225

MVS update

June 2005

In this issue

A tcP PUBLICATION

MVS Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

Phone: 214-340-5690
Fax: 214-341-7081

Editor
Trevor Eddolls
E-mail: trevore@xephon.com

Publisher
Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues
A year's subscription to *MVS Update*, comprising twelve monthly issues, costs $505.00 in the USA and Canada; £340.00 in the UK; £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 2000 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.

Contributions
When Xephon is given copyright, articles published in *MVS Update* are paid for at the rate of $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. 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.

DFSORT performance enhancements

This article will introduce various features of IBM's DFSORT that can improve application performance and programmer productivity.

As you will be aware, DFSORT is a feature-rich product. Most people do basic sorting and some enhanced operations with it. However, there are many improvements that have been made to the product over the past few years that can seriously enhance and improve productivity for both programming users and also in a production workload.

Saving CPU and elapsed time when an application runs is one benefit; saving programmer time to write and maintain those applications represents significant cost reductions as well.

ICEGENER

There are still many sites that have not integrated ICEGENER as a replacement for IEBGENER. This is amazing in itself because ICEGENER has now been around for over 10 years. It is worth reviewing the manuals to see how to code the necessary system modifications to ensure that when IEBGENER runs, ICEGENER is called automatically.

ICEGENER uses DFSORT to process IEBGENER jobs when possible, and transfers control to IEBGENER when DFSORT can't be used. Most IEBGENER jobs that use DUMMY for SYSIN can be processed by DFSORT, resulting in significant performance improvements. As an added benefit, DFSORT issues messages containing useful information, such as the number of records copied and the RECFM, LRECL, and BLKSIZE of the SYSUT1 and SYSUT2 datasets.

Installing ICEGENER as a direct replacement for IEBGENER even lets RACF's IRRUT200 utility take advantage of ICEGENER's performance improvements.

OUTFIL

OUTFIL is used to create multiple output datasets of unedited records, ranges of records, reports, and sample records. It can do this from one pass of an input dataset. This obviously reduces many overheads. Run one job, reduce CPU and storage usage, and generally cut down on system resource usage. IBM estimates in some environments a reduction in excess of 80% in job elapsed times and up to a 30% reduction in CPU use. EXCPs can be reduced by over 90%.

ICETOOL

ICETOOL is one of IBM's hidden gems. It allows complex tasks to be tackled with ease, and is efficient in how it handles input and output. Chiefly, ICETOOL is a reporting utility. It has 13 commands that can allow it to perform a multitude of tasks. These are detailed below:

- COPY – copies a dataset to one or more output datasets. Multiple output is handled using a single pass over the input.
- COUNT – prints a message containing a count of the records in a dataset. It can also be used to set RC=12 or RC=0, based on the count of records in a dataset being empty, not empty, higher, lower, equal, or not equal.
- DEFAULTS – prints the DFSORT installation defaults in a separate list dataset.
- DISPLAY – prints the values and characters of a specified numeric, including SMF date/time, and character fields, in a separate list dataset.
- MODE – sets or resets scanning and error actions.
- OCCUR – prints each unique value for a specified numeric, including SMF date/time, and character fields, and the number of times it occurs, in a separate list dataset.

- RANGE – prints a message containing a count of values in a range (that is, higher, lower, equal, or not equal) for a numeric field.
- SELECT – selects records for an output dataset based on meeting criteria (that is, all duplicates, no duplicates, first, last, first duplicate, last duplicate, higher, lower, or equal) for the number of times numeric or character field values occur. Records that are not selected can be saved in a separate output dataset.
- SORT – sorts a dataset to one or more output datasets. Multiple output is handled using a single pass over the input.
- SPLICE – splices together fields from records that have the same numeric or character field values (that is, duplicate values), but different information. Fields from two or more records can be combined to create an output record. The fields to be spliced can originate from records in different datasets, so you can use SPLICE to do various 'join' and 'match' operations.
- STATS – prints messages containing the minimum, maximum, average, and total of values in numeric fields.
- UNIQUE – prints a message containing the count of unique values in a numeric or character field.
- VERIFY – prints a message identifying each invalid value found in decimal fields.

ELIMINATION OF PROGRAMMING

IBM has provided a number of commands/control statements that can be executed via DFSORT and that remove the need to write small programs to manipulate data. This not only saves on programmer time but also eliminates many jobs that would be required to compile and link edit programs and the maintenance of these programs. The most widely used DFSORT statements are shown below:

- INCLUDE – selects records to be kept, based on logical criteria.
- OMIT – selects records to be deleted, based on logical criteria.
- SUM – produces a single record for each unique sort or merge key with optional field totals.
- OUTREC/INREC – specifies reformatting, timestamps, translation, numeric editing, numeric conversion, substitution, arithmetic operations, hexadecimal display, and sequence numbers.
- OPTION – overrides installation defaults and supplies optional information such as the number of records to skip before sorting.
- OUTFIL – specifies multiple output datasets, subsets, sampling, repetition, reports, reformatting, timestamps, translation, numeric editing, numeric conversion, substitution, arithmetic operations, hexadecimal display, sequence numbers, VB to FB and FB to VB conversion, and more.

SMF DATE AND TIME FORMATS

DFSORT allows the use of a wide variety of character and numeric formats. DFSORT's SMF date formats (DT1, DT2, and DT3) and SMF time formats (TM1, TM2, TM3, and TM4) can be used with INREC, OUTREC, OUTFIL's OUTREC, and ICETOOL's DISPLAY and OCCUR operators to display the normally unreadable SMF date and time values in a wide range of recognizable ways.

An example of the DISPLAY control statement to format SMSF date and time is shown below:

```
DISPLAY FROM(SMF3Ø) LIST(SMFOREC) -
        TITLE('SMF Type-3Ø Records') DATE(4MD/) -
        HEADER('Date') ON(11,4,DT1,E'9999/99/99') -
        HEADER('Time') ON(7,4,TM1,E'99:99:99') -
        HEADER('Sys') ON(15,4,CH) -
        HEADER('Jobname') ON(19,8,CH) -
        HEADER('Datasetname') ON(69,44,CH)
```

The output pointed to by SMFOREC will contain date and time fields in the format C'YYYY/MM/DD' and HH:MM:SS respectively.

PROGRAM CALLS

DFSORT control statements can do the work of your programs as well. COBOL and PL/I programs can call DFSORT using SORT or the PLISRT*x* routines. You can pass DFSORT control statements to these programs by coding the //DFSPARM DD in your JCL.

USING SYMBOLS

A symbol is a name that you can use to refer to a field or constant. Sets of symbols, also called mappings, can be used to describe a group of related fields and constants such as the information in a particular type of record.

DFSORT's symbol processing feature gives you a powerful, simple, and flexible way to create symbol mappings for your own frequently-used data. IBM also provides a number of symbol mappings for DCOLLECT, the SMS Rmm product, and RACF. The RACF and DCOLLECT ones can be downloaded from IBM's Web site. The SMS symbols are contained in SYS1.MACLIB in members EDGACTSY, EDGEXTSY, and EDGSMFSY.

By using symbols, the product's syntax becomes like a high-level language. Symbols can help to standardize your DFSORT applications and increase your productivity. Once you create or obtain symbol mappings, you can use the symbols they define anywhere you can use a field or constant in any DFSORT control statement or ICETOOL operator. DFSORT symbols can be up to 50 characters, are case-sensitive, and can include underscore and hyphen characters. Thus, you can create meaningful descriptive names for your symbols, such as Bill_of_Material, which make sorts much easier to understand than just number references to record fields.

NATIONAL LANGUAGE SUPPORT

If you need to sort, merge, and create subsets of data according to a defined country specification and then create reports using the preferred date, time, and numeric notations of individual countries, DFSORT has features that can meet all of these requirements.

DFSORT allows the selection of an active locale at installation or run time with the LOCALE operand, and will sort, merge, and select subsets of records according to the collating rules defined in the active locale. Both ICETOOL and OUTFIL allow date, time, and numeric values in reports to be formatted in many of the notations used throughout the world.

DATE AND TIME CONTROLS

DFSORT allows you to use extra system resources based on the time of day. This feature is accomplished using the ICETD1, ICETD2, ICETD3, and ICETD4 modules. You can use any or all of ICETD1–4 to raise the DSA value from the default value of 32MB to a larger value of 64MB for DFSORT jobs that start at weekends. For example:

```
ICEMAC JCL,ENABLE=TD1,SVC=(,ALT)
ICEMAC INV,ENABLE=TD1,SVC=(,ALT)
ICEMAC TD1,WKEND=ALL,SVC=(,ALT),DSA=64
```

By setting your ICEAM1-4 and ICETD1-4 defaults appropriately, you can fine-tune DFSORT's resource usage for special situations at your site.

INTEGRATION WITH IDCAMS BLDINDEX

IBM now provides a method that allows unchanged BLDINDEX IDCAMS jobs to call DFSORT. BLDINDEX's use of this new DFSORT support also allows sorting of indexes in an all-DFSMS environment.

In IBM tests on an index of half a million records, IBM claims that it saw improvements of 75% elapsed time reduction, 70% CPU time reduction, and 82% EXCP time reduction at the maximum performance levels.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

A basic suite of table subroutines for ISPF DM

One of the major functions of the IBM ISPF Dialog Manager (DM) is the ability to create and manipulate tables. Nothing looks as impressive as a table, and this form of input and output is often the most appropriate. For example, a panel might contain 20 input fields, but it will look very busy and it cannot be expanded. A table can contain thousands of rows and will be displayed in a scrollable fashion.

Unfortunately, some of the processes, particularly displaying, are necessarily complex and very frustrating to establish satisfactorily. Consequently, inconsistent, half-hearted implementations abound. I have given here what I believe are a handful of small and well-defined REXX procedures that reveal how to create, load, delete, and, most importantly of all, display tables. These subroutines can be easily incorporated into any REXX program requiring the function. A wide range of additional options exist. You may, for example, want to open an existing table (TBOPEN), make a permanent table (open with WRITE and use TBCLOSE) or give a table to a JCL skeleton – file tailoring (TABSEL). These subroutines will give the basics and avoid the early frustrations. A future article will offer subroutines to find text in a table, send table output to a sequential file, and carry out file-tailoring with tables.

Notes:

1 The display subroutine is surprisingly sophisticated. It offers full-screen editing and will consecutively process line operators, primary commands, and scroll requests presented simultaneously whilst maintaining the table sort order and dealing with the complications of key editing.

2 The Trace, Control, and first TBEND statement (which tries to delete the table before attempting to create it) are all used for debugging purposes. They are useful in the long term, and may be worth keeping.

3 To execute this dialog, you need to put the EXEC, XEPH1, in a library concatenated to your logon procedure SYSEXEC or SYSPROC DD, and the panel in a library concatenated to your logon procedure ISPPLIB statement. More dynamic alternatives, such as LIBDEF and ALTLIB, are preferable but beyond the scope of this article.

4 The REXX code is written in upper case and in the somewhat terse fashion of a CLIST. This makes printed reproduction clearer and displayed output more compact, and avoids case confusion in variables.

5 I have permitted myself one small extravagance, which has a big effect. On entry to the EXEC, you may specify whether you wish to BROWSE or EDIT the table. I have done this to demonstrate how, with a little extra coding (and that largely on the panel) the functions of table management may be expanded.

This is the XEPH1 EXEC. Put it in a library concatenated to your log-on procedure's SYSEXEC or SYSPROC DD statement. Call it with 'TSO %XEPH1' or 'TSO %XEPH1 EDIT'.

```
/* rexx */
TRACE("O")                 /* make this "C" to turn command trace on   */
ADDRESS ISPEXEC            /* We will talk to the ISPF dialog manager  */
"CONTROL ERRORS RETURN"    /* I will process my own errors.            */

/* --- Main process ------------------------------------------------ */
```

```
ARG ACCMODE                /* get user-supplied input, if any          */
IF (ACCMODE <> "EDIT") THEN ACCMODE="BROWSE" /* not edit then browse  */
TABNAME="XEPH1TAB"         /* this will be my table name,              */
PANNAME="XEPH1PAN"         /* and this will be my panel name,          */
AC=MAKETAB(TABNAME)        /* Go create the table.                     */
IF (AC=Ø) THEN DO          /* was that OK? if it was then load it...   */
 CALL LOADTAB TABNAME      /* Load: no RC requested or checked         */
 CALL DISPTAB TABNAME,     /* Go to display subroutine with table,     */
     ,PANNAME,ACCMODE      /* panel and access mode (browse or edit)   */
 AC=DELETAB(TABNAME)       /* Delete the table.                        */
 END
EXIT AC                    /* bye...                                   */

/* --- Create table subroutine ------------------------------------- */

MAKETAB: PROCEDURE         /* inherit ISPEXEC, no variables exposed    */
 ARG TAB                   /* I get the table name. It will be a temp- */
                           /* orary table, one key and one name field. */
 "TBEND" TAB
 "TBCREATE" TAB "KEYS(K1) NAMES(N1) NOWRITE" /* issue the create      */
 AC=RC                     /* save the return code                     */
 IF (AC=Ø) THEN DO         /* Create OK?                               */
                           /* just for fun, we can make it a sorted    */
  "TBSORT" TAB "FIELDS(K1,C,A)" /* table - sorting on the key field.  */
  AC=RC                             /* save return code for caller     */
  IF (AC>Ø) THEN SAY "Sort? r/c=" AC  /* sort error. tell user        */
  END
 ELSE SAY "Create? r/c=" AC
 RETURN AC                 /* and send back return code. Ø=OK          */

/* --- Delete table subroutine ------------------------------------- */

DELETAB: PROCEDURE         /* inherit ISPEXEC, no variables exposed    */
 ARG TAB                   /* I get the table name.                    */
 "TBEND" TAB               /* issue the end (delete, no save).         */
 AC=RC                     /* save the return code                     */
 IF (AC>Ø) THEN SAY "Delete? r/c=" AC                                  */
 RETURN AC                 /* and send back return code. Ø=OK          */

/* --- Load table subroutine --------------------------------------- */

LOADTAB: PROCEDURE         /* inherit ISPEXEC, no variables exposed    */
 ARG TAB                   /* I get the table name.                    */
 "TBVCLEAR" TAB            /* Clear fields ('columns' if you like)     */
 DO X = 1 TO 9             /* Now we add nine rows of data. One unique */
  K1=X                     /* key in K1 and some data in N1. Note that */
  N1="Data for line" X     /* 'order' is essential if we are to keep   */
  "TBADD" TAB "ORDER"      /* the table sorted                         */
  IF (RC>Ø) THEN SAY "Load" K1||"? r/c=" RC
  END                     /* we don't save the 'add' return code 'cos */
 RETURN Ø                 /* we always need to continue.              */



/* --- Display table subroutine ------------------------------------ */

DISPTAB: PROCEDURE        /* inherit ISPEXEC, no variables exposed    */
 ARG TAB,PAN,ACC          /* TAB=TABNAME PAN=PANNAME ACC=BROWSE/EDIT  */
 O=""                     /* Clear lineop                             */
 "TBTOP" TAB              /* Set table to top                         */
 AC=Ø                     /*                                          */
 OLDPOS=Ø                 /* dope oldpos for first time               */
 "VPUT (ACC) SHARED"
 DO WHILE AC<8            /*                                          */
  "TBDISPL" TAB "PANEL("||PAN||") ROWID(OLDROW) POSITION(OLDPOS)",
   "CSRROW("||OLDPOS||") AUTOSEL(NO)"
  AC=RC                   /*                                          */
  IF AC>8 THEN RETURN 16  /* error in display? out.                   */
                          /* --- obey line operators first ---        */
  DO WHILE ZTDSELS>Ø      /* ztdsels = # of rows selected, sort of.   */
   "CONTROL DISPLAY SAVE" /* save the world                           */
   O=TRANSLATE(O)         /* upper-case the lineop                    */
   BC=Ø                   /* we're going in...                        */
   SELECT                 /*                                          */
                          /*                                          */
    WHEN (ACC="BROWSE") THEN NOP
                          /*                                          */
    WHEN (O=" ") THEN DO  /* line overtype?                           */
     "TBPUT" TAB "ORDER"  /* replace the line.                        */
     BC=RC                /* save rc.                                 */
     IF (BC>Ø) THEN DO    /* not zero? edited a key perhaps...        */
      "TBADD" TAB "ORDER" /* so do this - add the 'new' row,          */
      BC=RC               /* save rc,                                 */
      IF (BC=Ø) THEN DO   /* zero? we edited a key then, so           */
       "TBSKIP" TAB "ROW("||OLDROW||")" /* skip to 'old' row and      */
       "TBDELETE" TAB                   /* delete it.                 */
       END
      ELSE SAY "Update? r/c=" BC
      END
     O=""                 /* clear lineop for this row.               */
     END                  /*                                          */

    WHEN (O="D") THEN DO  /* deleting a row? Just in case key was     */
     "TBSKIP" TAB "ROW("||OLDROW||")"  /* overwritten we skip to here */
     "TBDELETE" TAB       /* and delete the row as requested.         */
     IF (RC>Ø) THEN SAY "Delete? r/c=" RC
     O=""                 /* clear lineop for this row                */
     END
    WHEN (O="C") THEN DO  /* Copying a row? Make sure you change the  */
     "TBADD" TAB "ORDER"  /* key first! Then add the row.             */
     IF (RC>Ø) THEN SAY "Copy row? r/c=" RC
     O=""                 /* clear lineop for this row                */
     END

    OTHERWISE NOP         /* You cannot do anything else.             */
    END

   "CONTROL DISPLAY RESTORE" /* rebuild the world                     */
   O=""                      /* and last clean of lineop...           */
   IF ((ZTDSELS>1)&(BC=Ø)) THEN  /* more than one row? carry on then  */
    "TBDISPL" TAB "ROWID(OLDROW) POSITION(OLDPOS)",
     "CSRROW("||OLDPOS||") AUTOSEL(NO)"
   ELSE ZTDSELS=Ø         /* else purge lineop selections             */
   END
                          /* --- obey primary commands next ---       */
  IF ZCMD<>"" THEN DO     /* Any primary command?                     */
   SELECT                 /* Yes? process it.                         */
    WHEN (ZCMD="A") THEN DO  /* Add a row?                            */
     "TBVCLEAR" TAB       /* Clear the fields.                        */
     K1=TIME("S")         /* Set key to sort-of-unique value          */
     N1="Added"           /* and name to something...                 */
     "TBADD" TAB "ORDER"  /* try and add the row                      */
     IF (RC>Ø) THEN SAY "Add? r/c=" RC
     END
    WHEN (ZCMD="X") THEN AC=8  /* we want to leave, so...             */
    OTHERWISE NOP         /* no other commands for us, send it back   */
    END
   END
                          /* --- obey scroll commands last ---        */
  "TBTOP" TAB             /* Go to top                                */
  "TBSKIP" TAB "NUMBER("ZTDTOP")"      /* Skip to where we last were  */
  "VGET (ZVERB ZSCROLLN)" /* Did you want to scroll?                  */
  IF ZVERB=UP THEN "TBSKIP" TAB "NUMBER(-"ZSCROLLN")"   /* Go up      */
  IF ZVERB=DOWN THEN "TBSKIP" TAB "NUMBER(+"ZSCROLLN")" /*  or down    */
  END                     /* and do it all again!                     */
 RETURN Ø                 /*                                          */
```

These are the (minimal) panels. Put them in a library concatenated to your log-on procedure’s ISPPLIB DD statement.

Main panel XEPH1PAN (mandatory – this panel must be provided):

```
)ATTR DEFAULT(%*_)
 _ TYPE(NEF) CAPS(ON)
 { TYPE(VOI) CAPS(OFF)   /* initially set to browse */
 } TYPE(NT)  SKIP(ON)

)BODY EXPAND(\\)
}Command =>_ZCMD                                 }Scroll =>_AMT }
}O Key      Name
}\-\
)MODEL ROWS(ALL)
_O{K1      {N1                              }
)INIT
 .HELP = XEPHPANH
 &ZTDMARK = ''
 IF (&ACC='EDIT') .ATTRCHAR({)='TYPE(NEF) CAPS(OFF)' /* edit */
)END
```

Help panel XEPHPANH (optional – needed if user hits PF1 for help):

```
)ATTR DEFAULT(%*_)
 _ TYPE(NEF) CAPS(ON)
 { TYPE(NEF) CAPS(OFF)
 } TYPE(NT)  SKIP(ON)

)BODY WINDOW(72,10) EXPAND(\\)
}Command =>_ZCMD
}
}To update a row, simply overtype it. If a key is edited, make sure the
}new key is unique and note the sort order may change. To delete a row,
}use the (D)elete line operator. To copy a row, overtype the key - to
}something unique - and use the (C)opy command. To add a new row, use
}the (A)dd primary command. To leave the edit session, use your exit key
}(usually PF3) or the e(X)it primary comand. Nothing is saved.
}
\ \Hit PF3 to return
)END
```

Deryck Swatman
System Programmer
HM Land Registry (UK)

Generation datasets – some basic information

One of the most commonly-used types of dataset, the generation data group (or GDG as it is commonly referred to), is not always what it might seem to be. I was amazed, after discussing these structures, that many of my colleagues did not really understand what generation datasets are or how they are actually catalogued.

In this article I will try to explain some of the basics of generation datasets, and I hope that this brief insight will help those who do not fully understand them.

One of the most common misconceptions I have encountered is how the GDGs are catalogued. In ISPF, if you enter a base generation dataset name under Option 3.4 you will get a list containing all the active generation dataset entries. The display will be similar to the one presented below:

```
DSLIST - Data Sets Matching JXB7884.DSET                Row 1 of 6
Command ===>                                              Scroll ===> CSR

Command - Enter "/" to select action       Volume

JXB7884.DSET                               ??????
JXB7884.DSET.GØØ93VØØ                      PRDØØ1
JXB7884.DSET.GØØ94VØØ                      PRDØ1Ø
JXB7884.DSET.GØØ95VØØ                      PRDØ3Ø
JXB7884.DSET.GØØ96VØØ                      PRDØ79
JXB7884.DSET.GØØ97VØØ                      PRDØ8Ø
```

The generation dataset in the example has five active generations. It would appear that each entry is a unique catalog entry. However, this is not the case; all the information that is stored about active generations is actually part of one catalog record, a type ‘B’ record. This record will contain the generation dataset attributes such as the LIMIT and whether or not SCRATCH or NOSCRATCH was specified when the GDG was defined. It is common for a generation data group actually to exceed the type ‘B’ catalog record’s capacity if there are a lot of active generations in retention across many volumes. The system handles this by acquiring additional space in a type ‘J’ record. These are known as catalog extension records.

The generation dataset base catalog record, called the Generation Aging Table (or GAT for short), is used to establish the relativity of the different generations.

The actual placement of an entry in the Generation Aging Table is determined using the ‘nnnn’ component of the G*nnnn*V*vv* qualifier of the dataset during creation or recovery. It is the physical GAT entries that determine the desired generation dataset when referencing a generation by number enclosed in brackets after the base name – for example JXB7884.DSET(-1).

The whole catalog mechanism needs to be understood because it can cause some challenges if the base GDG entry has issues. If the base entry has problems, then all access to the active generation datasets will be lost. In a copy or move situation, the base catalog record is only the storage for the sequence of generations and the generation data group’s attributes. Therefore, if you recreate the base entry to recover from corruption or previous loss, or it is not manipulated as part of a data move (say movement of data from one catalog to another), you may have trouble recreating or accessing generations.

All generations are catalogued initially in what is termed a ‘deferred roll-in’ status. They only become active when de-allocated for the first time. Generations that are specified with a disposition of KEEP, even though they roll off the list of active generations, are termed ‘rolled off’. In both cases, deferred roll-in and rolled off generation datasets have unique catalog entries that are non-VSAM type catalog entries. This needs to be remembered when using ISPF 3.4, for example, because, even though the display may show 100 entries, only the deferred roll-in and the rolled off generation datasets actually have unique catalog entries. The rest are part of the type ‘B’ catalog record for that generation data group.

IBM, near the end of the last century, introduced the Automatic Wrap Mechanism for generation datasets. This allows generation data groups to automatically wrap when they reach generation 9999. It is useful to understand how this works because it can be required if you have issues with the generation dataset.

When wrapping from 9999, the programs processing the operation must have a way of identifying that the new generation 0001 comes after the old generation 9999 and not before it – as one would expect! This might seem obvious if you know that a generation dataset limit is 255. There is, however, no rule that says generations have to increase by a unit of 1, and therefore the wrap logic needs to take this into consideration. IBM set up a wrap flag for the lower numbered generations that are created as a result of a wrap. This flag, when set, shows that these generations come after 9999, not before. It is necessary to reset these flags when the last higher numbered generation has rolled over, or if the generation data group is emptied. The logic takes this into account. You do need to be aware that the wrap bit will be turned off when generation 10,000 is created. This means the generation data group can extend beyond 10,000 without having to be redefined.

Things can go astray, however, when absolute generation numbers are used, generations are skipped, or something outside the normal sequencing occurs. An example of this could be an HSM Recall of an old migrated generation dataset.

Also, IBM had to consider the ordering of generations when absolute generation numbers are used to create generation datasets. This is more common if you have to restore a generation dataset. The absolute generation number is when G*nnnn*V*vv* is coded. For example, a relative generation is JXB7884.DSET(-1). The absolute generation would be JXB7884.DSET.G0098V00.

It would be unusual to see large gaps in generation numbers, but to ensure flexibility IBM did need to consider this. For example, if you were going to restore generation 100 and generation 3100, which would be relative generation 0 and which –1?

To resolve this potential problem situation, IBM created a rule for what is termed ‘predictable wrapping’, stating that the numerical range of a generation data group’s generations must not exceed 1000. For most generation data groups this rule is reasonable because most increment by a unit of 1 and the active generation limit is 255. This limit of 1000 can be a concern, however, where a scheme exists that does not increment by a unit of 1.

When backing up and restoring generation datasets, the easiest way to avoid issues is to back up the base catalog record and all active generations in one job step. A number of utilities can be set up to do this for you. Restoring generations out of sequence can cause problems. Restoring more generations than the limit can also cause very undesirable results. The generation dataset processing will always attempt to use the G*nnnn*V*vv* qualifier to set the relativity for the dataset in the Generation Aging Table. For example, if you restore G0099V00 first and G0098V00 second, the G0099V00 will end up being the (0) generation and G0098V00 will be (-1). This, however, may not be true when active wrapped generations are restored. If you start with an empty generation data group and restore G0001V00 first and then follow it with G9999V00 – it is the G9999V00 dataset that ends up being the (0) generation and the G0001V00 that becomes (-1). This is most likely the reverse of what is the true generation set-up.

Another issue with restoring more generations than the generation data group's limit is that if the generation data group is full (active generations are equal to the LIMIT) and a lower numbered generation dataset is restored, then the lower numbered generation dataset ends up replacing the oldest generation in the Generation Aging Table. You should try to back up and restore the base record and then the generation datasets in order. This ensures that if a wrap is included in the process it will be restored correctly and all base record attributes will be current.

Defining generation datasets is simple and can be achieved easily using IDCAMS DEFINE. The structure of this command is illustrated below:

```
DEFINE   GENERATIONDATAGROUP
         (NAME(entryname)
         LIMIT(limit)
         [EMPTY|NOEMPTY]
         [OWNER(ownerid)]
         [SCRATCH|NOSCRATCH]
         [TO(date)|FOR(days)])
       [CATALOG(catname[/password])]
```

The special fields that can be seen when running a LISTCAT against a generation dataset describe the attributes of the generation data group:

- EMPTY – all generation datasets in the generation data group are uncatalogued when the maximum number (given under LIMIT) is reached and one more dataset is to added to the group.
- LIMIT – the maximum number of generation datasets allowed in the generation data group.
- NOEMPTY – only the oldest generation dataset in the generation data group is uncatalogued when the maximum number (given under LIMIT) is reached and one more dataset is to be added to the group.
- NOSCRATCH – generation datasets are not to be scratched (see SCRATCH below) when uncatalogued.
- SCRATCH – generation datasets are to be scratched (that is, the DSCB describing each one is removed from the VTOC of the volume where it resides) when uncatalogued.

Another misunderstood aspect of generation datasets is that they must be non-VSAM. VSAM datasets cannot be generation datasets. IBM also strongly recommends that, when creating generation datasets, relative generation number is used and not absolute generation number. When creating a generation dataset, the relative generation number tells the system whether this is the first dataset being added during the job, or the second, or the third, etc. When retrieving a generation dataset, the relative generation number tells the system how many datasets have been added to the group since this dataset was added. The first time you use a relative generation number for a generation data group within a job, the system establishes the relationship between the relative generation number and the absolute generation number. The system maintains this relationship throughout the job.

For example, if you create a generation dataset with a relative generation number of (+1), the system recognizes any subsequent reference to (+1) throughout the job as having the same absolute generation number. You need to remember this when writing JCL.

If the generation data group is to be SMS-managed, it must consist of sequential or direct access type datasets that reside on DASD volumes. The actual dataset organizations of the datasets can have unlike or like attributes. The actual generation data group can include both SMS and non-SMS-managed datasets. Non-SMS-managed generation datasets can have the same features as SMS-managed ones and can also reside on tape.

You will find that the most common errors that occur with generation datasets are duplicate dataset name conditions and out of sequence errors. Duplicate dataset name checking occurs early in generation data group processing against G0000V00 and does not take wrap bits into consideration. When a mixture of generations exist with wrap bits on and off, if you specify a +1 generation, the new generation will be retained but not rolled in. The following messages are normally issued in such a circumstance:

```
IGDØ7ØØ1I GDG ROLL IN ERROR - RETURN CODE 14Ø   (msgIGDØ7ØØ1I
                    REASON CODE 122 MODULE IGGØCLEL         rc14Ø rsn122)
IGD1Ø4I     GREG.GDS.TEST.G1ØØØVØØ RETAINED,  DDNAME=GDG1DD1
```

RETURN CODE 140 means that inconsistent or conflicting arguments were provided.

REASON CODE 122 means that catalog G1000V*xx* will cause the GDG to exceed the limit of 10,999.

Programmer response is to clean up the GDG in error then catalog G1000V*xx*.

If this occurs, you should first try to process the +1 generation again.

IBM informational APAR II07276 describes many different potential problem scenarios and is well worth reading to understand some of the pitfalls of generation datasets.

Three other reference sources that I would recommend are:

1 APAR II06463.

2 APAR II08285.

3 *JCL Users Guide Appendix B*.

Elizabeth Bradley
Systems Programmer
Meerkat Computer Services (UK)

Load analyser utility

"Only load modules contain accurate information about the code running in a system."

A load module begins with a source module written by a programmer in a high-level programming language. The source module must then be submitted to a compiler for the language the source is written in (there is a unique compiler for each programming language). The compiler changes the programming language into machine language – the machine instructions a processor can execute – and puts them in an object module. The object module is then fed into the linkage editor, which transforms it into a load module.

The load analyser utility processes load modules, examines their object code, and extracts information – such as load module names, language used, compilers and release levels, compiler signature – which is critical for the preparatory phase of decompilation of load modules to recover lost source code. (For a full description, see ‘No Source, No Worry – Generating Source Code from the Load Module’, *MVS Update*, May 2005, issue 224.)

JCL TO RUN REXX ‘MAINPROG’

```
//LOADXOM  JOB (SSG,MVS,SSGLAX,D2,ST99X),'RUN-REX',
//            MSGLEVEL=(1,1),MSGCLASS=X,CLASS=O,NOTIFY=&SYSUID
//STEP1    EXEC PGM=IKJEFTØ1
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//* CONCAT FILES
//ISPMLIB  DD DISP=SHR,DSN=ISP.SISPMENU
//ISPPLIB  DD DISP=SHR,DSN=ISP.SISPPENU
//ISPTLIB  DD DISP=SHR,DSN=ISP.SISPTENU
//ISPSLIB  DD DISP=SHR,DSN=ISP.SISPSENU
//ISPPROF  DD DISP=SHR,DSN=SSGLAX.ISPF.ISPPROF.BKP
//SYSPROC  DD DISP=SHR,DSN=SSGLAX.REXX.LOAD.FINDER.FINAL
//SYSTSIN  DD *
  ISPSTART CMD(%MAINPROG)
/*
```

EDIT MACRO $MACRO

```
/****************************** REXX *********************************/

"ISREDIT MACRO"
ADDRESS ISPEXEC VGET ('COMP') PROFILE
ADDRESS ISREDIT "FIND '&COMP' FIRST"
RCVAL = RC
ADDRESS ISPEXEC VPUT ('RCVAL') PROFILE
ADDRESS ISREDIT "CANCEL"
EXIT
```

DATASET: PRODNAME (DSORG: PO, RECFM: F)

```
    36ØSASØ37       A             S/36Ø OS ASSEMBLER (F)
    5734AS1ØØ       A             OS ASSEMBLER H
    5741SC1Ø3       A             OS/VS ASSEMBLER (XF)
    5668962Ø1       A             ASSEMBLER H V2
    5696234ØØ       A             HIGH-LEVEL ASSEMBLER
    4ØCB1           CO            OS/VS COBOL R2M2      (VSR2)
    574ØCB1Ø3       CO            OS/VS COBOL R2M3 R2M4 (VSR1)
    5668958Ø1       CO            VS COBOL II
```

566895807	CO	COBOL/370 OR COBOL FOR MVS & VM
5648A2500	CO	COBOL FOR OS/390 AND VM V2
5655G5300	CO	ENTERPRISE COBOL FOR Z/OS V3
360SED521	LE	S/360 OS LINKAGE EDITOR (F)
5752SC104	LE	OS/VS LINKAGE EDITOR
566528408	LE	DFP/XA LINKAGE EDITOR
566529508	LE	DFP/370 LINKAGE EDITOR
5695DF108	LE	DFSMS/MVS BINDER
5695PMB01	LE	Z/OS BINDER
5688040	C++	C/370 COMPILER V1
5688187	C++	C/370 COMPILER V2
5688216	C++	SAA AD/CYCLE C/370
5645001	C++	C/C++ OS/390 R2
5647A01	C++	C/C++ OS/390 R4
5694A01	C++	C/C++ Z/OS R5
5668-806	F	VS FORTRAN V2 (COMP/LIB/DEBUG)
5796-PKR	F	EXT. EXPONENT RANGE FOR FORTRAN
5734-PL1	PL	OS PL/I OPTIMIZING COMPILER V1
5668-910	PL	OS PL/I V2 (COMP/LIB)
566876701	MIB	VS PASCAL
569501301	MIB	REXX/370
5744AN101	MIB	ACF/NCP
566893801	MIB	ACF/NCP
516896201	MIB	ASSEMBLER H V2 (BIZARRE)
ASMG21FEB	MNIB	ASSEMBLER G (U OF WATERLOO)
CA-PLNK 0	MNIB	CA PL/S CLONE ?
GREXX-00	MNIB	GOAL SYSTEMS REXX
LATTICE_C	MNIB	LATTICE C
RXTCOMP	MNIB	REXXTOOLS
SAS/C	MNIB	SAS/C
SAS/C/	MNIB	SAS/C
SDS080888	MNIB	SAS/C
BLD121988	MNIB	SAS/C
TMCOMPIL	MNIB	XPEDITER
52ASM31686	MNIB	RUSSIAN ASSEMBLER

PROGRAM

```
/******************************REXX ***********************************/
/*                           LOAD ANALYSER UTILITY                    */
/*                                                                    */
/* This Utility scans a Load Module PDS and extracts information –    */
/* such as Load module name, Compiler and release level, Link editor, */
/* Assembler used and its corresponding signatures.                   */
/*                                                                    */
/*     INDSNAME: DATASET   (DSORG: PO, RECFM: FB)                     */
/*                Give input DATASET which contains the Load modules  */
/*     OUTPDS:   DATASET   (DSORG: PS, RECFM: FB, LRECL=100)          */
/*                It is the output dataset where details about each   */
/*               load module is stored.                             */
/*      PRODNAME: DATASET   (DSORG: PO, RECFM: FB)                  */
/*     It has the Product Signature, Product ID, and Product name.  */
/*      TEMPFILE:  DATASET   (DSORG: PS, RECFM: FB)                 */
/*              It's a temporary file used by the program.          */
/********************************************************************/
 "PROFILE NOPREFIX"
 INDSNAME = 'SSGLAX.DISASM.LOAD'
 OUTPDS        = 'SSGLAX.REXX.LOAD.TEXT'
 PRODNAME = 'SSGLAX.REXX.LOAD.FINDER.FINAL (PRODNAME)'
 TEMPFILE    = 'SSGLAX.REXX.LOAD.TEMP'
 COUNT = Ø
 CALL READ_PRODUCT_SIGNATURE
 CALL HEADER
 CALL READ_LOAD_MODULE
 CALL WRITE_TO_FILE
 EXIT
/********************************************************************/
/* This function reads the product signature, product id and product  */
/* name from the PRODNAME file.                                     */
/********************************************************************/
 READ_PRODUCT_SIGNATURE:
    "ALLOC DA("PRODNAME") F(INFILE) SHR REUSE"
    "EXECIO * DISKR INFILE (FINIS STEM IN1."
    PROD_COUNT = IN1.Ø
    DO I = 1 TO PROD_COUNT
        X = LEFT(IN1.I,72)
        PARSE VALUE X WITH SIGNATURE.I  PROD_CODE.I  PROD_NAME.I
        PROD_NAME.I = STRIP(PROD_NAME.I,'B')
    END
 RETURN
/********************************************************************/
/* This function scans the Load module PDS, extracts each member and  */
/* passes it to the LOADXMIT REXX Program.                          */
/********************************************************************/
 READ_LOAD_MODULE:
   ADDRESS ISPEXEC
   'LMINIT DATAID(INPDSN) DATASET('"'"INDSNAME"'"') ENQ(SHR) ORG(DSO)'
   IF RC = 8 THEN
   DO
      SAY 'INVALID DATASET NAME ' || INDSNAME
      EXIT
   END
   ELSE IF RC <> Ø THEN
   DO
      SAY 'INCORRECT LMINIT -FAILED WITH RC ' || RC
      EXIT
   END
   IF DSO <> 'PO' THEN
   DO
```

```
      SAY ' DATASET ' || INDSNAME || ' NOT A PDS '
      EXIT
   END
   ADDRESS ISPEXEC
   'LMOPEN DATAID(&INPDSN) OPTION(INPUT)'
   IF RC <> Ø THEN
   DO
      SAY 'INCORRECT LMOPEN -FAILED WITH RC ' || RC
      EXIT
   END
   DO UNTIL LMRC <> Ø
     ADDRESS ISPEXEC
        'LMMLIST DATAID(&INPDSN) OPTION(LIST) MEMBER(MEMVAR) STATS(YES)'
        IF RC > 8 THEN
        DO
          SAY 'INCORRECT LMMLIST - FAILED WITH RC ' || RC
          EXIT
        END
     LMRC=RC
     IF RC=Ø THEN
     DO
      PROGRAM = STRIP(MEMVAR,'B')
      INPDS = INDSNAME||'('||PROGRAM||')'
                                 /* STRIP TO REMOVE TRAILING BLANKS */
      ADDRESS TSO "%LOADXMIT  "INPDS" "TEMPFILE" "
      CALL SIGNATURE_FINDER TEMPFILE
     END
   END /* DO UNTIL */
 RETURN
/*********************************************************************/
/* This function scans each module for the Compiler, Link editor,     */
/* Assembler signature and populates the details in the string array  */
/*********************************************************************/
 SIGNATURE_FINDER:
 ARG TEMPFILE
 LOOP_COUNT=Ø
   DO I = 1 TO PROD_COUNT
     COMP = SIGNATURE.I
     ADDRESS ISPEXEC VPUT ('COMP') PROFILE
     ADDRESS ISPEXEC
     "VIEW DATASET('"TEMPFILE"') MACRO("$MACRO")"
     ADDRESS ISPEXEC VGET ('RCVAL') PROFILE
     IF RCVAL = Ø  THEN
     DO
       /* SAY '**********' COMP '-' SIGNATURE.I '-' PROD_NAME.I */
       LOOP_COUNT = LOOP_COUNT + 1
       COUNT = COUNT + 1
       SELECT
         WHEN PROD_CODE.I = 'A'    THEN
         DO
           TMP1 = 'ASSEMBLER     '
           IF SIGNATURE.I= '36ØSASØ37' THEN TMP2 = 'IEU     '
           IF SIGNATURE.I= '5734AS1ØØ' THEN TMP2 = 'IEV     '
           IF SIGNATURE.I= '5741SC1Ø3' THEN TMP2 = 'IFO,IFN'
           IF SIGNATURE.I= '5668962Ø1' THEN TMP2 = 'IEV     '
           IF SIGNATURE.I= '5696234ØØ' THEN TMP2 = 'ASM     '
         END
         WHEN PROD_CODE.I = 'LE'    THEN
        DO
           TMP1 = 'LINKAGE EDITOR'
           IF SIGNATURE.I= '36ØSED521' | ,
              SIGNATURE.I= '5695DF1Ø8' | ,
              SIGNATURE.I= '5695PMBØ1'  THEN TMP2 = 'IEW     '
           IF SIGNATURE.I= '5752SC1Ø4' | ,
             SIGNATURE.I= '5665284Ø8' | ,
             SIGNATURE.I= '5665295Ø8'  THEN TMP2 = 'HEW     '
        END
        WHEN PROD_CODE.I = 'CO'   THEN
        DO
          TMP1 = 'COMPILER      '
          TMP2 = 'COBOL  '
        END
        WHEN PROD_CODE.I = 'C++'  THEN
        DO
          TMP1 = 'COMPILER      '
          TMP2 = 'C & C++'
        END
        WHEN PROD_CODE.I = 'F'    THEN
        DO
          TMP1 = 'COMPILER      '
         TMP2 = 'FORTRAN'
       END
       WHEN PROD_CODE.I = 'PL'   THEN
       DO
         TMP1 = 'COMPILER      '
         TMP2 = 'PL/I   '
       END
       WHEN PROD_CODE.I = 'MIB'  THEN TMP1 = '              '
       WHEN PROD_CODE.I = 'MNIB' THEN TMP1 = '              '
       OTHERWISE SAY '***********NEW INSTRUCTION TYPE***********'
       END /* SELECT */
       STRING.COUNT = PAD(PROGRAM)||'  '||TMP1||'  '||TMP2||'  '   ,
                    ||SIGNATURE.I||'  '||PROD_NAME.I
       /* LOOP_COUNT =3 FOR COMPILER,LINKAGE EDITOR,ASSEMBLER */
       IF LOOP_COUNT = 3 THEN LEAVE
     END /* IF */
   END /* I */
   COUNT = COUNT + 1
   STRING.COUNT =  ,
    '----------------------------------------------------------------'
```

```
RETURN
/*******************************************************************/
/* This function writes the STRING array into the output file.     */
/*******************************************************************/
  WRITE_TO_FILE:
   ADDRESS TSO
      /* WRITE INTO FILE */
      "ALLOC DA("OUTPDS") F(OUTFILE) SHR REUSE"
      "EXECIO * DISKW OUTFILE (FINIS STEM STRING."
      "FREE DD (OUTFILE)"
   RETURN
/*******************************************************************/
/* This function generates the header.                             */
/*******************************************************************/
HEADER:
  COUNT = COUNT + 1
  STRING.COUNT = ,
  ' LOAD     PRODUCT      LANGUAGE    PRODUCT         PRODUCT'
  COUNT = COUNT + 1
  STRING.COUNT =  ,
  ' MODULE     TYPE                   SIGNATURE         NAME'
  COUNT = COUNT + 1
  STRING.COUNT = ,
   '_________________________________________________________________'
  COUNT = COUNT + 1
  STRING.COUNT =  '       '
RETURN
/*******************************************************************/
/* This function is used for padding spaces to make the input string */
/* of length eight.                                                */
/*******************************************************************/
  PAD:
  ARG PROGNAME
      LEN1 = LENGTH(PROGNAME)
      LEN2 = 8 - LEN1
      DO K = 1 TO LEN2
         PROGNAME=PROGNAME||' '
      END
  RETURN PROGNAME
/*******************************REXX********************************/
/* This is the REXX sub program called from program PROGMAIN. This  */
/* generates the xmited output of each Load module member passed as */
/* input parameter in INPDS.                                       */
/*******************************************************************/
PARSE ARG INPDS OUTPDS
"PROFILE NOPREFIX"
ADDRESS TSO "XMIT LOCAL.SSGLAX DA("INPDS") OUTDS("OUTPDS") "
EXI
```

SAMPLE OUTPUT

```
 LOAD       PRODUCT      LANGUAGE     PRODUCT                 PRODUCT
 MODULE      TYPE                    SIGNATURE                  NAME
--------------------------------------------------------------------------

ADDRR      ASSEMBLER        ASM      569623400        HIGH-LEVEL ASSEMBLER
--------------------------------------------------------------------------
ADDSUB     ASSEMBLER        ASM      569623400        HIGH-LEVEL ASSEMBLER
ADDSUB     COMPILER         COBOL    5648A2500   COBOL FOR OS/390 AND VM V2
ADDSUB     LINKAGE EDITOR   IEW      5695PMB01        Z/OS BINDER
--------------------------------------------------------------------------
DISPLAY2   ASSEMBLER        ASM      569623400        HIGH-LEVEL ASSEMBLER
DISPLAY2   COMPILER         COBOL    5648A2500   COBOL FOR OS/390 AND VM V2
DISPLAY2   LINKAGE EDITOR   IEW      5695PMB01        Z/OS BINDER
--------------------------------------------------------------------------
FACTRIAL   ASSEMBLER        ASM      569623400        HIGH-LEVEL ASSEMBLER
FACTRIAL   LINKAGE EDITOR   IEW      5695DF108        DFSMS/MVS BINDER
```

T S Laxminarayan
System Programmer
Tata Consultancy Services (India)

Syslog analysis using LOGLYZER – part 2

This month we conclude the code to analyse syslog information.

```
/*********** @REFRESH BEGIN TSOTRAP  2002/12/15 05:18:45 *************/
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
/*********************************************************************/
 tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume 0  */
/*********************************************************************/
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
/*********************************************************************/
/* If RC = Ø then return                                             */
/*********************************************************************/
          if CRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return CRC
             end
          else
             do
              trapmsg = center(' TSO Command Error Trap ',78,'-')
              terrmsg = errmsg(sigl 'TSO Command:')
/*********************************************************************/
/* If RC <> Ø then format output depending on environment            */
/*********************************************************************/
              if tsoenv = 'BACK' | execenv = 'OMVS' then
                 do
                  say trapmsg
                  do c=1 to tsoout.Ø
                     say tsoout.c
                  end
                  say trapmsg
                  call rcexit CRC terrmsg tso_cmd
                 end
              else
/*********************************************************************/
/* If this is foreground and ISPF is available, use the ISPF LOG     */
/*********************************************************************/
                 do
                  if ispfenv = 'YES' then
                     do
                      zedlmsg = trapmsg
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
                      do c=1 to tsoout.Ø
```

```
                         zedlmsg = tsoout.c
                         address ISPEXEC "LOG MSG(ISRZ000)"
                      end
                      zedlmsg = trapmsg
                      address ISPEXEC "LOG MSG(ISRZ000)"
                      call rcexit CRC terrmsg tso_cmd,
                          ' see the ISPF Log (Option 7.5) for details'
                     end
                  else
                     do
                      say trapmsg
                      do c=1 to tsoout.0
                         say tsoout.c
                      end
                      say trapmsg
                      call rcexit CRC terrmsg tso_cmd
                     end
                 end
             end
/*********** @REFRESH END   TSOTRAP  2002/12/15 05:18:45 *************/
/*********** @REFRESH BEGIN TSOQUIET 2003/05/26 10:22:58 *************/
/* TSOQUIET - Trap all output from a TSO command and ignore failures */
/*-------------------------------------------------------------------*/
/* TSOCMD   - TSO command to execute                                 */
/*********************************************************************/
 tsoquiet: module = 'TSOQUIET'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           call modtrace 'START' sigl
/*********************************************************************/
/* Accept command to execute and throw away results                  */
/*********************************************************************/
           arg tsocmd
           call outtrap 'garbage.' 0
           address TSO tsocmd
           call outtrap 'off'
/*********************************************************************/
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/*********** @REFRESH END   TSOQUIET 2003/05/26 10:22:58 *************/
/*********** @REFRESH BEGIN SAYDD    2004/03/29 23:48:37 *************/
/* SAYDD    - Print messages to the requested DD                     */
/*-------------------------------------------------------------------*/
/* MSGDD    - DDNAME to write messages to                            */
/* MSGLINES - number of blank lines to put before and after          */
/* MESSAGE  - Text to write to the MSGDD                             */
```

```
/*********************************************************************/
 saydd: module = 'SAYDD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        parse arg msgdd msglines message
        if words(msgdd msglines message) < 3 then
           call rcexit 33 'Missing MSGDD or MSGLINES'
        if datatype(msglines) <> 'NUM' then
           call rcexit 34 'MSGLINES must be numeric'
/*********************************************************************/
/* If this is not background then bypass                            */
/*********************************************************************/
        if tsoenv <> 'BACK' then
           do
            pull tracelvl . module . sigl . sparms
            call modtrace 'STOP' sigl
            interpret 'trace' tracelvl
            return
           end
/*********************************************************************/
/* Confirm the MSGDD exists                                          */
/*********************************************************************/
        call ddcheck msgdd
/*********************************************************************/
/* If a number is provided, add that number of blank lines before    */
/* the message                                                       */
/*********************************************************************/
        msgb = 1
        if msglines > Ø then
           do msgb=1 to msglines
              msgline.msgb = ' '
           end
/*********************************************************************/
/* If the linesize is too long break it into multiple lines and      */
/* create continuation records                                       */
/*********************************************************************/
        msgm = msgb
        if length(message) > 6Ø & substr(message,1,2) <> '@@' then
           do
            messst = lastpos(' ',message,6Ø)
            messseg = substr(message,1,messst)
            msgline.msgm = date() time() strip(messseg)
            message = strip(delstr(message,1,messst))
            do while length(message) > Ø
               msgm = msgm + 1
               if length(message) > 55 then
                  messst = lastpos(' ',message,55)
               if messst > Ø then
                  messseg = substr(message,1,messst)
               else
                  messseg = substr(message,1,length(message))
               msgline.msgm = date() time() 'CONT:' strip(messseg)
               message = strip(delstr(message,1,length(messseg)))
            end
           end
        else
/*******************************************************************/
/* Build print lines. Default strips and prefixes date and timestamp */
/* @BLANK - Blank line, no date and timestamp                      */
/* @      - No stripping, retains leading blanks                   */
/* @@     - No stripping, No date and timestamp                    */
/*******************************************************************/
           do
            select
               when message = '@BLANK@' then msgline.msgm = ' '
               when word(message,1) = '@' then
                    do
                     message = substr(message,2,length(message)-1)
                     msgline.msgm = date() time() message
                    end
               when substr(message,1,2) = '@@' then
                    do
                     message = substr(message,3,length(message)-2)
                     msgline.msgm = message
                    end
               otherwise msgline.msgm = date() time() strip(message)
            end
           end
/*******************************************************************/
/* If a number is provided, add that number of blank lines after    */
/* the message                                                      */
/*******************************************************************/
        if msglines > Ø then
           do msgt=1 to msglines
              msge = msgt + msgm
              msgline.msge = ' '
           end
/*******************************************************************/
/* Write the contents of the MSGLINE stem to the MSGDD              */
/*******************************************************************/
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
```

```
/*********** @REFRESH END   SAYDD    2004/03/29 23:48:37 *************/
/*********** @REFRESH BEGIN JOBINFO  2004/11/23 22:11:25 *************/
/* JOBINFO  - Get job related data from control blocks               */
/*-------------------------------------------------------------------*/
/* ITEM     - Optional item number desired, default is all           */
/*********************************************************************/
 jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/*********************************************************************/
/* Chase control blocks                                              */
/*********************************************************************/
          tcb      = ptr(540)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+180)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',0)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          progname = stg(jscb+360,8)
          jobdata  = jobtype jobnum stepname procstep progname asid
/*********************************************************************/
/* Return job data                                                   */
/*********************************************************************/
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return jobdata
             end
/*********** @REFRESH END   JOBINFO  2004/11/23 22:11:25 *************/
/*********** @REFRESH BEGIN COMMAFY  2004/11/04 18:53:50 *************/
/* COMMAFY  - Add commas to a number                                 */
/*-------------------------------------------------------------------*/
/* NUM      - The number to add commas to                            */
```

```
/*******************************************************************/
 commafy: module = 'COMMAFY'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
/*******************************************************************/
/* Insert commas by reversing the number and inserting every three   */
/*******************************************************************/
          arg commanum
          n=commanum'.9'
          do j=verify(n,123456789Ø,,verify(n,123456789Ø.,'M'))-4 to,
             verify(n,987654321,"M") by -3
             commanum=insert(',',commanum,j)
          end
/*******************************************************************/
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return commanum
/*********** @REFRESH END   COMMAFY  2ØØ4/11/Ø4 18:53:5Ø *************/
/*********** @REFRESH BEGIN STEM     2ØØ2/11/17 17:5Ø:41 *************/
/* STEM     - Set a value into a stem and increment the number       */
/*-----------------------------------------------------------------*/
/* STEMVAR  - The stem variable name (prefix with $ to accept blanks)*/
/* STEMNUM  - The stem variable number                               */
/* STEMVAL  - The stem variable value                                */
/*******************************************************************/
 stem: module = 'STEM'
       if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
/*******************************************************************/
/* Accept and validate the incoming values                           */
/*******************************************************************/
       parse arg stemvar stemnum stemval
       if stemvar = '' then call rcexit 91 'STEMVAR is missing'
       if stemnum = '' then call rcexit 92 'STEMNUM is missing'
       if datatype(stemnum) <> 'NUM' then
          call rcexit 94 'STEMNUM is not numeric "'stemnum'"'
/*******************************************************************/
/* If stemvar is prefixed with '$' then strip it and honor nulls     */
/*******************************************************************/
       if substr(stemvar,1,1) = '$' then
          do
           stemvar = strip(stemvar,'L','$')
           if stemval = '' then stemval = ' '
          end
       else
          if stemval = '' then call rcexit 93 'STEMVAL is missing'
/*********************************************************************/
/* Double up any imbedded quotes                                     */
/*********************************************************************/
       start = pos('"',stemval)
       if start <> Ø then
          do forever
             stemval = insert('"',stemval,start)
             start = pos('"',stemval,start+2)
             if start = Ø then leave
          end
/*********************************************************************/
/* Set incoming value into the stem entry                            */
/*********************************************************************/
       interpret stemvar'.'stemnum '= "'stemval'"'
/*********************************************************************/
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return stemnum + 1
/*********** @REFRESH END   STEM     2ØØ2/11/17 17:5Ø:41 *************/
/*********** @REFRESH BEGIN STEMSORT 2ØØ4/11/18 11:13:58 *************/
/* STEMSORT - Sort a stem variable (caller must strip nulls)         */
/*-------------------------------------------------------------------*/
/* SORTLEN  - LRECL of SORTIN                                        */
/* SORTPARM - PARM for SORT program                                  */
/*********************************************************************/
 stemsort: module = 'STEMSORT'
           if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           call modtrace 'START' sigl
/*********************************************************************/
/* Confirm parms                                                     */
/*********************************************************************/
           arg sortlen sortparm
           if datatype(sortlen) <> 'NUM' then
              call rcexit 54 'SORT record length is invalid'
           if sortparm = '' then
              call rcexit 55 'SORT parms missing'
           else
              sortparm = '  'sortparm
/*********************************************************************/
/* Sort the stem variable                                            */
/*********************************************************************/
           call saydd msgdd 1 'STEMSORT started:' strip(sortparm)
           call tsoquiet "FREE F(SYSIN SORTIN SORTOUT SYSOUT)"
           call stem 'sysin' 1 sortparm
           call viodd 'SYSIN'
           call viodd 'SORTIN' sortlen
           call tsotrap "ALLOC F(SYSOUT) UNIT("@vio") SPACE(1 1) CYL"
           call tsotrap "ALLOC F(SORTOUT) UNIT("@vio") SPACE(1 1)",
                        "CYLINDERS RECFM(F B) LRECL("sortlen")",
                        "BLKSIZE(Ø)"
           signal off failure
           address ATTCHMVS "SORT" "SORTPARM"
           SORTRC = RC
/*******************************************************************/
/* Sort error processing                                           */
/*******************************************************************/
           if SORTRC <> Ø then
              do
               call tsotrap "EXECIO * DISKR SYSOUT (STEM SYSOUT. FINIS"
               do serr=1 to sysout.Ø
                  if tsoenv = 'BACK' then
                     call saydd msgdd Ø strip(sysout.serr,'T')
                  else
                     say strip(sysout.serr,'T')
               end
               if SORTRC < Ø then
                  call rcexit SORTRC 'Sort error ('sortrc')'
               else
                  call rcexit SORTRC 'Sort error'
              end
           signal on failure name trap
/*******************************************************************/
/* Read the sorted records                                         */
/*******************************************************************/
           call tsotrap "EXECIO * DISKR SORTOUT (STEM SORTOUT. FINIS"
           call saydd msgdd 'Ø STEMSORT completed RC='SORTRC
           call tsotrap "FREE F(SYSIN SORTIN SORTOUT SYSOUT)"
/*******************************************************************/
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return sortout.Ø
/*********** @REFRESH END   STEMSORT 2ØØ4/11/18 11:13:58 *************/
/*********** @REFRESH BEGIN VIODD    2ØØ4/1Ø/13 17:54:25 *************/
/* VIODD    - EXECIO a stem into a sequential dataset              */
/*-----------------------------------------------------------------*/
/* VIODD    - The member to create                                 */
/* VIOLRECL - The LRECL for the VIODD (defaults to 8Ø)             */
/*******************************************************************/
 viodd: module = 'VIODD'
        if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg viodd violrecl viorecfm viofree
        if viodd = '' then call rcexit 88 'VIODD missing'
        if violrecl = '' then violrecl = 8Ø
        if viorecfm = '' then viorecfm = 'F B'
/*********************************************************************/
/* Resolve VIOFREE in case VIORECFM is present (retrofit)            */
/*********************************************************************/
        if viofree  = '' then
           viofree  = 'YES'
        else
           do
            viofw = words(viofree)
            if viofw > 1 then
               do
                do vw=1 to viofw-1
                   viorecfm = viorecfm word(viofree,vw)
                end
                viofree = word(viofree,viofw)
               end
            else
               do
                viorecfm = viorecfm viofree
                viofree = 'YES'
               end
           end
/*********************************************************************/
/* If VIOFREE is YES, reallocate                                     */
/*********************************************************************/
        if viofree = 'YES' then
           do
/*********************************************************************/
/* If DD exists, FREE it                                             */
/*********************************************************************/
            if listdsi(viodd 'FILE') = Ø then
               call tsotrap "FREE F("viodd")"
/*********************************************************************/
/* ALLOCATE a VIO DSN                                                */
/*********************************************************************/
            call tsotrap "ALLOC F("viodd") UNIT("@vio") SPACE(1 5)",
                         "LRECL("violrecl") BLKSIZE(Ø) REUSE",
                         "RECFM("viorecfm") CYLINDERS"
           end
/*********************************************************************/
/* Write the stem variables into the VIO DSN                         */
/*********************************************************************/
        call tsotrap "EXECIO * DISKW" viodd "(STEM" viodd". FINIS"
/*********************************************************************/
/* DROP the stem variable                                            */
```

```
/*********************************************************************/
        interpret 'drop' viodd'.'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   VIODD    2004/10/13 17:54:25 *************/
/*********** @REFRESH BEGIN G2J      2005/04/03 13:03:44 *************/
/* G2J      - Convert a Gregorian date to a Julian data              */
/*-------------------------------------------------------------------*/
/* SDATE    - Sortable date in YYYYMMDD format                       */
/*********************************************************************/
 g2j: procedure expose probe sigl modtrace
      module = 'G2J'
      if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/*********************************************************************/
/* Convert to Julian                                                 */
/*********************************************************************/
      arg indate
      parse var indate . 3 year 5 month 7 day
      mdays = '31 28 31 30 31 30 31 31 30 31 30 31'
      jdays = 0
      do i=1 to (month-1)
         jdays = jdays + word(mdays,i)
      end
      if year//4 = 0 & month > 2 then
         jdate = year||right(jdays+day+1,3,0)
      else
         jdate = year||right(jdays+day,3,0)
/*********************************************************************/
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return jdate
/*********** @REFRESH END   G2J      2005/04/03 13:03:44 *************/
/*********** @REFRESH BEGIN PTR      2002/07/13 15:45:36 *************/
/* PTR      - Pointer to a storage location                          */
/*-------------------------------------------------------------------*/
/* ARG(1)   - Storage Address                                        */
/*********************************************************************/
 ptr: return c2d(storage(d2x(arg(1)),4))
/*********** @REFRESH END   PTR      2002/07/13 15:45:36 *************/
/*********** @REFRESH BEGIN STG      2002/07/13 15:49:12 *************/
/* STG      - Return the data from a storage location                */
/*-------------------------------------------------------------------*/
/* ARG(1)   - Location                                               */
/* ARG(2)   - Length                                                 */
/*********************************************************************/
 stg: return storage(d2x(arg(1)),arg(2))
/*********** @REFRESH END   STG      2002/07/13 15:49:12 *************/
/*********** @REFRESH BEGIN MODTRACE 2003/12/31 21:56:54 *************/
/* MODTRACE - Module Trace                                           */
/*-------------------------------------------------------------------*/
/* TRACETYP - Type of trace entry                                    */
/* SIGLINE  - The line number called from                            */
/*********************************************************************/
 modtrace: if modtrace = 'NO' then return
           arg tracetyp sigline
           tracetyp = left(tracetyp,5)
           sigline = left(sigline,5)
/*********************************************************************/
/* Adjust MODSPACE for START                                         */
/*********************************************************************/
           if tracetyp = 'START' then
              modspace = substr(modspace,1,length(modspace)+1)
/*********************************************************************/
/* Set the trace entry                                               */
/*********************************************************************/
           traceline = modspace time('L') tracetyp module sigline sparms
/*********************************************************************/
/* Adjust MODSPACE for STOP                                          */
/*********************************************************************/
           if tracetyp = 'STOP' then
              modspace = substr(modspace,1,length(modspace)-1)
/*********************************************************************/
/* Determine where to write the traceline                            */
/*********************************************************************/
           if ispfenv = 'YES' & tsoenv = 'FORE' then
/*********************************************************************/
/* Write to the ISPF Log, do not use ISPWRAP here                    */
/*********************************************************************/
              do
               zedlmsg = traceline
               address ISPEXEC "LOG MSG(ISRZ000)"
              end
           else
              say traceline
/*********************************************************************/
/* SAY to SYSTSPRT                                                   */
/*********************************************************************/
           return
/*********** @REFRESH END   MODTRACE 2003/12/31 21:56:54 *************/
```

Robert Zenuk
Systems Programmer (USA)

REXX routines to format HSM command output

A useful command provided by HSM is the **F HSM,QUERY SPACE(volser)** command. This command can be used to find out how much free space exists on migration volumes and it allows you to take action in the event of space shortages in the pool of volumes that are used to store migrated datasets.

However, the command as supplied produces output that is not easy to read – see the example below:

```
-F HSM,QUERY SPACE(ML2021)
  ARC0400I VOLUME ML2021 IS 13% FREE, 00000419 FREE
  ARC0400I (CONT.) TRACK(S), 001355 FREE CYLINDER(S), FRAG .270
  ARC0401I LARGEST EXTENTS FOR ML2021 ARE CYLINDERS
  ARC0401I (CONT.) 367, TRACKS     5511
  ARC0402I VTOC FOR ML2021 IS 0074 TRACKS(03700 DSCBS),
  ARC0402I (CONT.) 03214 FREE DSCBS(86% OF TOTAL)
```

To overcome this limitation I have written two REXX EXECs that can be executed as part of a batch schedule to report on space usage and to reformat the output so that it is understandable.

The first REXX EXEC, HSMEAVOL, is used to read your HSM parameter library and extract any volumes that are added at HSM start up as part of the ADDVOL command. It then formats the extracted information and builds a stream of SDSF input cards. The source code for the REXX EXEC is shown below:

```
/* REXX */
/*       **********************************************************    */
/*       * PROGRAM NAME: HSMEAVOL                                 *    */
/*       *                                                        *    */
/*       * PROGRAMMER:   JOHN BRADLEY                             *    */
/*       *                                                        *    */
/*       * PURPOSE:      THIS ROUTINE WILL READ THE ARCCMDXX      *    */
/*       *               MEMBER PERTAINING TO HSM AND EXTRACT     *    */
/*       *               ANY VOLUMES USED FOR DATASET MIGRATION   *    */
/*       *               AND IT WILL PRODUCE AS OUTPUT CARDS      *    */
/*       *               TO INPUT TO SDSF TO PRODUCE SPACE        *    */
/*       *               INFORMATION ABOUT THOSE VOLUMES.         *    */
/*       *                                                        *    */
/*       * DATE:         16/11/2004.                              *    */
/*       *                                                      *    */
/*       * MODIFICATIONS.                                       *    */
/*       ********************************************************    */
MAINØØ1Ø:
I        = Ø                         /* INITIALIZE RECORD COUNTER. */
OUTREC1  ='/F HSM,QUERY SPACE('      /* MAIN OUTPUT RECORD.        */
OUTREC2  =')'                        /* END OF MAIN OUTPUT RECORD. */
SDSFCRD1 = 'SET CONSOLE CLONESYS'    /* SDSF RECORD 1.             */
SDSFCRD2 = 'SET DELAY 1Ø'            /* SDSF RECORD 2.             */
SDSFCRD3 = 'U'                       /* SDSF RECORD 3.             */
SDSFCRD4 = 'SET CONSOLE'             /* SDSF RECORD 4.             */
SDSFCRD5 = 'SET DELAY 1'             /* SDSF RECORD 5.             */
MAINØØ2Ø:
DROP RECORD.                         /* DROP STEM VARIABLE.        */
/* FOLLOWING OPENS DD NAMED 'IN' AND CHECKS OPEN WORKED.           */
"EXECIO * DISKR IN (FINIS STEM RECORD.)"
IF RC <> Ø THEN
  DO
   SAY "HSMAEØØ1I: ERROR OPENING INPUT DATASET - ROUTINE ENDING"
   SAY "HSMAEØØ2I: RETURN CODE - "||RC
   EXIT RC
  END
/* FOLLOWING OPENS DD NAMED 'OUT' AND CHECKS OPEN WORKED.          */
"EXECIO * DISKW OUT (OPEN)"
IF RC <> Ø THEN
  DO
   SAY "HSMAEØØ3I: ERROR OPENING OUTPUT DATASET - ROUTINE ENDING"
   SAY "HSMAEØØ2I: RETURN CODE - "||RC
   EXIT RC
  END
PUSH SDSFCRD2                        /* PUSH OUTPUT RECORD.        */
PUSH SDSFCRD1                        /* PUSH OUTPUT RECORD.        */
EXECIO 2 DISKW OUT                   /* WRITE 2 RECORDS TO DATASET.*/
DO RECORD.Ø                          /* LOOP FOR NUMBER OF RECORDS.*/
  I=I+1                              /* ADD 1 TO COUNT.            */
  RECORD.I = STRIP(RECORD.I)         /* REMOVE SPACES FROM RECORD. */
/* NEXT LINE PARSES THE RECORD INTO VARIABLES.                     */
  PARSE UPPER VAR RECORD.I R1 R2 R3 R4 R5 R6 R7 R8
  IF R1 = 'ADDVOL' THEN              /* FIRST WORD ADDVOL?         */
    DO                               /* YES THEN PROCESS.          */
     OUTPUT = OUTREC1||R2||OUTREC2   /* BUILD F HSM CARD.          */
     PUSH OUTPUT                     /* PUT IT ON STACK.           */
     EXECIO 1 DISKW OUT              /* WRITE IT OUT.              */
    END
END                                  /* GO TO START OF LOOP.       */
PUSH SDSFCRD5                        /* PUSH RECORD ONTO STACK.    */
PUSH SDSFCRD4                        /* PUSH RECORD ONTO STACK.    */
PUSH SDSFCRD3                        /* PUSH RECORD ONTO STACK.    */
EXECIO 3 DISKW OUT                   /* WRITE THEM OUT.            */
EXIT                                 /* FINISH PROCESSING.         */
```

The output it produces is shown below:

```
SET CONSOLE MAINSYS
SET DELAY 1Ø
/F HSM,QUERY SPACE(ML2Ø15)
/F HSM,QUERY SPACE(ML2Ø16)
/F HSM,QUERY SPACE(ML2Ø17)
/F HSM,QUERY SPACE(ML2Ø18)
/F HSM,QUERY SPACE(ML2Ø2Ø)
/F HSM,QUERY SPACE(ML2Ø2Ø)
/F HSM,QUERY SPACE(ML2Ø21)
U
SET CONSOLE
SET DELAY 1
```

The output is then used as input to a batch SDSF routine, which then writes its output to another dataset. This dataset is then read by the second REXX EXEC, HSMSPACE. This then produces the easily readable report. HSMSPACE is shown below:

```
/* REXX */
/*       ********************************************************    */
/*       * PROGRAM NAME: HSMSPACE                               *    */
/*       *                                                      *    */
/*       * PROGRAMMER:   JOHN BRADLEY                           *    */
/*       *                                                      *    */
/*       * PURPOSE:      THIS ROUTINE WILL READ OUTPUT FROM     *    */
/*       *               F HSM,QUERY SPACE AND EXTRACT SPECIFIC *    */
/*       *               INFORMATION THEN PRODUCE A REPORT THAT *    */
/*       *               IS MORE UNDERSTANDABLE AND EASIER TO   *    */
/*       *               VIEW THAN STANDARD COMMAND OUTPUT.     *    */
/*       *                                                      *    */
/*       * DATE:         16/11/2ØØ4.                            *    */
/*       *                                                      *    */
/*       * MODIFICATIONS.                                       *    */
/*       * --------------                                       *    */
/*       ********************************************************    */
MAINØØ1Ø:
VOLSER   = ' '                         /* INITIALIZE VARIABLE.       */
DISKFREE = ' '                         /* INITIALIZE VARIABLE.       */
TRKSFREE = ' '                         /* INITIALIZE VARIABLE.       */
CYLSFREE = ' '                         /* INITIALIZE VARIABLE.       */
CYLSLRG  = ' '                         /* INITIALIZE VARIABLE.       */
TRKSLRG  = ' '                         /* INITIALIZE VARIABLE.       */
LINECNT  = Ø                           /* INITIALIZE VARIABLE.       */
FLAG1    = Ø                           /* INITIALIZE VARIABLE.       */
FLAGSTRT = Ø                           /* INITIALIZE VARIABLE.       */
FLAGSTOP = Ø                           /* INITIALIZE VARIABLE.       */
I        = Ø                           /* INITIALIZE VARIABLE.       */
```

```
PAGNO    = Ø                           /* INITIALIZE VARIABLE.        */
/* INITIALIZE PRINT OUTPUT HEADER LINES.                              */
HDR1 = ' '
HDR2 = ' VOLUME   DISKFREE   TRKSFREE   CYLSFREE   LRGCYLS   LRGTRKS'
HDR3 = ' ------   --------   --------   --------   ------   ------'
HDRPG1 = '        PAGE: '
MAINØØ2Ø:
DROP RECORD.                           /* CLEAR STEM RECORDS.         */
/* OPEN INPUT DATASET. CONTAINS SDSF OUTPUT FROM F HSM COMMANDS.      */
/* IF OPEN FAILS THEN ISSUE ERROR MESSAGES.                           */
"EXECIO * DISKR IN (FINIS STEM RECORD.)"
IF RC <> Ø THEN
  DO
   SAY "HSMSPØØ1I: ERROR OPENING INPUT DATASET - ROUTINE ENDING"
   SAY "HSMSPØØ2I: RETURN CODE - "||RC
   EXIT RC
  END
/* OPEN OUTPUT DATASET.                                               */
/* IF OPEN FAILS THEN ISSUE ERROR MESSAGES.                           */
"EXECIO * DISKW OUT (OPEN)"
IF RC <> Ø THEN
  DO
   SAY "HSMSPØØ3I: ERROR OPENING OUTPUT DATASET - ROUTINE ENDING"
   SAY "HSMSPØØ2I: RETURN CODE - "||RC
   EXIT RC
  END
DO RECORD.Ø                            /* LOOP FOR NUMBER OF RECORDS.*/
  I=I+1                                /* INCREMENT RECORD COUNTER.  */
  RECORD.I = STRIP(RECORD.I)           /* REMOVE BLANKS FROM RECORD. */
/* NEXT LINE GETS RECORD AND PLACES IT INTO VARIABLES.                */
  PARSE UPPER VAR RECORD.I R1 R2 R3 R4 R5 R6 R7 R8
  IF FLAGSTOP = 1 THEN EXIT            /* IF STOP SET THEN FINISHED. */
  IF R2 = 'TOP' THEN FLAGSTRT = 1      /* TOP OF SDSF? SET FLAG.     */
  IF R2 = 'BOTTOM' THEN FLAGSTOP = 1   /* END OF SDSF? SET FLAG.     */
/* IF ARC4ØØI MESSAGE AND FLAGSTRT SET THEN PROCESS.                  */
  IF SUBSTR(RECORD.I,1,8) = 'ARCØ4ØØI' & FLAGSTRT = 1 THEN
    DO
      IF R2 = 'VOLUME' THEN            /* VOLUME RECORD?             */
        DO
         VOLSER = R3                   /* EXTRACT INFO AND FORMAT.   */
         DISKFREE = R5
         DISKFREE = STRIP(DISKFREE,B,' ')
         TRKSFREE = R7
         TRKSFREE = STRIP(TRKSFREE,L,Ø)
        END
      ELSE                             /* OTHERWISE PROCESS ARC4ØØI  */
        DO                             /* CONTINUATION RECORD.       */
         CYLSFREE = R4
         CYLSFREE = STRIP(CYLSFREE,L,Ø)
         FLAG1 = 1                     /* SHOW ARC4ØØI COMPLETE.     */
        END
    END
/* IF ARC4Ø1I MESSAGE AND FLAGSTRT SET THEN PROCESS.                  */
  IF SUBSTR(RECORD.I,1,8) = 'ARCØ4Ø1I' & FLAGSTRT = 1 THEN
    DO
      IF R2 = "(CONT.)" THEN            /* ONLY WANT THE CONT PART?   */
        DO
         CYLSLRG = R3                   /* IF VALID THEN PROCESS AND  */
         TRKSLRG = R5                   /* FORMAT THE OUTPUT.         */
         CYLSLRG = STRIP(CYLSLRG,L,Ø)
         CYLSLRG = STRIP(CYLSLRG,T,',')
         TRKSLRG = STRIP(TRKSLRG,L,Ø)
         FLAG1 = FLAG1 + 1              /* FINISHED THE ARC4Ø1I.      */
        END
    END
  IF FLAG1 = 2                          /* RECORD READY TO WRITE?     */
    THEN
     DO
/* NEXT LINE CHECKS TO SEE WHETHER HEADER PAGE REQUIRED. IF SO       */
/* THEN SUBROUTINE HEADER IS CALLED.                                  */
      IF LINECNT = Ø | LINECNT >= 64 THEN CALL HEADER
      CALL FMTOPUT                      /* CALL SUBROUTINE TO FORMAT. */
      PUSH OUTPUT                       /* PUSH TO STACK.             */
      EXECIO 1 DISKW OUT                /* WRITE THE RECORD.          */
      FLAG1 = Ø                         /* RESET FLAG.                */
      LINECNT = LINECNT + 1             /* ADD 1 TO LINE COUNT.       */
     END
END
EXIT                                    /* END OF MAIN PROGRAM.       */
/* HEADER SUBROUTINE WRITES PAGE HEADERS AND NUMBERS TO REPORT.       */
HEADER:
   LINECNT = Ø                          /* RESET LINE COUNT.          */
   PAGNO = PAGNO + 1                    /* INCREMENT PAGE NUMBER.     */
   PUSH HDR1                            /* PUSH FIRST RECORD.         */
   PUSH HDR3                            /* PUSH SECOND RECORD.        */
   PUSH HDR2||HDRPG1||PAGNO             /* PUSH THIRD RECORD.         */
   PUSH HDR1                            /* PUSH FOURTH RECORD.        */
   EXECIO 4 DISKW OUT                   /* WRITE IT OUT.              */
   LINECNT = LINECNT + 4                /* ADD 4 TO LINE COUNT.       */
RETURN                                  /* GO BACK TO MAIN ROUTINE.   */
/* FMTOPUT ROUTINE FORMATS OUTPUT RECORD PRIOR TO WRITING IT.         */
FMTOPUT:
 DISKFREE = CENTER(DISKFREE,8)          /* CENTER VARIABLE.           */
 CYLSFREE = CENTER(CYLSFREE,8)          /* CENTER VARIABLE.           */
 TRKSFREE = CENTER(TRKSFREE,8)          /* CENTER VARIABLE.           */
 CYLSLRG = CENTER(CYLSLRG,8)            /* CENTER VARIABLE.           */
 TRKSLRG = CENTER(TRKSLRG,8)            /* CENTER VARIABLE.           */
 OUTPUT = ' '||VOLSER||,                /* BUILD OUTPUT RECORD.       */
          '   '||,
          DISKFREE||,
          '   '||,
          TRKSFREE||,
          '   '||,
          CYLSFREE||,
          '   '||,
          CYLSLRG||,
          '   '||,
          TRKSLRG
RETURN                                 /* RETURN TO MAIN ROUTINE.    */
```

The final report is shown below:

```
 VOLUME   DISKFREE   TRKSFREE  CYLSFREE   LRGCYLS  LRGTRKS  PAGE: 1
 ------   --------   --------  --------   -------  -------

 ML2Ø15       38%         49Ø     382Ø        965    14491
 ML2Ø16       16%         519     1659        399     6ØØ2
 ML2Ø17       3Ø%         478     3Ø44        6Ø7     9117
 ML2Ø18       21%         914     2142        263     3965
 ML2Ø19       14%         889     144Ø        333     5Ø18
 ML2Ø2Ø       24%         645     2382        365     5489
 ML2Ø21       13%         419     1355        367     5511
```

Compare this with the original output. The batch JCL to run both routines and to do the necessary housekeeping is shown below:

```
//JXB7884S      JOB   (GEN),'7884',CLASS=A,NOTIFY=JXB7884
//*      *********************************************************
//*      * QUERY HSM ML1 SPACE USAGE.                            *
//*      *********************************************************
//STEP1       EXEC  PGM=IDCAMS
//SYSPRINT    DD    SYSOUT=*
//SYSIN       DD    *
  DEL JXB7884.SDSFCARD,PURGE
  DEL JXB7884.HSMVOL,PURGE
  DEL JXB7884.HSMSPACE,PURGE
  IF MAXCC=8 THEN SET MAXCC=Ø
/*
//*
//STEP2       EXEC  PGM=IRXJCL,PARM='HSMEAVOL',REGION=4M
//SYSTSPRT    DD    SYSOUT=*,RECFM=FBA,LRECL=133,BLKSIZE=Ø
//SYSEXEC     DD    DSN=JXB7884.REXX.LIB,DISP=SHR
//SYSTSIN     DD    DUMMY
//IN          DD    DSN=HSM.PARMLIB(ARCCMDØØ),DISP=SHR
//OUT         DD    DSN=JXB7884.SDSFCARD,DISP=(,CATLG,DELETE),
//            SPACE=(TRK,(1,1)),DCB=(RECFM=FB,DSORG=PS,LRECL=8Ø)
//*
//STEP3       EXEC  PGM=SDSF
//ISFOUT      DD DSN=JXB7884.HSMVOL,DISP=(,CATLG),
//              SPACE=(TRK,(1,1))
//ISFOUT2       DD DSN=JXB7884.HSMSPACE,DISP=(,CATLG),
//              SPACE=(TRK,(1,1)),
//              DCB=(DSORG=PS,RECFM=FB,LRECL=8Ø,BLKSIZE=Ø)
//ISFIN         DD    DSN=JXB7884.SDSFCARD,DISP=SHR
//*
//STEP4         EXEC  PGM=IRXJCL,PARM='HSMSPACE',REGION=4M
//SYSTSPRT      DD    SYSOUT=*,RECFM=FBA,LRECL=133,BLKSIZE=Ø
//SYSEXEC       DD    DSN=JXB7884.REXX.LIB,DISP=SHR
//SYSTSIN       DD    DUMMY
//IN            DD    DSN=JXB7884.HSMVOL,DISP=SHR
//OUT           DD    DSN=JXB7884.HSMSPACE,DISP=SHR
//*
```

We run this batch job once a week for each system. The only changes you will need to make to the batch JCL should be the usual job card and dataset name changes.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

DFSORT performance tuning aid

As we all know, sorting is one of the most important parts of a site’s data processing workload. It is difficult to think of a non-trivial data processing application that does not use sorting. DFSORT is often invoked hundreds or thousands of times per day in a typical MVS installation – it is perhaps the most commonly invoked batch program. Moreover, most installations have experienced explosive growth in recent years in the volume of data to be stored and processed, both in operational applications and in decision-support and data warehousing. Thus, performance of a sort is the issue of prime importance. More efficient use of DFSORT can lead to the reduced use of system resources and reduced job turnaround time, as well as to improved productivity. When one considers how much resource is used for sort jobs and how critical sorting is to the data processing production environment, it becomes clear that the investment in time and resources to build a truly representative sort performance test is well worth the effort. The consequence of not making that investment may be that the selected sort product does not achieve the anticipated performance levels, and the throughput of the overall data processing environment will be negatively impacted. The most important task in running a sort performance test is the proper selection of the jobs that make up the test. Constructing a sort performance test requires the most effort. The amount of effort required varies according to the degree of analysis that you want to perform. The analysis can range from selecting jobs that can be easily run to analysing the sort workload in the production environment. The effort should result in selecting sort jobs that are either frequently run, or the biggest consumers of system resources, or both. When constructing a sort performance test, one should consider selecting only those sort jobs that are representative of sorts run in the production environment.

When selecting sort performance test candidates, look for sort jobs that have one or more of the following characteristics:

- Use a relatively large percentage of CPU resource.
- Have a relatively large amount of I/O activity.
- Are long running and must run within a fixed batch window.
- Are frequently run (one or more times per day).
- Use sort options and functions found in production jobs, such as exits, record level editing (INREC, OUTREC, and OUTFIL), record filtering (INCLUDE, OMIT, and OUTFIL), record summarization (SUM), multiple output datasets (OUTFIL), and reports (OUTFIL).

Once the sort performance test job characteristics are identified, one can obtain the jobs that make up a sort test directly from the production environment, or one can develop a prototype of the sort jobs. Whichever is chosen, one should select sort jobs that represent the following characteristics in the majority of production jobs since these characteristics will most likely have the greatest impact on sort performance:

- Dataset size (number of records, number of bytes).
- Record length.
- Sort key length.
- Record format (fixed, variable, spanned).
- Input and output datasets: DASD or tape, striped or compressed.
- Number and location of work datasets.
- Number of output datasets.
- Sort product functions and options (for example exits, INREC, OUTREC, INCLUDE, OMIT, SUM, and OUTFIL).
- Virtual storage size.
- Amount of hiperspace available.
- Amount of data space available.

Now that you understand how to select sort performance test jobs, it is also of vital importance to understand how to run sort performance tests. The three most common problems that occur when conducting performance comparisons between sort tests are:

- Non-repeatable results.
- Statistical bias.
- Lack of planning to determine which data will be measured and how the measurements will be obtained.

The single most important item when running any performance test is the ability to repeat the results. Results that are not repeatable can invalidate a test or, even worse, lead to misinterpretations of the results and erroneous conclusions.

For sort jobs, elapsed time is very susceptible to large variations, CPU time is moderately susceptible, and EXCP counts have very low variation.

Also, one should develop a plan to determine which data one will measure and how to collect the measurement information. The data most commonly collected for sort performance tests includes CPU time, elapsed time, device connect time, and EXCP counts. SMF type 30 (subtype 4) records provide CPU time and elapsed time along with EXCP counts and device connect times for each sort step.

Elapsed time is the amount of ‘wall clock’ time from job initiation to job termination. For typical sorting applications, elapsed time is composed primarily of I/O time, with CPU time and I/O queueing time also contributing significantly. It is best collected by using SMF type 16 and 30 records. Elapsed time is important in all shops and is critical when the workload distribution creates limited batch windows.

You can collect CPU time for each job through SMF used to measure the amount of work performed by the processor, as opposed to work performed by the I/O and storage subsystems. The SMF type 30 record breaks down CPU time into five fields:

- TCB time – this is the dominant component in a sort job and represents the CPU time used to perform user program activity on behalf of user tasks for a job step.
- SRB time is the CPU time used to perform system service requests on behalf of user tasks for a job step. This is a fairly small percentage of the total CPU time.
- Region Control Task (RCT) time – if the user task is swapped out while running, this is the amount of CPU time used to swap the task out and back in again. This is not a significant component of the total CPU time.
- I/O Interrupt Processing (IIP) time is the CPU time used to handle I/O interrupts on behalf of user tasks for a job step. This is not a significant component of the total CPU time.

- Hiperspace Processing Time (HPT) – this component is significant if you are using hiperspaces (such as with hipersorting). In other words, if the user task reads from or writes to a hiperspace using the HSPSERV macro, this is the amount of CPU time used to service these reads and writes.

Note: SMF automatically receives CPU consumption data when a job is in problem program state and, in fact, DFSORT performs all sort activity in problem program state. So the SMF type 30 (subtype 4) records accurately reflect DFSORT CPU consumption.

The first step in any tuning exercise is, naturally, to audit the installation options of the software. For DFSORT, this is accomplished by using the ICETOOL DEFAULTS operator with a job like this one:

```
//DEFAULTS  JOB
//SHOWDEF   EXEC PGM=ICETOOL
//TOOLMSG   DD   SYSOUT=A
//DFSMSG    DD   SYSOUT=A
//LIST1     DD   SYSOUT=A
//TOOLIN    DD   *
  DEFAULTS LIST(LIST1)
/*
```

The output from DEFAULTS can help you analyse how you are using DFSORT and is beneficial to have regardless of the level of analysis you are performing. Descriptions of all the ICEMAC options can be found in *DFSORT Installation and Customization.* Of course, many of these installation option values can be overridden at run-time by one or more of the following:

- An installation-written DFSORT initialization exit, ICEIEXIT.
- An application EXEC statement parameter or DFSPARM statement parameter.
- An application OPTION statement, supplied either via DFSPARM or SYSIN.

In addition, a number of values (for example hiperspace and dataspace pages allocated) may be determined by the availability of resources at the time of execution. Therefore, it remains necessary to examine the output (or SMF) data of each individual DFSORT task to determine the option values that may have affected the performance of a particular task.

In general, there are two primary objectives when tuning DFSORT performance options: elapsed time reduction and elapsed time scalability – which in fact means the ability to increase data volumes with the sole impact being a near-linear increase in elapsed time. In essence, the struggle to reduce the elapsed times of DFSORT tasks reduces to an I/O contest. Stated in simple terms, the options available are to reduce the I/O, accelerate the I/O, convert the I/O from random to sequential or to in-storage processing, and finally to parallelize the I/O. Although DFSORT automatically optimizes many of its tuning decisions, there are additional actions that a site and its DFSORT users can take to further improve DFSORT performance. However, please note that an improvement in performance is not free. There are some potential trade-offs: increased paging, increased swapping, increased CPU time, and changes needed to applications.

COLLECTING DFSORT PERFORMANCE DATA

DFSORT performance indicators can be found in a number of different places. Where one chooses to find them depends on one's own knowledge of z/OS, available tools, and how much effort one wants to spend. If you can take the time to do some analysis of all your DFSORT applications, you can pinpoint more precisely where to tune your use of DFSORT to perform more efficiently. The easiest way to start is to use information optionally provided by DFSORT. To do this, you can use data from SMF records. Also, a useful directory of the sources and potential analysis activities for DFSORT performance data is given in *DFSORT Tuning Guide* (SC26-7526), and you are strongly advised to take a closer look at recommendations provided therein.

Of immense value in tracking and analysing the performance of DFSORT tasks is SMF record type 16. The layout and contents of this record can be found described in detail in Appendix D of *DFSORT Installation and Customization* (SC26-7524-00), or by expanding the ICESMF macro supplied in the ICEUSER dataset at installation. The DFSORT SMF type 16 record contains a wealth of performance data for each DFSORT task executed. If you want to produce DFSORT SMF records, DFSORT's supplied ICEMAC default of SMF=NO must be changed to SMF=FULL or SMF=SHORT, or SMF=FULL or SMF=SHORT must be specified on an OPTION statement in DFSPARM, or in the extended parameter list at run-time. This must be done in addition to setting up the appropriate SMFPRM*xx* member. The DFSORT SVC must be available for DFSORT to be able to write the SMF records to an SMF dataset. Refer to *Making the DFSORT SVC Available* in *DFSORT Tuning Guide* (SC26-7526) for additional information. Some of the fields in the type 16 record containing performance-related data are:

- Type of application (sort, merge, or copy).
- DFSORT technique used.
- Elapsed, TCB, and SRB time used.
- Type and length of records being sorted.
- Total bytes and records sorted.
- Work dataset allocation data.
- Access methods and block sizes used.
- Control field length.
- Amount of hiperspace, data space, and memory object storage used.
- Number of input and output dataset I/O calls.
- Number of work dataset EXCPs.

CODE

In order to extract DFSORT performance information from SMF data, I have constructed a three-part job stream. In the first part (DUMP16), SMF records 16 are extracted from the SMF weekly dataset to a file, which can be used as a base of archived records. Please note that the sorting of SMF data may issue an error message (ICE204A), set a return code of 16, and terminate if it detects an incomplete spanned record. In order to overcome this potential obstacle, DFSORT's SPANINC=RC4 option was used to remove the incomplete spanned records. It should be noted that SPANINC=RC0 tells DFSORT (Release 14) to issue a warning message, set a return code of 0, and eliminate all incomplete spanned records it detects. Valid records (that is complete spanned records) are recovered and written to the output dataset, while SPANINC=RC4 does the same thing as SPANINC=RC0, but with a return code of 4 instead of 0. The shipped default is SPANINC=RC16.

In the second step (SORT16) previously extracted records (selection being defined by INCLUDE's condition) are sorted and copied to a file, which is then input to the analysis and reporting SORTPERF EXEC invoked in the REX16 step.

Four reports are generated by this REXX EXEC. The first one is *DFSORT performance report*, which provides all essential timing measurements and, in addition to that, the I/O processing rate as well as sorting rate. The next two reports (*DFSORT file report* and *DFSORT VB file report*) are sort file-oriented, providing a summary of the files being used by each sort job. Finally, the REXX routine was modified to parse basic SMF type 16 data into a CSV format file for downloading to a spreadsheet or database – perhaps the most amenable medium for analysis.

Sample JCL to execute SMF type 16 data extract:

```
//*--------------------------------------------------------------
//* The SMF extract job.
//* This job will extract the SMF records type 16 from the SMF
//* weekly dataset
```

```
//*-----------------------------------------------------------
//DUMP16    EXEC PGM=IFASMFDP,REGION=ØM
//INDD      DD DSN=your.smf.weekly.ds,DISP=SHR
//OUTDD     DD DSN=&&SMF16,DISP=(NEW,PASS),
//          SPACE=(CYL,(25,5)),DCB=(your.smf.weekly.ds)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
    INDD(INDD,OPTIONS(DUMP))
    OUTDD(OUTDD,TYPE(16))
/*
//SORT16    EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=&&SMF16,DISP=SHR
//SMF16    DD DSN=your.rec16.ds,
//          SPACE=(CYL,(6,1)),DISP=(NEW,KEEP),
//          DCB=(RECFM=VB,LRECL=32756,BLKSIZE=3276Ø)
//DFSPARM  DD *
      OPTION SMF=FULL
//TOOLIN   DD *
  SORT FROM(RAWSMF) TO(SMF16) USING(SMFI)
//SMFICNTL DD *
*
* Eliminate header and trailer records
* Sort by ended date and time
*
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,16)
  SORT FIELDS=(11,4,PD,A,7,4,BI,A)
/*
//REX16     EXEC PGM=IKJEFTØ1,REGION=ØM,DYNAMNBR=5Ø
//SYSEXEC  DD DISP=SHR,DSN=your.rexx.lib
//SYSTSPRT DD SYSOUT=*
//SMF       DD DD DSN=your.rec16.ds,DISP=SHR
//SYSTSIN  DD *
prof nopref
%SORTPERF
/*
```

SORTPERF EXEC

```
/* REXX  to process the SMF records Type 16 - DFSORT Statistics. */
/* Record type 16 is written to record information about events  */
/* and operations of the DFSORT program.                         */
/* Depending on the option specified at initialization           */
/* a SHORT record, FULL record, or NO record is produced.         */
/* Some information in the SMF record will not be provided for   */
/* certain types of abnormal endings.                            */
ADDRESS TSO
```

```
numeric digits 16
userid=SYSVAR(SYSUID)
sorr = userid||'.sort.rpt'      /* DFSORT performance report file */
sorf = userid||'.sortf.rpt'     /* DFSORT file report dataset     */
sorv = userid||'.sortv.rpt'     /* DFSORT VB file report dataset  */
csvr = userid||'.sort.csv'      /* DFSORT CSV file summary report */
x = MSG('ON')
IF SYSDSN(sorr) = 'OK'
   THEN "DELETE "sorr" PURGE"
IF SYSDSN(sorf) = 'OK'
   THEN "DELETE "sorf" PURGE"
IF SYSDSN(sorv) = 'OK'
   THEN "DELETE "sorv" PURGE"
IF SYSDSN(csvr) = 'OK'
   THEN "DELETE "csvr" PURGE"
"ALLOC FILE(SORT16) DA("sorr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(7Ø,9) CATALOG",
   " REUSE RELEASE LRECL(22Ø) RECFM(F B)"
"ALLOC FILE(SORTFI) DA("sorf")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(7Ø,9) CATALOG",
   " REUSE RELEASE LRECL(95)  RECFM(F B)"
"ALLOC FILE(SORTFV) DA("sorv")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(7Ø,9) CATALOG",
   " REUSE RELEASE LRECL(165) RECFM(F B)"
"ALLOC FILE(SORTCS) DA("csvr")",
   " UNIT(SYSALLDA) NEW TRACKS SPACE(7Ø,9) CATALOG",
   " REUSE RELEASE LRECL(22Ø) RECFM(F B)"
/*---------------------------------------------------------------*/
/* Report header for DFSORT performance report file              */
/*---------------------------------------------------------------*/
hdr.1 = left('DFSORT PERFORMANCE TUNING REPORTER',6Ø)
hdr.2 = left(' ',1)
hdr.3 = left('Date',11)          left('started',11)       ,
        left('Jobname',8)        left('Technique',1Ø)     ,
        right('rc',3)            right('rs',3)            ,
        right('Elapsed',11)      right('Clock',6)         ,
        right('CPUtm',7)         right('Tcb',7)           ,
        right('Srb',5)           right('In rec',1Ø)       ,
        right('Sort rec',1Ø)     right('KB sorted',1Ø)    ,
        right('Sort rate 1',12)  right('Sort rate 2',12)  ,
        right('Excp',7)          right('Io rate 1',1Ø)    ,
        right('Io rate 2',1Ø)    right('HSPu',6)          ,
        right('DSPu',4)          right('Memory only sorting?',22)
 hdr.4 = left('-',217,'-')
 "EXECIO * DISKW SORT16 (STEM hdr.)"
 ddr.1 = left('DFSORT FILE REPORT',22)
 ddr.2 = left(' ',1)
 "EXECIO * DISKW SORTFI (STEM ddr.)"
/*---------------------------------------------------------------*/
/* Report header for DFSORT VB file report                       */
```

```
/*---------------------------------------------------------------*/
vin.1 = left('DFSORT VB FILE RECORD LEN. DISTRIBUTION REPORT',5Ø)
vin.2 = left(' ',1)
vin.3 = left('Date',11)      left('started',11)     ,
        left('Jobname',8)    left('Records in',11)  ,
        left('int.Ø ',6)     left('int.1 ',6) ,
        left('int.2 ',6)     left('int.3 ',6) ,
        left('int.4 ',6)     left('int.5 ',6) ,
        left('int.6 ',6)     left('int.7 ',6) ,
        left('int.8 ',6)     left('int.9 ',6) ,
        left('int.1Ø',6)     left('int.11',6) ,
        left('int.12',6)     left('int.13',6) ,
        left('int.14',6)     left('int.15',6)  left('int.16',6)
vin.4 = left('-',163,'-')
 "EXECIO * DISKW SORTFV (STEM vin.)"
/*---------------------------------------------------------------*/
/* Report header for DFSORT summary CSV file                     */
/*---------------------------------------------------------------*/
csh.1 = 'Date'||";"||'Time'||";"||'Lpar'||";"||'Job'||";"||,
        'Step #'||";"||'Stepname'||";"||'Termination'||";"||,
        'Sorttype'||";"||'Elapstm'||";"||'CPUtm'||";"||'Service'||";"||,
        'rec. sorted'||";"||'avg. lrecl'||";"||'bytes compared'||";"||,
        'bytes sorted'||";"||'alloc wrk ds'||";"||,
        'wrk ds tracks'||";"||'sortin i/o'||";"||'sortout i/o'||";"||,
        'sortwknn excp'||";"||'hiperspace pages'||";"||,
        'data space pages'
 "EXECIO * DISKW SORTCS (STEM csh.)"
/*---------------------------------------------------------------*/
/* SU/sec calculations (1 cpu second equals "sufa" serv. units   */
/*---------------------------------------------------------------*/
cvt  = storage(1Ø,4)
rmct = storage(d2x(c2d(cvt)+6Ø4),4)
su   = storage(d2x(c2d(rmct)+64),4)
sufa = 16ØØØØØØ/c2d(su)

'EXECIO *  DISKR SMF ( STEM x. FINIS'
 do i = 1 to x.Ø
/*---------------------------------------------------------------*/
/* function: define the header, product, data, record length     */
/*          distribution, and dataset sections of the smf        */
/*          record type 16 written by dfsort.                    */
/* source: icesmf (sys1.siceuser)                                */
/*---------------------------------------------------------------*/
/* Standard SMF header                                           */
/*---------------------------------------------------------------*/
icertyp  = c2d(substr(x.i,2,1))         /* smf record type           */
icebtime = smf(c2d(substr(x.i,3,4)))
                                 /* Time rec. was moved to smf buff. */
icebdate = substr(c2x(substr(x.i,7,4)),3,5) /* smf write date        */
icesid   = substr(x.i,11,4)             /* system identification     */
```

```
icejobnm = substr(x.i,15,8)             /* jobname                    */
icerst   = smf(c2d(substr(x.i,23,4)))
                                  /*  time reader recognized jobcard  */
icerds   = substr(c2x(substr(x.i,27,4)),3,5)
                                          /* date recognized jobcard */
iceuif   = substr(x.i,31,8)             /* installation specific info*/
icestn   = c2d(substr(x.i,39,1))        /* step no.                   */
iceres1  = c2d(substr(x.i,4Ø,1))        /* reserved                   */
icetrn   = c2d(substr(x.i,41,2))        /* no. of section descriptors*/
icesubid =     substr(x.i,43,4)         /* subsystem id               */
icersub  = c2d(substr(x.i,47,2))        /* record subtype             */
select
 when icersub = 1 then subtyp = 'short,ok'  /*short, successful exec */
 when icersub = 2 then subtyp = 'full ,ok'  /*full, successful exec  */
 when icersub = 3 then subtyp = 'short,failed'
                                         /*short, unsuccessful exec  */
 otherwise             nop
end
/*-------------------------------------------------------------------*/
/* Self-defining section                                             */
/*-------------------------------------------------------------------*/
iceprod  = c2d(substr(x.i,49,4))     /* offset to product section    */
iceprodl = c2d(substr(x.i,53,2))     /* product section len.         */
iceprodn = c2d(substr(x.i,55,2))     /* no. of product sections      */
icedata  = c2d(substr(x.i,57,4))
                               /* offset to sec.common to short/recs  */
icedatal = c2d(substr(x.i,61,2))     /* data section len.            */
icedatan = c2d(substr(x.i,63,2))     /* no. of data sections         */
icestat  = c2d(substr(x.i,65,4))     /* offset to record len. stat   */
icestatl = c2d(substr(x.i,69,2))     /* record len. stat section len.*/
icestatn = c2d(substr(x.i,71,2))     /* no. of record len. stat sect.*/
iceinds  = c2d(substr(x.i,73,4))  /*offset to 1st input dataset sect.*/
iceindsl = c2d(substr(x.i,77,2))     /*len. of input dataset section */
iceindsn = c2d(substr(x.i,79,2))     /* no. of input dataset sections*/
iceotds  = c2d(substr(x.i,81,4))  /*offset to sortout dataset section*/
iceotdsl = c2d(substr(x.i,85,2))    /*len. of sortout dataset section*/
iceotdsn = c2d(substr(x.i,87,2))    /*no. of sortout dataset sections*/
iceofds  = c2d(substr(x.i,89,4)) /*offset to 1st outfil dataset sect.*/
iceofdsl = c2d(substr(x.i,93,2))     /*len. of outfil dataset section*/
iceofdsn = c2d(substr(x.i,95,2))     /*no. of outfil dataset sections*/
iceres1b = c2d(substr(x.i,97,2))     /* reserved                     */
icespgn  = c2d(substr(x.i,99,2))     /* performance group no.        */
iceuser  =     substr(x.i,1Ø1,8)  /*user id of job of session in exec*/
icegroup =     substr(x.i,1Ø9,8)
                               /* group id of job of session in exec  */
iceres1d = c2d(substr(x.i,117,8))    /* reserved - do not use        */
/*-------------------------------------------------------------------*/
/* Product section                                                   */
/*-------------------------------------------------------------------*/
if iceprod  >Ø  then do
```

```
   pf  = iceprod - 3
icerecv  =     substr(x.i,pf,2)           /* record version.        */
iceprdct =     substr(x.i,pf+2,8)         /*product name. '574Øsm1 '*/
icerelnm =     substr(x.i,pf+1Ø,4)        /*dfsort release level no.*/
iceres1c = c2d(substr(x.i,pf+14,2))       /* reserved               */
 end
/*------------------------------------------------------------------*/
/* Data section. Common part                                         */
/*------------------------------------------------------------------*/
if icedatan  > Ø  then do
   df  = icedata - 3
iceres2  = c2d(substr(x.i,df,2))          /* reserved               */
icestpnm = substr(x.i,df+2,8)             /* stepname               */
icercds  = c2d(substr(x.i,df+1Ø,4))       /* no. of rec. sorted     */
icebytes = c2d(substr(x.i,df+14,4))       /* no. of bytes sorted    */
icecput  = c2d(substr(x.i,df+18,4))*Ø.Ø1
                                   /* sort cpu tcb, Ø.Ø1 of a sec.  */
                                        /* set to zero if STIMER=NO */
icelen   = c2d(substr(x.i,df+22,2))       /* specified record len.  */
iceiblk  = c2d(substr(x.i,df+24,2))  /* input blksize (max) or cisize*/
iceoblk  = c2d(substr(x.i,df+26,2))     /* output blocksize or cisize*/
icekeyln = c2d(substr(x.i,df+28,2))  /* total control field len.(#.of*/
                                          /* bytes compared by sort)*/
icewblk  = c2d(substr(x.i,df+3Ø,4)) /* total # of work ds tracks used*/
iceflbyt = x2b(c2x(substr(x.i,df+34,1)))
 iceflbyØ= substr(iceflbyt,1,1)           /* bit Ø: reserved        */
 icerctf = substr(iceflbyt,2,2)       /* bit 1 - 2: record len. type */
/*-----------------------------------------------------------------*/
/* Note:                                                           */
/* in case of fixed length records, the short form of the SMF record */
/* is produced, even if the user has specified the full smf record.  */
/*-----------------------------------------------------------------*/
 select
  when icerctf = 'ØØ' then reclen = 'F, FB'    /*    fixed len. rec. */
  when icerctf = 'Ø1' then reclen = 'V, VB'    /* variable len. rec  */
  when icerctf = '1Ø' then reclen = 'VS, VBS'
                                    /* variable len. spanned rec. */
  otherwise nop
 end
 iceblkst= substr(iceflbyt,4,2)       /* bit 3-4: sort tech.ue used  */
/*-----------------------------------------------------------------*/
/* Note:                                                           */
/* in case of conventional technique the following fields are      */
/* provided - all of the fields in the header section except icetrn, */
/*   iceuser and icegroup.                                         */
/* - all of the fields in the product section                      */
/* - icestpnm and the technique and program invoked flags of iceflbyt*/
/*   in the data section.                                          */
/*-----------------------------------------------------------------*/
 select
```

```
   when iceblkst = 'ØØ' then sortt ='Blockset'
   when iceblkst = 'Ø1' then sortt ='Peerage '
   when iceblkst = '1Ø' then sortt ='Vale    '
   when iceblkst = '11' then sortt ='Conventional'
   otherwise nop
  end
 icevoked = substr(iceflbyt,6,1)           /* bit 5: sort invocation  */
  select
   when icevoked ='1' then inv = 'DFSORT was invoked through a program'
   otherwise      inv = '  '
  end
 iceinmem = substr(iceflbyt,7,1)        /* bit 6: memory only sorting?*/
  select
   when iceinmem = '1' then stor = 'Main storage sorting'
   otherwise                stor = 'Sort work space used'
  end
 icendyna = c2d(substr(x.i,df+35,1))      /*no. of alloc work datasets*/
 /*-----------------------------------------------------------------*/
 /* Note: if all bits of iceflby2 are zero, then the function to be  */
 /*       performed could not be determined.                         */
 /*-----------------------------------------------------------------*/
 iceflby2 = x2b(c2x(substr(x.i,df+36,1)))       /* type of operation  */
  icesort = substr(iceflby2,1,1)
  icemerg = substr(iceflby2,2,1)
  icecopy = substr(iceflby2,3,1)
  icetool = substr(iceflby2,4,1)
  select
     when icesort ='1' then op1='Sort/'     /* 1=a sort was specified */
     otherwise           op1=''
  end
  select
     when icemerg ='1' then op2='Merge/'    /* 1=a merge was specified*/
     otherwise           op2=''
  end
  select
     when icecopy ='1' then op3='Copy/'     /* 1=a copy was specified */
     otherwise           op3=''
  end
  select
     when icetool ='1' then op4='Icetool'   /* 1=icetool called sort  */
     otherwise           op4=''
  end
 oper=op1||op2||op3||op4
 iceiotyp = x2b(c2x(substr(x.i,df+37,1)))
                                  /* type of input source/output dest.*/
 iceioe15 = substr(iceiotyp,1,1)           /* e15 used                */
 iceioe32 = substr(iceiotyp,2,1)           /* e32 used                */
 iceioe35 = substr(iceiotyp,3,1)           /* e35 used                */
 iceiosin = substr(iceiotyp,4,1)           /* sortin used             */
 iceioinn = substr(iceiotyp,5,1)           /* sortinxx used           */
```

```
iceioout = substr(iceiotyp,6,1)           /* sortout used             */
iceioofl = substr(iceiotyp,7,1)           /* outfil used              */
 select
    when iceioe15 ='1' then to1='e15 /'
    otherwise               to1=''
 end
 select
    when iceioe32 ='1' then to2='e32 /'
    otherwise               to2=''
 end
 select
    when iceioe35 ='1' then to3='e35 /'
    otherwise               to3=''
 end
 select
    when iceiosin ='1' then to4='sortin /'
    otherwise           to4=''
 end
 select
    when iceioinn ='1' then to5='sortinxx /'
    otherwise           to5=''
 end
 select
    when iceioout ='1' then to6='sortout /'
    otherwise           to6=''
 end
 select
    when iceioofl ='1' then to7='outfil'
    otherwise           to7=''
 end
typo=to1||to2||to3||to4||to5||to6||to7
icecsflg = x2b(c2x(substr(x.i,df+38,1))) /* control statement flags */
icecsalt = substr(icecsflg,1,1)              /* altseq specified   */
icecsinr = substr(icecsflg,2,1)              /* inrec specified    */
icecsinc = substr(icecsflg,3,1)              /* include specified  */
icecsomi = substr(icecsflg,4,1)              /* omit specified     */
icecsorc = substr(icecsflg,5,1)              /* outrec specified   */
icecssum = substr(icecsflg,6,1)              /* sum specified      */
icecsofl = substr(icecsflg,7,1)              /* outfil specified   */
   select
     when icecsalt ='1' then csfØ = 'altseq; '
     otherwise               csfØ =
   end
   select
     when icecsinr ='1' then csf1 = 'inrec; '
     otherwise               csf1 =
   end
   select
     when icecsinc ='1' then csf2 = 'include; '
     otherwise               csf2 =
   end
   select
     when icecsomi ='1' then csf3 = 'omit; '
     otherwise               csf3 =
   end
   select
     when icecsorc ='1' then csf4 = 'outrec; '
     otherwise               csf4 =
   end
   select
     when icecssum ='1' then csf5 = 'sum; '
     otherwise               csf5 =
   end
   select
     when icecsofl ='1' then csf6 = 'outfil'
     otherwise               csf6 =
   end
csf=csfØ||csf1||csf2||csf3||csf4||csf5||csf6 /*control statement used*/
iceres3  = c2d(substr(x.i,df+39,1))              /* reserved         */
icetimes = smf(c2d(substr(x.i,df+4Ø,4))) /*time sort started process.*/
icetime1 = c2d(substr(x.i,df+4Ø,4))
icedates = substr(c2x(substr(x.i,df+44,4)),3,5)
                                        /* date sort started process.*/
icetimee = smf(c2d(substr(x.i,df+48,4)))  /* time sort ended process.*/
icetime2 = c2d(substr(x.i,df+48,4))
icedatee = substr(c2x(substr(x.i,df+52,4)),3,5)
                                        /* date sort ended process.  */
elaps    = cross(icetime2,icetime1)
clock    = (icetime2 - icetime1)*Ø.Ø1
/*-------------------------------------------------------------*/
/* Note: icercbyt field is zero if termination is normal.      */
/*-------------------------------------------------------------*/
icercbyt = x2b(c2x(substr(x.i,df+56,1)))       /* return code status */
icesysab = substr(icercbyt,1,1)               /*system abend detected*/
iceusrab = substr(icercbyt,2,1)                /* user abend detected*/
iceuabnd = substr(icercbyt,5,1)                /*user requested abend*/
icerc16  = substr(icercbyt,6,1)                   /* rc=16 returned  */
icerc2Ø  = substr(icercbyt,7,1)                   /* rc=2Ø returned  */
 select
   when icesysab ='1' then term ='system abend'
   when iceusrab ='1' then term ='user abend'
   when iceuabnd ='1' then term ='user req. abennd'
   when icerc16  ='1' then term ='rc=16'
   when icerc2Ø  ='1' then term ='rc=2Ø'
   otherwise               term ='ok'
 end
icerc    = c2d(substr(x.i,df+57,1)) /*ret.code of sort to its invoker*/
iceresn  = c2d(substr(x.i,df+58,2))      /* reason code              */
iceavlr  = c2d(substr(x.i,df+6Ø,4))   /*avg rec.len. of sorted vb rec*/
icedsa   = c2d(substr(x.i,df+64,2))      /* dsa value in effect      */
```

```
iceflby3 = x2b(c2x(substr(x.i,df+66,1)))   /*       misc flag byte 3 */
icehssrt = substr(iceflby3,1,1)            /*  hipersorting was used */
icedssrt = substr(iceflby3,2,1)            /*    data space was used */
icewds   = substr(iceflby3,3,1)            /*work datasets were used */
icelssm  = substr(iceflby3,4,1)        /*locale sensitive sort/merge */
                                    /*  control field was identified */
icelsio  = substr(iceflby3,5,1)     /* locale sensitive include/omit */
                                    /*  compare field was identified */
icequals = substr(iceflby3,6,1)            /*       equals in effect */
iceblk31 = substr(iceflby3,7,1)            /*      31-bit block size */
 select
   when icehssrt ='1' then flgØ ='hipersorting was used; '
   otherwise               flgØ =''
 end
 select
   when icedssrt ='1' then flg1 ='data space was used; '
   otherwise               flg1 =''
 end
 select
   when icewds   ='1' then flg2 ='work ds were used; '
   otherwise               flg2 =''
 end
 select
   when icelssm  ='1' then flg3 ='locale sensitive sort/merge; '
   otherwise               flg3 =''
 end
 select
   when icelsio  ='1' then flg4 ='locale sensitive include/omit; '
   otherwise               flg4 =''
 end
 select
   when icequals ='1' then flg5 ='compare field was identified; '
   otherwise               flg5 =''
 end
 select
   when iceblk31 ='1' then flg6 ='31-bit block size'
   otherwise               flg6 =''
 end
flg=flgØ||flg1||flg2||flg3||flg4||flg5
icewkflg = x2b(c2x(substr(x.i,df+67,1)))   /*      sortwk flags byte */
icewkdyn = substr(icewkflg,1,1)        /*sortwk tracks dyn allocated */
icewkcw  = substr(icewkflg,2,1)       /*cache fast write used for one*/
                                           /*  or more work datasets */
select
 when icewkdyn = '1' then sortwk = 'dyn.alloc; '
 otherwise                sortwk = ''
end
select
 when icewkcw  = '1' then cache = 'cache fast write used'
 otherwise                cache = ''
```

```
end
swk=sortwk||cache
icewkrf6 = substr(icewkflg,7,1)       /* reserved - do not use       */
icewkrf7 = substr(icewkflg,8,1)       /* reserved - do not use       */
icewexs  = c2d(substr(x.i,df+68,2))
                                  /* initial no. of extents allocated*/
icewexe  = c2d(substr(x.i,df+70,2))  /*final no. of extents allocated*/
icewalls = c2d(substr(x.i,df+72,4))
                               /* initially alloc sortwork ds tracks */
icewalle = c2d(substr(x.i,df+76,4))  /*final alloc sortwork ds tracks*/
iceres5  = c2d(substr(x.i,df+80,3))   /* reserved                    */
/*-----------------------------------------------------------*/
/* Note: if all the if all the bits of iceiamb are zero        */
/*       and sort terminated abnormally, then the type of      */
/*       access method used for SORTIN could not be determined.*/
/*-----------------------------------------------------------*/
iceiamb  = x2b(c2x(substr(x.i,df+83,1)))  /*sortin access method byte*/
iceiexcp = substr(iceiamb,1,1)           /* excp was used for sortin*/
iceivsam = substr(iceiamb,2,1)            /*vsam was used for sortin*/
iceibsam = substr(iceiamb,3,1)            /*bsam was used for sortin*/
  select
   when iceiexcp = '1' then sinam = 'excp'
   when iceivsam = '1' then sinam = 'vsam'
   when iceibsam = '1' then sinam = 'bsam'
   otherwise                sinam = ''
  end
/*----------------------------------------------------------------*/
/* Note: the number of calls to the access method used for a        */
/*       particular dataset (sortin, sortout) will be the total     */
/*       count of excps, or read/writes (bsam), or get/puts (vsam). */
/*----------------------------------------------------------------*/
iceinio  = c2d(substr(x.i,df+84,4))   /*no. of calls to sortin access*/
iceres6  = c2d(substr(x.i,df+88,3))       /* reserved                */
/*------------------------------------------------------------*/
/* Note: if all the bits of iceoamb are zero                    */
/*       and sort terminated abnormally, then the type of       */
/*       access method used for SORTout could not be determined.*/
/*------------------------------------------------------------*/
iceoamb  = x2b(c2x(substr(x.i,df+91,1))) /*sortout access method byte*/
iceoexcp = substr(iceoamb,1,1)           /*excp was used for sortout*/
iceovsam = substr(iceoamb,2,1)           /*vsam was used for sortout*/
iceobsam = substr(iceoamb,3,1)           /*bsam was used for sortout*/
 select
  when iceoexcp = '1' then sotam = 'excp'
  when iceovsam = '1' then sotam = 'vsam'
  when iceobsam = '1' then sotam = 'bsam'
  otherwise                sotam = ''
 end
iceoutio = c2d(substr(x.i,df+92,4))
                              /* no. of calls to sortout access meth */
```

```
iceiblkf = c2d(substr(x.i,df+96,4))   /*max input blocksize or cisize*/
icewkio  = c2d(substr(x.i,df+100,4))  /*no. of excps for all sortwknn*/
icesrbts = c2d(substr(x.i,df+104,4))
                                 /* cumulative srb when sort started */
icesrbte = c2d(substr(x.i,df+108,4))
                                   /* cumulative srb when sort ended */
select
  when icesrbte >0 then srb = (icesrbte - icesrbts)*0.01
  otherwise             srb = 0
end
icetcbs  = c2d(substr(x.i,df+112,4))  /*no. of tcbs defined in region*/
icekeynm = c2d(substr(x.i,df+114,2))
                                  /* no. of sort or merge key fields */
icehspmx = c2d(substr(x.i,df+116,2))   /* hiprmax value in effect    */
icedspsz = c2d(substr(x.i,df+118,2))   /* dspsize value in effect    */
iceexrcs = c2d(substr(x.i,df+120,8))   /* no. of rec. sorted         */
iceexbys = c2d(substr(x.i,df+128,8))   /* no. of bytes sorted        */
iceexinn = c2d(substr(x.i,df+136,8))
                               /* no. of sortin access method calls  */
iceexout = c2d(substr(x.i,df+144,8))
                               /* no. of sortout access method calls */
icehspn  = c2d(substr(x.i,df+152,2))   /* no. of hiperspaces created */
icehspu  = c2d(substr(x.i,df+154,4))   /*no. of hiperspace pages used*/
icedspn  = c2d(substr(x.i,df+158,2))   /* no. of data spaces created */
icedspu  = c2d(substr(x.i,df+160,4))   /*no. of data space pages used*/
iceprcnm =     substr(x.i,df+164,8)    /* procedure step name        */
iceidsnm =     substr(x.i,df+172,44)   /* sortin dataset name        */
iceivols =     substr(x.i,df+216,6)    /* sortin volume serial       */
iceodsnm =     substr(x.i,df+222,44)   /* sortout dataset name       */
iceovols =     substr(x.i,df+266,6)    /* sortout volume serial      */
iceinpds = c2d(substr(x.i,df+272,2))
                                /* no. of sortin datasets (including */
                                       /* concatenated datasets)     */
iceinnds = c2d(substr(x.i,df+274,2))   /* no. of sortinxx datasets   */
iceoutds = c2d(substr(x.i,df+276,2))   /* no. of sortout datasets    */
iceoflds = c2d(substr(x.i,df+278,2))   /* no. of outfil datasets     */
icercinp = c2d(substr(x.i,df+280,8))   /* no. of input rec.          */
icercout = c2d(substr(x.i,df+288,8))   /* no. of output rec.         */
icercins = c2d(substr(x.i,df+296,8))   /* no. of inserted rec.       */
icercdel = c2d(substr(x.i,df+304,8))   /* no. of deleted rec.        */
icemd15n =     substr(x.i,df+312,8)
                               /* routine  specified for mods e15    */
icemd15m = c2d(substr(x.i,df+320,4)) /*storage specified for mods e15*/
icemd15s =     substr(x.i,df+324,8)   /*ddname specified for mods e15*/
icemd15e =     substr(x.i,df+332,1) /*requirements spec. for mods e15*/
iceres10 = c2d(substr(x.i,df+333,3))   /* reserved                   */
icemd35n =     substr(x.i,df+336,8) /* routine specified for mods e35*/
icemd35m = c2d(substr(x.i,df+344,4)) /*storage specified for mods e35*/
icemd35s =     substr(x.i,df+348,8)   /*ddname specified for mods e35*/
icemd35e =     substr(x.i,df+356,1) /*requirements spec. for mods e35*/
```

```
iceres11 = c2d(substr(x.i,df+357,3))   /* reserved                    */
icelcale =     substr(x.i,df+360,32)   /* locale name                 */
iceesmax = c2d(substr(x.i,df+392,4))   /* expmax value in effect      */
iceesold = c2d(substr(x.i,df+396,4))   /* expold value in effect      */
iceesres = c2d(substr(x.i,df+400,4))   /* expres value in effect      */
iceoblkf = c2d(substr(x.i,df+404,4))   /* output blocksize or cisize */
icefilsz = c2d(substr(x.i,df+408,8))
                               /* value specified for filsz/size      */
iceavgrl = c2d(substr(x.i,df+416,4))   /*value specified for avgrlen */
icefszfl = x2b(c2x(substr(x.i,df+420,1)))     /* filsz/size flags     */
icefszn  = substr(icefszfl,1,1)    /*filsz/size value specified as n */
icefszen = substr(icefszfl,2,1)    /*filsz/size value specified as en*/
icefszun = substr(icefszfl,3,1)    /*filsz/size value specified as un*/
icefszno = substr(icefszfl,4,1)    /*filsz/size value not specified &*/
                               /* file size could not be determined  */
select
 when icefszn  = '1' then filsz = 'n'
 when icefszen = '1' then filsz = 'en'
 when icefszun = '1' then filsz = 'un'
 otherwise                filsz = 'not spec'
end
iceres2a = c2d(substr(x.i,df+421,2))    /* reserved                   */
iceres2b = c2d(substr(x.i,df+423,1))    /* reserved - do not use      */
iceres2c = c2d(substr(x.i,df+424,8))    /* reserved - do not use      */
cputm = icecput+srb
select
    when iceexbys <1024 then bysort= iceexbys
    when iceexbys >1024 then bysort = format((iceexbys/1024),8,2)
    otherwise   bysort = iceexbys
end
select
   when iceexbys > 0 then rate = format(((iceexbys/1024)/clock),8,2)
   otherwise              rate = iceexbys
end
select
   when iceexbys > 0 then rac=format(((iceexbys/1024)/cputm),8,2)
   otherwise              rac= iceexbys
end
EXCP  = iceexinn+iceexout+icewkio
io    = format((excp/clock),9,2)
io1   = format((excp/cputm),9,2)
sutot = format((sufa*cputm),9,2) /* CPU - total SU (an approximation)*/
sutcb = format((sufa*icecput),9,2)      /* Tcb cpu - service units   */
susrb = format((sufa*srb),9,2)          /* Srb cpu - service units   */
pfr.1 = right(date('n',icedates,'j'),11) left(icetimes,11)  ,
        left(icejobnm,8)                 left(sortt,10)     ,
        right(icerc,3)                   right(iceresn,3)   ,
        right(elaps,11)                  right(clock,6)     ,
        right(cputm,7)                   right(icecput,7)   ,
        right(srb,5)                     right(icercinp,10) ,
        right(icercds,1Ø)                right(bysort,1Ø)   ,
        right(rate,12)                   right(rac,12)      ,
        right(excp,7)                    right(io,1Ø)       ,
        right(io1,1Ø)                    right(icehspu,6)   ,
        right(icedspu,4)                 right(stor,22)
 "EXECIO * DISKW SORT16 (STEM pfr.)"
     csv.1  = icebdate ||";"|| icebtime ||";"||,
              icesid   ||";"|| icejobnm ||";"||,
              icestn   ||";"|| icestpnm ||";"||,
              term     ||";"|| oper     ||";"||,
              elaps    ||";"|| cputm    ||";"|| sutot ||";"||,
              iceexrcs ||";"|| iceavlr  ||";"||,
              icekeyln ||";"|| iceexbys ||";"||,
              icendyna ||";"|| icewblk  ||";"||,
              iceinio  ||";"|| iceoutio ||";"||,
              icewkio  ||";"|| icehspu  ||";"||,
              icedspu
 "EXECIO * DISKW SORTCS (STEM csv.)"
/*-----------------------------------------------------------------------*/
/* Record len. distribution section                                      */
/*-----------------------------------------------------------------------*/
if (icestat > Ø) Then do
 vbo  = icestat - 3
icectr   = c2d(substr(x.i,vbo,4))  /*record counters for int.s 1 - 16*/
icectrØ1 = c2d(substr(x.i,vbo+4 ,4)) /* rec. in int.  1 len.s   5-15 */
icectrØ2 = c2d(substr(x.i,vbo+8 ,4)) /* rec. in int.  2 len.s  16-31 */
icectrØ3 = c2d(substr(x.i,vbo+12,4)) /* rec. in int.  3 len.s  32-63 */
icectrØ4 = c2d(substr(x.i,vbo+16,4)) /* rec. in int.  4 len.s 64-127 */
icectrØ5 = c2d(substr(x.i,vbo+2Ø,4)) /* rec. in int.  5 len.s 128-191*/
icectrØ6 = c2d(substr(x.i,vbo+24,4)) /* rec. in int.  6 len.s 192-255*/
icectrØ7 = c2d(substr(x.i,vbo+28,4)) /* rec. in int.  7 len.s 256-511*/
icectrØ8 = c2d(substr(x.i,vbo+32,4)) /*rec. in int.  8 len.s 512-1Ø23*/
icectrØ9 = c2d(substr(x.i,vbo+36,4)) /*rec. in int.  9 len.s 1Ø24-2Ø47*/
icectr1Ø = c2d(substr(x.i,vbo+4Ø,4)) /*rec. in int. 1Ø len.s 2Ø48-4Ø95*/
icectr11 = c2d(substr(x.i,vbo+44,4)) /*rec. in int. 11 len.s 4Ø96-7167*/
icectr12 = c2d(substr(x.i,vbo+48,4))
                                   /*rec. in int. 12 len.s 7168-1Ø751*/
icectr13 = c2d(substr(x.i,vbo+52,4))
                                  /*rec. in int. 13 len.s 1Ø752-15359*/
icectr14 = c2d(substr(x.i,vbo+56,4))
                                  /*rec. in int. 14 len.s 1536Ø-2Ø991*/
icectr15 = c2d(substr(x.i,vbo+6Ø,4))
                                  /*rec. in int. 15 len.s 2Ø992-26623*/
icectr16 =     substr(x.i,vbo+64,4)
                                  /*rec. in int. 16 len.s 26624-32767*/
 select
   when icectr16  ='4Ø4Ø4Ø4Ø'X  then icectr16 = Ø
   otherwise  icectr16 = c2d(substr(x.i,vbo+64,4))
 end
vbr.1 = right(date('n',icedates,'j'),11) left(icetimes,11)   ,
        left(icejobnm,8)                  right(icercinp,1Ø) ,
        right(icectr,6)                   right(icectrØ1,6)  ,
        right(icectrØ2,6)                 right(icectrØ3,6)  ,
        right(icectrØ4,6)                 right(icectrØ5,6)  ,
        right(icectrØ6,6)                 right(icectrØ7,6)  ,
        right(icectrØ8,6)                 right(icectrØ9,6)  ,
        right(icectr1Ø,6)                 right(icectr11,6)  ,
        right(icectr12,6)                 right(icectr13,6)  ,
        right(icectr14,6)                 right(icectr15,6)  ,
        right(icectr16,6)
 "EXECIO * DISKW SORTFV (STEM vbr.)"
end
/*-------------------------------------------------------------*/
/* Input dataset section                                       */
/*  - one section for each of up to 16 sortin datasets         */
/*    (including concatenations) or                            */
/*  - one section for each of up to 16 sortinxx datasets       */
/*-------------------------------------------------------------*/
/*-------------------------------------------------------------*/
/* Report header for DFSORT file report                        */
/*-------------------------------------------------------------*/
hin.1 = left('Date',11)          left('started',11)       ,
        left('Jobname',8)        left('#files:',7)        ,
        left('Sortin:',7)        left('Sortout:',8)       ,
        left('Outfile:',8)
hin.2 = left(date('n',icedates,'j'),11) left(icetimes,11) ,
        left(icejobnm,18)               left(iceinpds,8)  ,
        left(iceoutds,8)                left(iceoflds,2)
hin.3 = left('-',75,'-')
hin.4 = left('No. of input rec.   :',22) right(icercinp,7)
hin.5 = left('No. of output rec.  :',22) right(icercout,7)
hin.6 = left('No. of inserted rec.:',23) right(icercins,6)
hin.7 = left('No. of deleted rec. :',22) right(icercdel,7)
hin.8 = left('-',75,'-')
hin.9 = left(' ',9,' ')                 left('ddname',8)       ,
        left('data set name',2Ø)        left('volser',9)       ,
        left('recfm',5)                 right('lrecl',8)       ,
        right('blksize',9)              right('bytes read',11) ,
        right('EXCP',6)
hin.1Ø = left(' ',9,' ')  left('-',83,'-')
   if (iceinds > Ø) Then do
 "EXECIO * DISKW SORTFI (STEM hin.)"
    do pp = Ø to (iceindsn -1)
    ino  = (iceinds  + (pp*iceindsl))- 3
/* ------------------------------------------------------------- */
/* Note: the values for iceinfl1, iceinfl2, iceinamb,iceintyp,      */
/* iceinrcf, iceinbyt, iceinamc, iceinlrl, iceinbkz, iceinddn,      */
/* iceinnam, iceinvol, iceinbkf vars. are only provided when a      */
/* blockset sort, copy, or merge application is successful and     */
/* smf=full is specified.                                           */
/* ------------------------------------------------------------- */
iceinfl1 = x2b(c2x(substr(x.i,ino,1)))      /* flags byte 1          */
iceinsin = substr(iceinfl1,1,1)             /* sortin dataset        */
iceinsnn = substr(iceinfl1,2,1)             /* sortinxx dataset      */
iceinfl2 = x2b(c2x(substr(x.i,ino+1,1)))    /* flags byte 2          */
iceinpip = substr(iceinfl2,1,1)             /* batchpipes/mvs        */
iceinstr = substr(iceinfl2,2,1)             /* striped               */
iceincpr = substr(iceinfl2,3,1)             /* compressed            */
iceinea  = substr(iceinfl2,4,1)            /* extended addressability*/
iceinhfs = substr(iceinfl2,5,1)             /* hfs file              */
iceinamb = x2b(c2x(substr(x.i,ino+2,1)))   /*access method flags byte*/
iceinexp = substr(iceinamb,1,1)             /* excp used             */
iceinvsm = substr(iceinamb,2,1)             /* vsam used             */
iceinbsm = substr(iceinamb,3,1)             /* bsam used             */
iceintyp = x2b(c2x(substr(x.i,ino+3,1)))    /* type flags byte       */
iceintap = substr(iceintyp,1,1)             /* tape dataset          */
iceindas = substr(iceintyp,2,1)             /* dasd dataset          */
iceinspd = substr(iceintyp,3,1)       /*spool, dummy or batchpipes/  */
                                        /* mvs dataset or hfs file   */
iceinrcf = c2d(substr(x.i,ino+4,1,))    /* recfm (flags are identical*/
                                            /* to those in jfcrecfm) */
iceinrs1 = c2d(substr(x.i,ino+5,3))         /* reserved              */
iceinbyt = c2d(substr(x.i,ino+8,8))         /* no. of bytes read     */
iceinamc = c2d(substr(x.i,ino+16,8))  /*no. of calls to access method*/
iceinrs2 = c2d(substr(x.i,ino+24,2))        /* reserved              */
iceinlrl = c2d(substr(x.i,ino+26,2))        /* lrecl                 */
iceinrs3 = c2d(substr(x.i,ino+28,2))        /* reserved              */
iceinbkz = c2d(substr(x.i,ino+3Ø,2))        /* blksize or cisize     */
iceinddn =     substr(x.i,ino+32,8)         /* ddname                */
iceinnam =     substr(x.i,ino+4Ø,44)        /* dataset name          */
iceinvol =     substr(x.i,ino+84,6)         /* first volume serial   */
iceinrs4 = c2d(substr(x.i,ino+9Ø,2))        /* reserved              */
iceinbkf = c2d(substr(x.i,ino+92,4))    /*blksize or cisize (31-bit) */
Select
 when pp = Ø then    kk.1  = left('SORTIN:',9)
 otherwise           kk.1  = left(' ',9,' ')
end
inr.1 = kk.1||left(iceinddn,8)                     ,
        left(iceinnam,2Ø)    left(iceinvol,9)   ,
        left(reclen,6)       right(iceinlrl,7)  ,
        right(iceinbkf,9)    right(iceinbyt,12) ,
        right(iceinamc,5)
 "EXECIO * DISKW SORTFI (STEM inr.)"
 end
   if (iceoutds = Ø) &  (iceoflds = Ø) then do
     inb.1 = left(' ',1)
     "EXECIO * DISKW SORTFI (STEM inb.)"
  end
end
/*-------------------------------------------------------------*/
```

```
/* Sortout dataset section                                          */
/*-----------------------------------------------------------------*/
   iF (iceotdsn > Ø) Then do
    oso  = iceotds - 3
iceotfl1 = x2b(c2x(substr(x.i,oso,1)))   /* flags byte 1            */
iceotpip = substr(iceotfl1,1,1)          /*     batchpipes/mvs      */
iceotstr = substr(iceotfl1,2,1)          /*     striped             */
iceotcpr = substr(iceotfl1,3,1)          /*     compressed          */
iceotea  = substr(iceotfl1,4,1)          /*     extended addres.    */
iceothfs = substr(iceotfl1,5,1)          /*     hfs file            */
iceotamb = x2b(c2x(substr(x.i,oso+1,1))) /* access method flags byte */
iceotexp = substr(iceotamb,1,1)          /*     excp used           */
iceotvsm = substr(iceotamb,2,1)          /*     vsam used           */
iceotbsm = substr(iceotamb,3,1)          /*     bsam used           */
iceottyp = x2b(c2x(substr(x.i,oso+2,1))) /* type flags byte         */
iceottap = substr(iceottyp,1,1)          /*     tape dataset        */
iceotdas = substr(iceottyp,2,1)          /*     dasd dataset        */
iceotspd = substr(iceottyp,3,1)        /*spool, dummy or batchpipes/ */
                                         /* mvs dataset or hfs file  */
iceotrcf = c2d(substr(x.i,oso+3,1))      /* recfm                   */
iceotrs1 = c2d(substr(x.i,oso+4,4))      /* reserved                */
iceotbyt = c2d(substr(x.i,oso+8,8))      /* no. of bytes written    */
iceotrec = c2d(substr(x.i,oso+16,8))     /* no. of rec. written     */
iceotamc = c2d(substr(x.i,oso+24,8))   /*no. of calls to access method*/
iceotrs2 = c2d(substr(x.i,oso+32,2))     /* reserved                */
iceotlrl = c2d(substr(x.i,oso+34,2))     /* lrecl                   */
iceotrs3 = c2d(substr(x.i,oso+36,2))     /* reserved                */
iceotbkz = c2d(substr(x.i,oso+38,2))     /* blksize or cisize       */
iceotddn =     substr(x.i,oso+4Ø,8)      /* ddname                  */
iceotnam =     substr(x.i,oso+48,44)     /* dataset name            */
iceotvol =     substr(x.i,oso+92,6)      /* first volume serial     */
iceotrs4 = c2d(substr(x.i,oso+98,2))     /* reserved                */
iceotbkf = c2d(substr(x.i,oso+1ØØ,4))    /*blksize or cisize (31-bit)*/
onr.1 = left('SORTOUT:',8)   left(iceotddn,8)  ,
        left(iceotnam,2Ø)    left(iceotvol,9)  ,
        left(reclen,6)       right(iceotlrl,7) ,
        right(iceotbkf,9)    right(iceotbyt,12),
        right(iceotamc,5)
 "EXECIO * DISKW SORTFI (STEM onr.)"
   inb.1 = left(' ',1)
 "EXECIO * DISKW SORTFI (STEM inb.)"
end
/*-----------------------------------------------------------------*/
/* Outfil dataset section                                           */
/*  - one section for each of up to 16 outfil datasets              */
/*-----------------------------------------------------------------*/
   if (iceofdsn > Ø) Then do
    do ff = Ø to (iceofdsn -1)
    ofo  = (iceofds  + (ff*iceofdsl))- 3
iceoffl1 = x2b(c2x(substr(x.i,ofo,1)))   /* flags byte 1            */
```

```
iceofpip = substr(iceoffl1,1,1)          /* batchpipes/mvs            */
iceofstr = substr(iceoffl1,2,1)          /* striped                   */
iceofcpr = substr(iceoffl1,3,1)          /* compressed                */
iceofea  = substr(iceoffl1,4,1)          /* extended addressability  */
iceofhfs = substr(iceoffl1,5,1)          /* hfs file                  */
  select
    when iceofpip = '1' then fil1 = 'batchpipes/mvs'
    otherwise                fil1 = 'mvs/ '
  end
  select
    when iceofstr = '1' then fil2 = 'striped'
    otherwise                fil2 = 'non striped/ '
  end
  select
    when iceofstr = '1' then fil3 = 'compressed'
    otherwise                fil3 = 'non compessed/ '
  end
  select
    when iceofea = '1' then fil4 = 'ext. addr'
    otherwise               fil4 = ''
  end
  select
    when iceofhfs = '1' then fil5 = 'hfs'
    otherwise                fil5 = ''
  end
fil=fil1||fil2||fil3||fil4||fil5
iceofamb = x2b(c2x(substr(x.i,ofo+1,1))) /* access method flags byte */
iceofexp = substr(iceofamb,1,1)          /* excp used                 */
iceofvsm = substr(iceofamb,2,1)          /* vsam used                 */
iceofbsm = substr(iceofamb,3,1)          /* bsam used                 */
 select
   when iceofexp = '1'  then amb = 'excp used'
   when iceofvsm = '1'  then amb = 'vsam used'
   when iceofbsm = '1'  then amb = 'bsam used'
   otherwise                 amb = ' '
 end
iceoftyp = x2b(c2x(substr(x.i,ofo+2,1))) /* type flags byte           */
iceoftap = substr(iceoftyp,1,1)          /* tape dataset              */
iceofdas = substr(iceoftyp,2,1)          /* dasd dataset              */
iceofspd = substr(iceoftyp,3,1)          /* spool, dummy or           */
                                         /* batchpipes/mvs dataset    */
                                         /* or hfs file               */
 select
  when iceoftap = '1' then ftype = 'tape dataset'
  when iceofdas = '1' then ftype = 'dasd dataset'
  when iceofspd = '1' then ftype = 'spool/dummy/hfs'
  othewise                 ftype = ' '
 end
iceofrcf = c2d(substr(x.i,ofo+3  ,1))    /* recfm                     */
iceofprm = x2b(c2x(substr(x.i,ofo+4,1))) /* outfil parm flags byte    */
```

```
iceofser = substr(iceofprm,1,1)           /* outfil startrec or endrec*/
                                          /* parm specified           */
iceofios = substr(iceofprm,2,1)         /*outfil include, omit or save*/
                                          /* parm spec.               */
iceofspl = substr(iceofprm,3,1)           /* outfil split parm spec.  */
iceoforc = substr(iceofprm,4,1)           /* outfil outrec parm spec. */
iceofcvt = substr(iceofprm,5,1)        /*outfil convert, vtof or ftov */
                                          /* parm spec.               */
iceofrep = substr(iceofprm,6,1)           /* outfil report parm spec. */
iceofvfl = substr(iceofprm,7,1)           /* outfil vlfill parm spec. */
iceofvtr = substr(iceofprm,8,1)           /* outfil vltrim parm spec. */
   select
      when iceofser = '1' then o1 = 'startrec/endrec; '
      otherwise                o1 = ''
   end
   select
      when iceofios = '1' then o2 = 'include, omit or save; '
      otherwise                o2 = ''
   end
   select
      when iceofspl = '1' then o3 = 'split; '
      otherwise                o3 = ''
   end
   select
      when iceoforc = '1' then o4 = 'outrec; '
      otherwise                o4 = ''
   end
   select
      when iceofcvt = '1' then o5 = 'convert, vtof or ftov; '
      otherwise                o5 = ''
   end
   select
      when iceofrep = '1' then o6 = 'report;'
      otherwise                o6 = ''
   end
   select
      when iceofvfl = '1' then o7 = 'vlfill; '
      otherwise                o7 = ''
   end
   select
      when iceofvtr = '1' then o8 = 'vltrim'
      otherwise                o8 = ''
   end
iceofpr2 = x2b(c2x(substr(x.i,ofo+5,1))) /* outfil parm flags byte   */
iceofrmc = substr(iceofpr2,1,1)          /* outfil removecc parm spec.*/
   select
      when iceofrmc = '1' then o9 = 'removecc'
      otherwise                o9 = ''
   end
 outfop=o1||o2||o3||o4||o5||o6||o7||o8||o9
```

```
iceofrs1 = c2d(substr(x.i,ofo+6,2))      /* reserved                 */
iceofbyt = c2d(substr(x.i,ofo+8,8))      /* no. of bytes written     */
iceofrec = c2d(substr(x.i,ofo+16,8))     /* no. of rec. written      */
iceofamc = c2d(substr(x.i,ofo+24,8))  /*no. of calls to access method*/
iceofrs2 = c2d(substr(x.i,ofo+32,2))     /* reserved                 */
iceoflrl = c2d(substr(x.i,ofo+34,2))     /* lrecl                    */
iceofrs3 = c2d(substr(x.i,ofo+36,2))     /* reserved                 */
iceofbkz = c2d(substr(x.i,ofo+38,2))     /* blksize or cisize        */
iceofddn =     substr(x.i,ofo+4Ø,8)      /* ddname                   */
iceofnam =     substr(x.i,ofo+48,44)     /* dataset name             */
iceofvol =     substr(x.i,ofo+92,6)      /* first volume serial      */
iceofrs4 = c2d(substr(x.i,ofo+98,2))     /* reserved                 */
iceofbkf = c2d(substr(x.i,ofo+1ØØ,4))    /*blksize or cisize (31-bit)*/
Select
 when ff = Ø then    tt.1  = left('OUTFILE:',9)
 otherwise           tt.1  = left(' ',9,' ')
end
fnr.1 = tt.1||left(iceofddn,8)  ,
        left(iceofnam,2Ø)     left(iceofvol,9)  ,
        left(reclen,6)        right(iceoflrl,7) ,
        right(iceofbkf,9)     right(iceofbyt,12),
        right(iceofamc,5)     right(iceofrec,8)
 "EXECIO * DISKW SORTFI (STEM fnr.)"
        end
inb.1 = left(' ',1)
 "EXECIO * DISKW SORTFI (STEM inb.)"
      end
   end
 end
ft.1 = left(' ',1)
ft.2 = left('TERMS AND ABBREVIATIONS USED:',9Ø)
ft.3 = left('Date         : Date sort started processing',9Ø)
ft.4 = left('started      : Time sort started processing',9Ø)
ft.5 = left('Technique    : Sort technique used',9Ø)
ft.6 = left('rc           : Return code',9Ø)
ft.7 = left('rs           : Reason code',9Ø)
ft.8 = left('Elapsed      : Sort elapsed time in hh:mm:ss:tt',9Ø)
ft.9 = left('Clock        : Clock time (in sec)',9Ø)
ft.1Ø= left('CPUtm        : CPU time (tcb + srb) in sec',9Ø)
ft.11= left('Tcb          : CPU tcb time (sec)',9Ø)
ft.12= left('Srb          : CPU srb time (sec)',9Ø)
ft.13= left('In rec       : Number of input records',9Ø)
ft.14= left('Sort rec     : Number of records sorted',9Ø)
ft.15= left('KB sorted    : Number of bytes sorted in KB',9Ø)
ft.16= left('Sort rate 1: Sort processing rate KB/Clock second',9Ø)
ft.17= left('Sort rate 2: Sort processing rate KB/CPU second',9Ø)
ft.18= left('EXCP         : The number of calls to the access method',9Ø)
ft.19= left('               used for a particular dataset (sortin,
sortout)',9Ø)
ft.2Ø= left('               will be the total count of excps, or read/
writes',9Ø)
```

```
ft.21= left('             (bsam), or get/puts (vsam)',90)
ft.22= left('Io rate 1  : EXCP/Clock  (in KB/sec)',90)
ft.23= left('Io rate 2  : EXCP/CPUtm  (in KB/cpu.sec)',90)
ft.24= left('HSPu       : Number of hiperspace pages used',90)
ft.25= left('DSPu       : Number of data space pages used',90)
 "EXECIO * DISKW SORT16 (STEM ft.)"
 /* Close & free all allocated files  */
  "EXECIO 0 DISKW SORT16(FINIS "
  "EXECIO 0 DISKW SORTFI(FINIS "
  "EXECIO 0 DISKW SORTFV(FINIS "
  "EXECIO 0 DISKW SORTCS(FINIS "
say
say 'DFSORT performance report file  ...:'sorr
say 'DFSORT file report dataset       ..:'sorf
say 'DFSORT VB file report dataset   ...:'sorv
say 'DFSORT CSV file summary report  ...:'csvr
say
   "FREE FILE(SMF SORT16 SORTFI SORTFV SORTCS)"
exit

CROSS: procedure
/* ------------------------------------------------------------ */
/*        Cover the midnight crossover                          */
/* ------------------------------------------------------------ */
arg endtime,startime
  select
   when endtime > startime then nop
   otherwise  endtime = endtime + 8640000
  end
diftm = smf(endtime - startime)
return diftm

SMF: procedure
/* REXX - convert a SMF time to hh:mm:ss:hd format   */
arg time
    time1 = time % 100
    hh    = time1 % 3600
    hh    = RIGHT("0"||hh,2)
    mm    = (time1 % 60) - (hh * 60)
    mm    = RIGHT("0"||mm,2)
    ss    = time1 - (hh * 3600) - (mm * 60)
    ss    = RIGHT("0"||ss,2)
    fr    = time //  1000
    fr    = RIGHT("0"||fr,2)
    rtime = hh||":"||mm||":"||ss||":"||fr
    return rtime
```

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

Mile Pekic
Systems Programmer (Serbia and Montenegro)

MVS news

IBM has announced WebSphere Application Server (WAS) Version 6 for IBM eServer zSeries z/OS. It includes new features such as a common code base, mainframe processor optimization, and enhanced software configuration management, which are designed to help speed up application development and deployment. IBM suggests that WAS 6 can cut mainframe application development by up to 25 percent.

The latest release is optimized to take advantage of the zSeries Application Assist Processor (zAAP).

IBM also announced the general availability of CICS Transaction Server for z/OS Version 3.1, WebSphere Studio Asset Analyzer Version 4.1, Asset Transformation Workbench Version 1.1, and introduced enhancements to its Software Configuration Management (SCM) solutions, including Rational ClearCase and Rational ClearQuest.

For further information contact your local IBM representative:
URL: www-1.ibm.com/servers/eserver/zseries/zos/zos_sods.html.

* * *

PKWARE has announced the latest version of SecureZIP, its enterprise-wide data security platform for zSeries systems. SecureZIP enables organizations to extend data security and compression capabilities across all major computing platforms (from desktops to mainframes) for secure storage and transfer of sensitive data such as financial records, health information, and other private consumer and business-related data.

The new releases include digital signature capabilities for protecting data integrity and for authenticating the originator of the information.

Data confidentiality is supported through the use of certificate-based or password-based encryption, making it possible to securely transmit files to recipients with diverse security capabilities.

For further information contact:
URL: www.pkware.com/news/releases/2005/040505.php.

* * *

DataMirror has announced Version 4.7 of Transformation Server for z/OS, its real-time, bi-directional mainframe data integration software.

The product allows organizations to synchronize and integrate mainframe data with information systems and business applications running on any platform. It captures changes to data in the databases of a mainframe and other systems as they happen, flowing the data in real-time to other platforms or data stores.

For further information contact:
URL: www.datamirror.com/investors/2005/apr11-05.aspx.

* * *

Phoenix Software has announced Version 7.2 of Key/101, its software for integrating data input via IBM 370 architecture mainframe, PCs, data capture devices, and communication networks.

The Key/101 system is now 31-bit, and has been upgraded to support RMODE ANY execution.

For further information contact:
URL: www.phoenixsoftware.com/Key101.htm.

xephon