + 1;
        END;
END;
IF IND_TRACE
THEN PUT SKIP EDIT('--> TOTAL CELLS IN ',N,'ROWS AND',COL2 - COL1 + 1,
                   'COLUMNS') (A,X(1),F(7),X(1),A,X(1),F(3),X(1),A);
END CALCULATE_CELLS;
/*******************************************************************/
/* PROCEDURE CALCULATES CELL                                        */
/*******************************************************************/
CALCULATE_CELL: PROCEDURE(POK,OFFSET_C,ROWØ,OFFSET_R) RECURSIVE
                RETURNS(BIN FLOAT(53));
DCL POK PTR;
DCL OFFSET_C BIN FIXED;
DCL (ROWØ, OFFSET_R) BIN FIXED(31);
DCL (REZ, ARG) BIN FLOAT(53);
IF POK ¬= NULL
```

```
THEN DO;
     P_ARG=POK;
     IF ARG_OPER.TYPE = 'm'
     THEN REZ = - CALCULATE_CELL(ARG.NEXT,OFFSET_C,ROWØ,OFFSET_R);
     ELSE
     IF ARG_OPER.TYPE = '+' |
        ARG_OPER.TYPE = '-' |
        ARG_OPER.TYPE = '/' |
        ARG_OPER.TYPE = '*' |
        ARG_OPER.TYPE = 'P'
     THEN DO;
          REZ = CALCULATE_CELL(ARG.NEXT,OFFSET_C,ROWØ,OFFSET_R);
          ARG = CALCULATE_CELL(ARG.NEXT,OFFSET_C,ROWØ,OFFSET_R);
          SELECT(POK->ARG_OPER.TYPE);
          WHEN('+') REZ = REZ + ARG;
          WHEN('-') REZ = REZ - ARG;
          WHEN('*') REZ = REZ * ARG;
          WHEN('/') IF ARG ¬= Ø
                    THEN REZ = REZ / ARG;
                    ELSE CALL EXECUTE_ERROR('ZERO DEVIDE');
          WHEN('P') REZ = POWER(REZ,ARG);
          OTHERWISE;
          END; /* SELECT */
          END;
     ELSE DO;
          SELECT(ARG_OPER.TYPE);
          WHEN('#') REZ = CELL(ARG.COL_NO1, ARG.ROW_NO1);
          WHEN(' ') IF ARG.ROW_NO1 > Ø
                    THEN REZ = CELL(ARG.COL_NO1+OFFSET_C,
                                    ARG.ROW_NO1+OFFSET_R);
                    ELSE REZ = CELL(ARG.COL_NO1+OFFSET_C,
                                    ROWØ+OFFSET_R);
          WHEN('"') REZ = ARG_CONST.NUMBER;
          WHEN('1','2','3','4','5','6')
                    REZ = CALC_FUNCTION(ARG_OPER.TYPE, ROWØ, OFFSET_R,
                                        ARG_OPER.NEXT);
          OTHERWISE;
          END; /* SELECT */
          END;
     END;
ELSE REZ = Ø;
RETURN(REZ);
END CALCULATE_CELL;
/*******************************************************************/
/* PROCEDURE CALCULATES FUNCTION                                   */
/*******************************************************************/
CALC_FUNCTION: PROCEDURE(FUNCTION,ROWØ,OFFSET_R,POK)
               RETURNS(BIN FLOAT(53));
DCL FUNCTION CHAR(1);
DCL (ROWØ, OFFSET_R, R1, R2) BIN FIXED(31);
DCL POK PTR;
DCL REZ BIN FLOAT(53);
P_ARG=POK;
IF ARG.ROW_NO1 > Ø & ARG.ROW_NO2 > Ø
THEN DO;
     R1 = ARG.ROW_NO1;
     R2 = ARG.ROW_NO2;
     END;
ELSE R1,R2 = ROWØ + OFFSET_R;
SELECT(FUNCTION);
WHEN('1') REZ = SUM_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
WHEN('2') REZ = MUL_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
WHEN('3') REZ = MIN_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
WHEN('4') REZ = MAX_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
WHEN('5') REZ = AVRG_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
WHEN('6') REZ = RANGE_CELL(ARG.COL_NO1, ARG.COL_NO2,
          R1, R2, ARG.ROW_STEP);
OTHERWISE;
END; /* SELECT */
RETURN(REZ);
END CALC_FUNCTION;
/*-- INTERNAL Functions ----------------------------------------*/
CELL: PROCEDURE(C,R) RETURNS(BIN FLOAT(53));
   DCL C BIN FIXED;
   DCL R BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   PP_WR = INDEX_WORK_RECORD(R);
   IF PP_WR ¬= NULL
   THEN REZ = PP_WR->WORK_RECORD.DATA(C);
   ELSE DO;
        PUT SKIP EDIT(COLS(C).COL_NAME,R) (A,F(7),X(1));
        CALL EXECUTE_ERROR('ROW DOES NOT EXIST OR CELL ISNT NUMERIC!');
        END;
   RETURN(REZ);
END CELL;
POWER: PROCEDURE(REZ,ARG) RETURNS(BIN FLOAT(53));
   DCL (REZ,ARG) BIN FLOAT(53);
   IF REZ > Ø | (REZ=Ø & ARG>Ø)
   THEN REZ = REZ**ARG;
   ELSE DO;
        PUT SKIP EDIT('X=',REZ,'Y=',ARG) (A,A,SKIP,A,A);
        IF REZ < Ø
        THEN CALL EXECUTE_ERROR('X < Ø in X**Y');
        IF REZ = Ø & ARG <= Ø
        THEN CALL EXECUTE_ERROR('X = Ø and Y <= Ø in X**Y');
        END;
   RETURN(REZ);
```

```
END POWER;
SUM_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R) BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   REZ = Ø;
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
              REZ = REZ + PP_WR->WORK_RECORD.DATA(C);
           END;
   END;
   RETURN(REZ);
END SUM_CELL;
MUL_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R) BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   REZ = 1;
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
              REZ = REZ * PP_WR->WORK_RECORD.DATA(C);
           END;
   END;
   RETURN(REZ);
END MUL_CELL;
MIN_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R) BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   REZ = 99999999999;
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
              IF REZ > PP_WR->WORK_RECORD.DATA(C)
              THEN REZ = PP_WR->WORK_RECORD.DATA(C);
           END;
   END;
   RETURN(REZ);
END MIN_CELL;
MAX_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R) BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   REZ = Ø;
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
              IF REZ < PP_WR->WORK_RECORD.DATA(C)
              THEN REZ = PP_WR->WORK_RECORD.DATA(C);
           END;
   END;
   RETURN(REZ);
END MAX_CELL;
AVRG_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R,N INIT(Ø)) BIN FIXED(31);
   DCL REZ BIN FLOAT(53);
   REZ = Ø;
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
              REZ = REZ + PP_WR->WORK_RECORD.DATA(C);
              N = N + 1;
           END;
   END;
   REZ = REZ / N;
   RETURN(REZ);
END AVRG_CELL;
RANGE_CELL: PROCEDURE(C1,C2,R1,R2,R_STEP) RETURNS(BIN FLOAT(53));
   DCL TYPE CHAR(1);
   DCL (C1,C2,C,R_STEP) BIN FIXED;
   DCL (R1,R2,R) BIN FIXED(31);
   DCL (REZ,MIN_R INIT(99999999999), MAX_R INIT(Ø)) BIN FLOAT(53);
   DO R = R1 TO R2 BY R_STEP;
      PP_WR = INDEX_WORK_RECORD(R);
      IF PP_WR ¬= NULL
      THEN DO C = C1 TO C2;
           IF MIN_R > PP_WR->WORK_RECORD.DATA(C)
           THEN MIN_R = PP_WR->WORK_RECORD.DATA(C);
           IF MAX_R < PP_WR->WORK_RECORD.DATA(C)
           THEN MAX_R = PP_WR->WORK_RECORD.DATA(C);
           END;
   END;
   REZ = MAX_R - MIN_R;
   RETURN(REZ);
END RANGE_CELL;
END CALCULATE;
/*******************************************************************/
/* PROCEDURE CHECKS DATA FROM TABLE                                */
/*******************************************************************/
```

```
CHECK_DATA_IN_TABLE: PROCEDURE;
DCL (IND, IS_RECORD_EXIST, NO_DIFERENT_EXIST(COLS_NO)) BIT;
DCL (J, L, BRG INIT(Ø)) BIN FIXED,
    K BIN FIXED(31);
DCL (FR_POS, TO_POS) BIN FIXED;
DCL ROUND_ERROR(COLS_NO) BIN FLOAT(53) INIT((COLS_NO)Ø.51);
DCL TABLE_DATA(COLS_NO)  BIN FLOAT(53);
PUT SKIP EDIT(' ','=== CHECKING DATA IN TABLE ',REPEAT('=',41))
             (A,SKIP,2 A);
/*-- approximation of calculation error and rounding error --------*/
/*-- Ø.5 * (1Ø ** NO_DECIMALS) - is ????? ERROR AND -------------*/
/*-- 1 * (1Ø ** (NO_DECIMALS+1)) - is ??????ATED ERROR ----------*/
DO C=1 TO COLS_NO;
   ROUND_ERROR(C) = ROUND_ERROR(C) / (1Ø ** COLS(C).NO_OF_DECIMALS);
END;
NOT_EOF_TABLE='1'B;
OPEN FILE(TABLE) INPUT;
READ FILE(TABLE) INTO(RECORD_TABLE);
DO K = 1 BY 1 WHILE(NOT_EOF_TABLE);
   P_WR = INDEX_WORK_RECORD(K);
   IF P_WR ¬= NULL
   THEN DO;
        IND = 'Ø'B;
        DO C = 1 TO COLS_NO;
           IF P_WR->WORK_RECORD.IND_CHANGE(C)
           THEN DO;
                FR_POS     = COLS(C).FROM_POSITION;
                TO_POS     = COLS(C).TO_POSITION;
                IF IS_CELL_NUMERIC(C,(FR_POS),(TO_POS))
                THEN DO;
                     IND = '1'B;
                     TABLE_DATA(C)=GET_CELL_NUMERIC
                       (SUBSTR(RECORD_TABLE,FR_POS,TO_POS-FR_POS+1),C);
                     END;
                ELSE TABLE_DATA(C)=Ø;
                END;
           ELSE TABLE_DATA(C) = WORK_RECORD.DATA(C);
        END;
        IF IND
        THEN DO;
             NO_DIFERENT_EXIST(*) = '1'B;
             DO C=1 TO COLS_NO;
                IF ABS(TABLE_DATA(C) - WORK_RECORD.DATA(C)) >=
                   ROUND_ERROR(C)
                THEN NO_DIFERENT_EXIST(C) = 'Ø'B;
             END;
             IF ¬ ALL(NO_DIFERENT_EXIST)
             THEN DO;
                  BRG=BRG+1;
                  PUT SKIP EDIT('*** DIFFERENCE IN ROW:',
                      WORK_RECORD.ROW_NO) (A,F(7));
                  PUT SKIP EDIT('Table data:',
                      '>',RECORD_TABLE,'<') (A,SKIP,3(A));
                  CALL FORMAT_OUTPUT_ROW(WORK_RECORD.DATA);
                  PUT SKIP EDIT('CALCULATED DATA:',
                      '>',RECORD_TABLE,'<') (A,SKIP,3(A));
                  /*-- Mark wrong data with asterix ----------*/
                  CALL MARK_FIELD(NO_DIFERENT_EXIST);
                  END;
             END;
        END;
   READ FILE(TABLE) INTO(RECORD_TABLE);
END;
CLOSE FILE(TABLE);
IF BRG>Ø
THEN DO;
     PUT SKIP EDIT('### DATA IS NOT CORRECT IN ',BRG,' ROWS') (A);
     CALL PLIRETC(8);
     END;
ELSE PUT SKIP EDIT('### ALL DATA IS CORRECT !!!') (A);
END CHECK_DATA_IN_TABLE;
/*******************************************************************/
/* FORMAT  DATA FOR OUTPUT ROW                                     */
/*******************************************************************/
FORMAT_OUTPUT_ROW: PROCEDURE(NUMERIC_DATA);
DCL NUMERIC_DATA(*) BIN FLOAT(53),
   (NO INIT(Ø), L) BIN FIXED(31);
DCL REZULT CHAR(3Ø) INIT(' ');
ON SIZE BEGIN;
        PUT SKIP EDIT('*** ERROR - COL ',COLS(C).COL_NAME,
                      ' DEFINITIONS IS:', COLS(C).FROM_POSITION,' :',
                      COLS(C).TO_POSITION,'AND RESULT IS:',REZULT)
                     (3 A,F(7),A,F(7),SKIP,X(4),A,A);
        CALL EXECUTE_ERROR('RESULT OVERFLOWS LENGTH OF COLS');
        END;
/*-- Converting from standard numeric format into format with ----*/
/*-- separated group of 3 digit and write data only if data > Ø --*/
/*-- or output cell is not blank ----------------------------------*/
DO C = 1 TO COLS_NO;
   IF WORK_RECORD.IND_CHANGE(C)
   THEN DO;
        L =COLS(C).TO_POSITION - COLS(C).FROM_POSITION + 1;
        PUT STRING(REZULT) EDIT(NUMERIC_DATA(C))
                          (F(LENGTH(REZULT),COLS(C).NO_OF_DECIMALS));
        IF COLS(C).SEPARATOR ¬= ' ' & COLS(C).NO_OF_DECIMALS > Ø
        THEN DO;
             J=LENGTH(REZULT) - COLS(C).NO_OF_DECIMALS;
             SUBSTR(REZULT,J,1) = COLS(C).DECIMAL_POINT;
             DO J=J-4 TO 2 BY -4;
                IF SUBSTR(REZULT,1,1) = ' ' & SUBSTR(REZULT,J,1) ¬= ' '
                THEN DO;
                     SUBSTR(REZULT,1,J-1) = SUBSTR(REZULT,2,J-1);
```

```
                   SUBSTR(REZULT,J,1) = COLS(C).SEPARATOR;
                   END;
             END;
             END;
        IF SUBSTR(REZULT,LENGTH(REZULT)-L,1) ¬= ' '
        THEN SIGNAL SIZE;
        SUBSTR(RECORD_TABLE,COLS(C).FROM_POSITION,L) =
        SUBSTR(REZULT,31 - L,L);
     END;
END;
END FORMAT_OUTPUT_ROW;
/*******************************************************************/
/* PROCEDURE MOVES BACK DATA FROM WORK DATASET TO TABLE            */
/*******************************************************************/
WRITE_COUNT_DATA_INTO_TABLE: PROCEDURE;
DCL (NO INIT(Ø), K) BIN FIXED(31);
ON ERROR BEGIN;
         ON ERROR SNAP SYSTEM;
         PUT SKIP EDIT('*** DATA HAVE BEEN CHANGED IN ',NO,
                      ' ROWS TILL THE BREAK.' ) (A,F(7),A);
         END;
PUT SKIP EDIT(' ','=== WRITING CALCULATED DATA INTO TABLE ',
               REPEAT('=',31)) (A,SKIP,2 A);
NOT_EOF_TABLE='1'B;
OPEN FILE(TABLE) UPDATE;
READ FILE(TABLE) INTO(RECORD_TABLE);
DO K = 1 BY 1 WHILE(NOT_EOF_TABLE);
   P_WR = INDEX_WORK_RECORD(K);
   IF P_WR ¬= NULL
   THEN IF ANY(WORK_RECORD.IND_CHANGE(*))
        THEN DO;
             CALL FORMAT_OUTPUT_ROW(WORK_RECORD.DATA);
             NO=NO+1;
             REWRITE FILE(TABLE) FROM(RECORD_TABLE);
             END;
   READ FILE(TABLE) INTO(RECORD_TABLE);
END;
CLOSE FILE(TABLE);
PUT SKIP EDIT('### TOTAL - DATA IN ',NO,' ROWS ARE CALCULATED !')
             (A,F(7),A);
END WRITE_COUNT_DATA_INTO_TABLE;
/*******************************************************************/
/* PRINT STATEMENTS IF PARAMETER TRACE IS SELECTED                 */
/*******************************************************************/
PRINT_STMT: PROCEDURE(PS);
DCL (PS, PPS, PR) PTR;
DCL FUNC CHAR(1Ø) VAR;
PUT SKIP EDIT('-- STMT DEFINITION ------------------------') (A);
PUT SKIP EDIT('>',COLS(PS->STMT.COL_NO1).COL_NAME,
              PS->STMT.ROW_NO1,' :', PS->STMT.ROW_NO2, '< = ')
              (2(A),2(F(7),A));
```

```
 DO PPS = PS->STMT.P_ARG REPEAT(PPS->ARG.NEXT) WHILE(PPS ¬= NULL);
    SELECT(PPS->ARG_OPER.TYPE);
    WHEN('m') FUNC = 'SIGN -';
    WHEN(' ') FUNC = 'CELLS';
    WHEN('#') FUNC = 'CELL';
    WHEN('P') FUNC = '**';
    WHEN('1') FUNC = 'SUM';
    WHEN('2') FUNC = 'MULTIPLE';
    WHEN('3') FUNC = 'MIN';
    WHEN('4') FUNC = 'MAX';
    WHEN('5') FUNC = 'AVRG';
    WHEN('6') FUNC = 'RANGE';
    OTHERWISE FUNC = PPS->ARG_OPER.TYPE;
    END; /* SELECT */
    SELECT(PPS->ARG_OPER.TYPE);
    WHEN('#',' ')
              PUT SKIP EDIT(FUNC,
                  COLS(PPS->ARG.COL_NO1).COL_NAME,
                  PPS->ARG.ROW_NO1, ' :',
                  COLS(PPS->ARG.COL_NO2).COL_NAME,
                  PPS->ARG.ROW_NO2, ' ROW STEP:',
                  PPS->ARG.ROW_STEP)
                  (X(1),A,X(1),A,F(7),A,A,F(7),A);
    WHEN('"') PUT SKIP EDIT(' >"',PPS->ARG_CONST.NUMBER,'"') (A);
    OTHERWISE PUT SKIP EDIT(' >',FUNC,'<') (A,X(1));
    END; /* SELECT */
 END;
 END PRINT_STMT;
 END CALCTAB;
```

JCL TO EXECUTE CALCTAB

```
//useridC     JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID,MSGLEVEL=(2,Ø)
//CALCTAB   EXEC PGM=CALCTAB
//STEPLIB   DD DSN=userid.USER.LOAD,DISP=SHR
//TABLE     DD DSN=userid.TEST.TABLE,DISP=OLD
//SYSPRINT  DD SYSOUT=X
//SYSIN     DD *
* SAMPE DEFINITION OF ROWS AND COLUMNS
 COLUMNS   A=(25:33,Ø)  B=(35:46,2) C=(48:6Ø,2,',','.') D=(64:7Ø,2);
 ROWS      (5:11);
* SAMPLE FORMULAS
 C = B * A;
 A11 = SUM(A5:A9);
 C11 = SUM(C5:C9);
 D5:D11 = C5:C11   / C11 * 1ØØ ;
/*
//
```

EXAMPLES

Input table

```
=================================================================  001
|No.|        A      |     B       |      C    |     D      | E         |  002
|==================================================================|  003
| 1.| 123.45        | 0.9875      |           |            |           |  004
|---|--------------|------------|-----------|------------|-----------|  005
| 2.| 27.89        | 14.8765      |           |            |           |  006
|---|--------------|------------|-----------|------------|-----------|  007
| 3.| 92.71        | 28.1734      |           |            |           |  008
|---|--------------|------------|-----------|------------|-----------|  009
| 4.| 45.87        | 18.5475      |           |            |           |  010
|---|--------------|------------|-----------|------------|-----------|  011
| 5.| 25.46        | 34.9867      |           |            |           |  012
|---|--------------|------------|-----------|------------|-----------|  013
| 6.| 243.19       | 2.1785       |           |            |           |  014
|---|--------------|------------|-----------|------------|-----------|  015
| 7.| 17.99       | 57.1372      |           |            |           |  016
|---|--------------|------------|-----------|------------|-----------|  017
| 8.| 123.86       | 15.0001      |           |            |           |  018
|---|--------------|------------|-----------|------------|-----------|  019
| 9.| 3.00        | 188.9876     |           |            |           |  020
|---|--------------|------------|-----------|------------|-----------|  021
|10.| 15.05        | 35.1435      |           |            |           |  022
|==================================================================|  023
|   |             |            |           |            |           |  024
|==================================================================|  025
```

Job for invoking spreadsheet

```
//useridC JOB CLASS=A,MSGCLASS=X,NOTIFY=&SYSUID,MSGLEVEL=(2,0)
//CALCTAB EXEC PGM=CALCTAB
//STEPLIB DD DSN=userid.USER.LOAD,DISP=SHR
//TABLE DD DSN=userid.TEST.TAB2,DISP=OLD
//SYSPRINT DD SYSOUT=X
//SYSIN DD *
TRACE;
* DEFINITIONS OF COLUMNS AND ROWS
COLUMNS A=(07:16,2) B=(22:32,4) C=(35:44,2) D=(47:57,2) E=(60:69,4);
ROWS (04:24);
* DEFINITIONS OF ARITMETIC STATEMENETS
C = A * B - A + 100 ;
D = (RANGE(A4:A22:2) - B) / C;
E = SUM(A:D);
A24 = RANGE(A4:A22:2);
B24 = MIN(B4:B22:2);
C24 = MAX(C4:C22:2);
D24 = AVRG(D4:D22:2);
```

```
E24 = (-MIN(A4:D24)+MAX(A4:D24) - RANGE(A4:D24)) / (AVRG(A4:D24) *
SUM(D4:D22:2));
/*
// ØØ22ØØØ
```

Result table

```
=====================================================================  ØØ1
|No.|        A      |     B       |      C     |     D       | E          |  ØØ2
|=====================================================================|  ØØ3
| 1.| 123.45        | Ø.9875      | 98.46      | 2.43        | 225.3239   |  ØØ4
|---|---------------|-------------|------------|-------------|------------|  ØØ5
| 2.| 27.89        | 14.8765      | 487.Ø2     | Ø.46        | 53Ø.2447   |  ØØ6
|---|---------------|-------------|------------|-------------|------------|  ØØ7
| 3.| 92.71        | 28.1734      | 2619.25    | Ø.Ø8        | 274Ø.21Ø3 |  ØØ8
|---|---------------|-------------|------------|-------------|------------|  ØØ9
| 4.| 45.87        | 18.5475      | 9Ø4.9Ø     | Ø.24        | 969.5663   |  Ø1Ø
|---|---------------|-------------|------------|-------------|------------|  Ø11
| 5.| 25.46        | 34.9867      | 965.3Ø     | Ø.21        | 1Ø25.96Ø7 |  Ø12
|---|---------------|-------------|------------|-------------|------------|  Ø13
| 6.| 243.19        | 2.1785      | 386.6Ø     | Ø.62        | 632.5836   |  Ø14
|---|---------------|-------------|------------|-------------|------------|  Ø15
| 7.| 17.99         | 57.1372     | 11Ø9.91    | Ø.16        | 1185.2ØØ4 |  Ø16
|---|---------------|-------------|------------|-------------|------------|  Ø17
| 8.| 123.86        | 15.ØØØ1     | 1834.Ø5    | Ø.12        | 1973.Ø353 |  Ø18
|---|---------------|-------------|------------|-------------|------------|  Ø19
| 9.| 3.ØØ         | 188.9876     | 663.96     | Ø.Ø8        | 856.Ø275   |  Ø2Ø
|---|---------------|-------------|------------|-------------|------------|  Ø21
|1Ø.| 15.Ø5         | 35.1435     | 613.86     | Ø.33        | 664.3872   |  Ø22
|=====================================================================|  Ø23
|   | 24Ø.19       | Ø.9875       | 2619.25    | Ø.47        | Ø.ØØØØ     |  Ø24
|=====================================================================|  Ø25
```

Emina Spasic and Dragan Nikolic
Systems Programmers
Postal Savings Bank (Yugoslavia)

Comparing PDS files

The following utility was developed to compare the contents of two PDS files containing same-name members. The purpose is to detect which members are different or which members are missing from the second PDS, without going into full details about those differences, like the standard ISPF Compare utility can do. My goal is not a complete member comparison, but only to know which members are different.

The program lists both PDS directories, and then proceeds to compare member by member. If a difference is found, it issues a message stating that a difference was found in line X and proceeds to the next member. Also, if a member exists in one PDS but not in the other, it signals the fact.

You can limit the range of the members to search by entering the starting characters of the member names in the appropriate field in the input panel. For example, you can limit the comparison to members starting with ‘A*’, or whatever. No final asterisk is necessary, but no harm comes from it, either.

The result of the search can be displayed in two ways: you can choose an output file (which will be automatically browsed at the end of the process), or write directly to the screen by choosing none. If you choose a file, and that file already exists, you will be warned of the fact, and you can confirm that you want to reuse it, or you can choose another name. If the file does not exist, it will be created as a sequential 80-byte file.

This utility consists of a REXX EXEC and an associated ISPF panel. The EXEC accepts as an optional parameter the name of the first PDS. If you supply this parameter, the second PDS name is automatically assumed to be equal to the first when the panel is displayed. This is because often the name of both PDSs is relatively similar, so it is generally easier to correct the first name to the second than to write it in full. If you do not enter a parameter, the ISPF panel will be empty, and you have to type in both names.

The output will look as follows. In the case that no differences are found, a single message ‘No differences were found’ is displayed.

```
 >>> AC54ARS  does not exist in second PDS
 >>> A124ØCA  does not exist in second PDS
 --- A2ØØART  Different number of lines
 --- A2ØØØAS  Difference found at line 15
 --- A21ØØCB  Difference found at line 21
 <<< A2344CA  exists in second PDS but not in first
 <<< A234CZ1  exists in second PDS but not in first
```

The entry panel looks as follows:

```
.----------- Differences between PDS datasets -----------.
|                                                         |
|    First PDS..: PRGT452.COB1.TST12                      |
|                                                         |
|    Second PDS.: PRGT452.COB1.TST12.BØ1Ø9                    |
|                                                             |
|    Optional member pattern.: A*                             |
|                                                             |
|    Output file: PRGT452.DIFFCO                              |
|    Mark R to reuse existing file.: R                        |
|                                                             |
'-------------------------------------------------------------'
```

DIFF REXX SOURCE CODE

```
/* REXX *=========================================================*/
/*  DIFF - Compare same name members in two PDS files.            */
arg dsn1 .
call display_panel
call list_members_1
call compare_members
call list_members_2
exit
/*                          Subroutines                             */
display_panel:
 panel_field  = "DSN1"
 if dsn1 <> "" then do
    dsn2 = dsn1
    panel_field = "DSN2"
 end
 do disp = Ø
    address ispexec
    'addpop row(1) column(1)'
    'display panel(diffe) cursor('panel_field')'
    if rc = 8 then exit
    'rempop'
    address tso
    if dsn2 = dsn1 then do
       MSG = "Datasets must be different"
       panel_field = "DSN2"
       iterate disp
    end
    xx = listdsi("'"dsn1"'")
    if sysreason <> Ø | sysdsorg <> "PO" then do
       MSG = "Dataset must be a PDS"
       panel_field = "DSN1"
       iterate disp
    end
    xx = listdsi("'"dsn2"'")
    if sysreason <> Ø | sysdsorg <> "PO" then do
       MSG = "Dataset must be a PDS"
       panel_field = "DSN2"
       iterate disp
    end
    if outfile <> "" then do
```

```
       xx = listdsi("'"outfile"'")
       if sysreason <> 0 then do
          call alloc_outfile "NEW"
       end
       else do
          if R = 'R' then do
             call alloc_outfile "OLD"
          end
          else do
             MSG="Outfile already exists. Mark reuse or specify other"
             panel_field = "OUTFILE"
             iterate disp
          end
       end
    end
    if filespec <>"" then filespec = strip(filespec,,"*")
    leave disp
 end
return
list_members_1:
 xx = outtrap(list.)
 address TSO "LISTDS ('"dsn1"') MEMBERS ST"
 xx = outtrap(off)
 if list.0 = 0 then do
    say "Pds" pds1 "contains no members"
    exit
 end
 m = 0
 mem = 0
 do z = 1 to list.0
    list.z = space(list.z,0)
    list.z = strip(list.z,,"-")
    if list.z = "MEMBERS" then do
       mem = 1
       iterate z
    end
    if mem = 1 then do
       m = m + 1
       member.m = space(list.z,0)
    end
 end
 drop list.
return
compare_members:
 text = ""
 do z = 1 to m
    name = member.z
    if left(name,length(filespec)) <> filespec then iterate z
    member1 = dsn1"("name")"
    member2 = dsn2"("name")"
    if sysdsn("'"member2"'") <> "OK" then do
       name = left(name,8)
```

```
         text = ">>>" name "does not exist in second PDS"
         call output_text
         iterate z
      end
      call alloc_member member1 ddmem1
      call alloc_member member2 ddmem2
      call read_member_contents
      if result > Ø then do
         name = left(name,8)
         text = "---" name "Difference found at line" result
         call output_text
         iterate z
      end
      if result = -1 then do
         name = left(name,8)
         text = "---" name "Different number of lines"
         call output_text
         iterate z
      end
   end
   if text = "" then do
      say "No differences were found"
   end
   else do
      if outfile <> "" then do
         execio Ø diskw dife "(" finis
         address ispexec "BROWSE DATASET('"outfile"')"
      end
   end
   'free dd(dife)'
  return
  read_member_contents:
   do xlinha = 1 to 2ØØØØ
      retcod = Ø
      execio 1 diskr ddmem1
      rc1 = rc
      execio 1 diskr ddmem2
      rc2 = rc
      select
         when rc1 = Ø & rc2 = Ø then do
            parse pull lin1
            parse pull lin2
            if lin1 <> lin2 then do
               retcod = xlinha
               leave xlinha
            end
         end
         when rc1 <> rc2 then do
            retcod = -1
            leave xlinha
         end
         when rc1 = rc2 & rc1 <> Ø then do
```

```
          retcod = Ø
          leave xlinha
       end
    end
 end
 dropbuf
 execio Ø diskr ddmem1 "(" finis
 execio Ø diskr ddmem2 "(" finis
return retcod
list_members_2:
 xx = outtrap(list.)
 address TSO "LISTDS ('"dsn2"') MEMBERS ST"
 xx = outtrap(off)
 if list.Ø = Ø then do
    say "Pds" pds2 "contains no members"
    exit
 end
 mem = Ø
 do z = 1 to list.Ø
    list.z = space(list.z,Ø)
    list.z = strip(list.z,,"-")
    if list.z = "MEMBERS" then do
       mem = 1
       iterate z
    end
    if mem = 1 then do
       member1 = dsn1"("list.z")"
       if sysdsn("'"member1"'") <> "OK" then do
          name = left(list.z,8)
          text = "<<<" name "exists in second PDS but not in first"
          call output_text
       end
    end
 end
return
output_text:
 if outfile = "" then say text
 else do
    queue text
    execio 1 diskw dife
 end
return
alloc_member:
 zz = msg(off)
 arg fic ddname
 "free dd("ddname")"
 "alloc da('"fic"') dd("ddname") shr"
 if rc <> Ø then do
    say "Error allocating" fic
    exit
 end
return
```

```
alloc_outfile:
 arg type
 zz = msg(off)
 "free dd(dife)"
 if type = "NEW" then do
    "alloc da('"outfile"') dd(dife) new reuse blksize(8000),
           lrecl(80) recfm(f,b) dsorg(ps) space(1 1) tracks"
    rc1 = rc
 end
 else do
    "alloc da('"outfile"') dd(dife) shr"
    rc1 = rc
 end
 if rc1 <> 0 then do
    say "Error allocating" ficout
    exit
 end
return
```

DIFF PANEL SOURCE CODE

```
)ATTR
  _ TYPE(INPUT) CAPS(ON) JUST(LEFT)  COLOR(PINK)
  ? TYPE(INPUT) CAPS(ON) JUST(LEFT)  COLOR(RED)
  % TYPE(TEXT)   INTENS(HIGH) SKIP(ON) COLOR(YELLOW)
  + TYPE(TEXT)   INTENS(HIGH) SKIP(ON) COLOR(WHITE)
  $ TYPE(TEXT)   INTENS(HIGH) SKIP(ON) COLOR(BLUE)
  * TYPE(OUTPUT) INTENS(HIGH) SKIP(ON) COLOR(WHITE) CAPS(OFF)
)BODY WINDOW(65,14)
+
%   First PDS..:_DSN1                                 +
%
%   Second PDS.:_DSN2                                 +
%
%   Optional member pattern.:_filespec+
%
%
$   Output file:?OUTFILE                              +
$   Mark R to reuse existing file.:?R+
%
%  *MSG
)INIT
&ZWINTTL = 'Differences between PDS datasets'
)PROC
VER(&DSN1,NONBLANK,dsname)
VER(&DSN2,NONBLANK,dsname)
)END
```

MVS news

MegaCryption Labs have announced Version 5.2 of MegaCryption/MVS, a mainframe-based encryption tool.

With this new version, MegaCryption supports both the OpenPGP standard and IBM's cryptographic coprocessor.

The major enhancements of this new release are:

- The RSA algorithm is supported (key generation, encryption, and digital signature).
- RSA keys can be managed by ICSF and the crypto coprocessor, which translates into more speed and more security. The crypto coprocessor can be used to process a PGP or GnuPG file. PGP or GnuPG keys can be managed by ICSF.
- Different modes combining hardware and software encryption are available. There is full hardware encryption (for both symmetric and asymmetric encryption) or mixed modes (hardware symmetric or asymmetric encryption).
- Improvements for PGP- or GnuPG-encrypted datasets (speed, no size limit, support for signatures embedded into a PGP/GnuPG-encrypted file).
- Decompression of PGP- or GnuPG-encrypted datasets.

For further information contact:
ASPG, 3185 Horseshoe Drive South, Naples, FL 34104-6138, USA.
Tel: (239) 649-1548.
URL: http://www.aspg.com/megacrypt.htm.

* * *

Diversified Software has announced the availability of JOB/SCAN Release 6.1.2F, its JCL validation and change product, promising error-free production JCL. This new release includes an enhancement to the existing ESP Workload Manager scheduler interface from Cybermation.

With the enhanced interface, the product invokes ESP simulation to resolve ESP variables whenever JCL is scanned in the edit macro environment, enabling users to scan JCL being developed for ESP without actually saving the JCL into the ESP environment.

There's also an option now to display the structured JCL listing reports produced by a foreground validation in View mode, which supports the use of edit macros.

Over 135 specific changes and enhancements have been implemented, including over 60 interface enhancements to applications such as Control-M, Tivoli Workload Scheduler (OPC/ESA), CA-Librarian, and ASG-Zara, as well as support for changes to ISPF, OS/390, z/OS, JES3, and SMS.

For further information contact:
Diversified Software Systems, 18635 Sutter Blvd, Morgan Hill, CA 95037, USA.
Tel: (408) 778 9914.
URL: http://www.diversifiedsoftware.com/html/jobscan.htm.

* * *

IBM has announced Version 6.0 of its VisualAge Smalltalk Enterprise for creating and deploying cross-platform, object-oriented applications.

For further information contact your local IBM representative.
URL: http://www.ibm.com/software/ad/smalltalk.