# 226

# MVS update

*July 2005*

## In this issue





A xephon PUBLICATION

## 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.



*Printed in England.*

## FLEX-ES mainframe device emulation

In a traditional mainframe environment, devices such as disk controllers, print controllers, communications controllers, and tape controllers are normally channel-attached special-purpose devices. In recent years, IBM, along with several other vendors, has begun to offer 'virtualization' of these devices. These often perform services identical to those of the real physical devices, but offer better performance. In addition, increased flexibility and operation make such devices desirable.

Fundamental Software offers a new type of virtual device controller that aims to simplify the data centre environment further. Using a Linux-based xSeries server, attached via an ESCON channel, the new system can act as an individual or multiple control units. This means you can remove many physical devices and replace them with one virtual server.

With FLEX-ES Control Unit Behavior (FLEXCUB), as the virtual device controller is called, the standard mainframe can access emulated mainframe devices via a channel. The xSeries server hosts the FLEXCUB and the server itself is attached to the mainframe via an ESCON channel using a Fundamental Software Serial Communications Adapter (SCA). Each FLEXCUB on the server allows emulated devices to be addressed by the mainframe system as if they were traditional mainframe devices. The xSeries server can be attached to more than one mainframe, FLEXCUB can then attach to DASD devices, virtual tape devices, physical tape devices, etc.

Figure 1 details the devices that FLEXCUB will represent. Even though some of these devices were never created to use ESCON channels, this is now allowed using FLEXCUB.

Note that an FSI Serial Channel Adapter (SCA-1) can be seen as one or more control unit interfaces on a mainframe channel. Up to 16 individual control units can be supported on each adapter. The SCA-1 also provides parity checking at all points where data movement occurs.

| *Device type* | *Models* |
|---|---|
| CKD DASD | 3390-1/2/3/9 |
| CKD DASD | 3380-A/D/E/J/K |
| CKD DASD | 3375, 3350, 3340-35/70, 3330-1/11 |
| CKD DASD | 9345-1/2 |
| CKD DASD | Custom size support. |
| FBA DASD | 9336-10/20, 9335, 9332-200/400 |
| FBA DASD | 3370-1/2, 3310 |
| FBA DASD | Custom size support. |
| Tape devices | 3480, 3490, 3490-E, 3590, 3420-4/6/8, 3422, 3423, 3430<br>All tapes will work as emulated devices using files on DASD. A number of formats can be used. |
| Consoles | Locally attached non-SNA 3270 via TN3270 clients. |
| Communications | 3172/XCA Ethernet or Token Ring. |
| Printers | 1403, 1403-N1, 3203-1/2/4/5, 3211, 3262-1/11, 4245-1/12/20, 4248-1/2 |
| Card devices | 2501, 2540 |

*Figure 1: Devices available using FLEXCUB*

FLEXCUB runs on Linux, and control unit recoverability and security are provided by a read-only version of Linux that is bootable directly from CD-ROM. This also protects against viruses and deliberate corruption. User customization data is stored on a diskette or in flash memory, or can be on both if required. Because Linux is used, the control unit functions can easily be linked to the open systems elements. Also, because of the configuration, it is possible to back up data on the devices attached to the controller using CPU cycles of the

controller rather than the actual mainframe – thus providing performance benefits.

IBM xSeries servers can be attached to high-speed disks that utilize RAID. Mainframe DASD are created on these volumes and the RAID configuration ensures that data is striped and protected against hardware failure. A failed disk can be replaced and the data recovered without any loss of data or service. Because disks are treated as open systems disks, a number of possibilities can be used for back-up and recovery. In addition, because of the way the disks are emulated, excellent I/O performance is possible. Because open systems software can be used for back-up, it is possible to save money by using open systems software to replace expensive mainframe back-up solutions. It is possible to use existing mainframe software if that is what is preferred.

In addition to all the above, the footprint in the data centre can be much smaller than if using traditional devices.

The benefits provided by FLEXCUB DASD are summarized below:

- Ability to emulate mainframe DASD on inexpensive SCSI RAID HDDs.
- Configurable control unit and DASD device caches.
- Ability to share devices between multiple S/390 and zSeries operating systems.
- Options to back up data to open systems tape devices.
- Exploitation of open systems capabilities.
- Addition of DASD by a simple change to configuration files.
- Ability to create volumes using custom sizes.

Another major area of the FLEXCUB is virtual tape. Virtual tape exists on DASD units and provides major speed benefits over real tape. This means that by eliminating real tape usage,

installations can reduce processing delays significantly. FLEXCUB provides benefits over other manufacturers' emulation because it does not just emulate 3480 and 3490 devices. You can see from Figure 1 that it provides emulation for many types of tapes. It also offers automated compression and decompression, thus reducing the tape volume on disk. This compression as well can be used on devices where the real versions have no compression capabilities – for example the 3422 tape media.

As well as the actual emulation, there is also a virtual tape library management package that will automatically mount virtual tape files for z/OS batch jobs or TSO users without any changes to system JCL. Tape libraries can be shared between systems. In addition a number of utilities and operator commands are provided to list and manage library inventories and to allow volumes to be entered, ejected, scratched, or erased.

It is also possible to use SCSI DLT tapes as 3490 devices. The SCSI tape can be mounted directly on an emulated tape device instead of as a file on disk. The tape can then be written to as if it were a 3490 tape. This means that you can use higher capacity and newer technology media without the need for new mainframe device support.

The benefits provided by FLEXCUB TAPE are summarized below:

- Improved speed in tape jobs.
- Fewer tape mounts needed because of higher capacity.
- Removal of rewind times and back hitching.
- Emulated tapes occupy less physical space than magnetic tapes.
- Tapes can be copied to CD or DVD.
- Exploitation of DLT technology.

- Adding tapes is simple because only a change to a configuration file is required.
- Mechanical failures will not occur.

In addition to DASD and tape benefits, obviously other devices can be emulated. One area particularly that is problematic is the MVS console systems. How many installations still have non-SNA 3270 control unit devices and terminals attached to them as consoles? In recent years the requirement for these devices has reduced, but it is still not eliminated. FLEXCUB allows local non-SNA 3274 emulation to be achieved with local 3270 devices appearing to be connected to the mainframe. You can connect using a TN3270 PC client, which connects via the network to the emulated device. This device can then be used to IPL the mainframe or to act as a terminal that can log on to the system in the event of a broken IP stack or SNA issues. This provides a level of recovery that was previously not possible. It can also be used as a systems programming productivity aid by allowing remote access to test LPARs, etc. The need for channel-attached 3274s and 3174s is removed when this aspect of FLEXCUB is implemented.

Finally, it is also possible to remove older channel-attached printers and to route output to newer TCP/IP print devices. This can be achieved without modifying mainframe JCL or programs. Data is automatically translated to ASCII from EBCDIC and can be directed to a file for viewing, printing, or archiving. This also removes the need to ensure network printers are stacked with paper and can circumvent many operational issues.

Overall the FLEXCUB product is ideally positioned to reduce equipment costs, floor space, environmental costs, and general support costs for maintenance and repair of equipment, and offers many enhancements over ‘real’ devices.

*John Bradley*
*Systems Programmer*
*Meerkat Computer Services (UK)*

# Get the contoken value present in the DBRM

The purpose of the tool described below is to get the contoken value present in the DBRM (DataBase Request Module).

## BACKGROUND

When a COBOL DB2 program (embedded SQL in COBOL) is compiled, the pre-compiler strips off all SQL from the COBOL DB2 source module and places it into a module called DBRM (DataBase Request Module). The stripped SQL is replaced by equivalent COBOL call statements.

DBRM is then used in the BIND process to create packages/ plans. In order for the DBRM and the load module to be in sync, the pre-compiler generates a timestamp and puts it into the DBRM and the load module. This is known as a contoken in the DB2 world. At the time of the BIND process, the contoken value is also stored in the DB2 system catalog. When the program executes any SQL coded in its source code, DB2 checks the load module timestamp with the timestamp value present in the package created by the associated DBRM. If they are different, the program abends with -818 (timestamp mismatch). If the DBRM is not bound after it is compiled, the program will abend with -805 (package/plan not found). In either case, it is essential to see whether the timestamp values in the DBRM and the load module match.

The timestamp value is 16 characters long and it is stored in the DBRM in 8 bytes in the packed form. So it takes some effort to grab these packed bytes and arrange them in the sequence of the timestamp value.

With contoken, getting this timestamp is just one command away.

The contoken value is stored in the DBRM in the first line and after the DBRM name for about 8 bytes.

## SOURCE CODE

Use the following simple REXX program to get the contoken value from a DBRM.

It's very important to store this source in a PDS that is concatenated to your profile (use TSO ISRDDN to see your current allocation). Otherwise, add the PDS to the concatenation of the SYSEXEC/SYSPROC DD statement.

YOUR.CLIST.REXXLIB should be allocated to your TSO session to execute this program like a macro.

```
    YOUR.CLIST.REXXLIB(GETTOKEN) - Ø1.Ø1            Columns ØØØØ1 ØØØ72
 ===>                                                 Scroll ===> CSR
****************************** Top of Data ******************************
/*rexx*/
"isredit macro "
"isredit (linetxt) = line 1"
hexval = c2x(substr(linetxt,25,8))
zedsmsg = 'X'||"'"||hexval||"'"
zedlmsg = 'X'||"'"||substr(hexval,9,8)||substr(hexval,1,8)||"'"
"ispexec setmsg msg(ISRZØØ1)"
**************************** Bottom of Data ****************************
```

## HOW TO EXECUTE

Open a valid DBRM member in edit or view mode:

```
VIEW     SOME.TESTPROD.DBRMLIB(SOMEPGM$) - Ø1.ØØ     Columns ØØØØ1 ØØØ72
Command ===>                                            Scroll ===> CSR
****** ************************** Top of Data ***************************
ØØØØØ1 DBRM  µYOURID4 SOMEPGM$ ±¢J  ^_  K
ØØØØØ2
ØØØØØ3 DBRM  Ç  ¬  <DECLARE X_PLT_TYPES TABLE ( TAGS_ID CHAR ( 2 ) NOT
NULL ,
ØØØØØ4 , DEAL_PARTY_ID CHAR ( 16 ) NOT NULL , DEAL_NAME CHAR ( 3Ø ) NOT
NULL ,
ØØØØØ5 UPDT_DT DECIMAL ( 9 , Ø ) NOT NULL , BRTH_STATE CHAR ( 1 ) NOT
NULL , CU
```

At the command prompt, enter the command GETTOKEN as shown below. (If your member name in the PDS YOUR.CLIST.REXXLIB above is different, the command is your member name):

```
VIEW     SOME.TESTPROD.DBRMLIB(SOMEPGM$) - Ø1.ØØ     Columns ØØØØ1 ØØØ72
```

```
Command ===> GETTOKEN                                    Scroll ===> CSR
****** ************************** Top of Data ***************************
000001 DBRM  µYOURID4 SOMEPGM$ ±¢J  ^_  K
000002
000003 DBRM  Ç  ¬  <DECLARE X_PLT_TYPES TABLE ( TAGS_ID CHAR ( 2 ) NOT
NULL ,
000004 , DEAL_PARTY_ID CHAR ( 16 ) NOT NULL , DEAL_NAME CHAR ( 30 ) NOT
NULL ,
000005 UPDT_DT DECIMAL ( 9 , 0 ) NOT NULL , BRTH_STATE CHAR ( 1 ) NOT
NULL , CU
```

After typing in the command GETTOKEN and pressing *Enter*, you will get a short message at the top right-hand corner as shown below:

```
   VIEW     SOME.TESTPROD.DBRMLIB(SOMEPGM$) - 01.00  X'173CAF180EA757F8'
Command ===>                                             Scroll ===> CSR
****** ************************** Top of Data ***************************
000001 DBRM  µYOURID4 SOMEPGM$ ±¢J  ^_  K
000002
000003 DBRM  Ç  ¬  <DECLARE X_PLT_TYPES TABLE ( TAGS_ID CHAR ( 2 ) NOT
NULL ,
000004 , DEAL_PARTY_ID CHAR ( 16 ) NOT NULL , DEAL_NAME CHAR ( 30 ) NOT
NULL ,
000005 UPDT_DT DECIMAL ( 9 , 0 ) NOT NULL , BRTH_STATE CHAR ( 1 ) NOT
NULL , CU
```

‘173CAF180EA757F8’ is the timestamp/CONTOKEN value of the DBRM member SOMEPGM$ in this example. The prefix ‘X’ is specified just to let you know that it is stored in packed form.

Now press F1 in your keyboard (make sure the F1 key is mapped to Help. If it is not, F1 will not work. Change the keyboard mapping for F1 to point to Help). Otherwise, type HELP in the command prompt after seeing the short message.

(Warning: pressing F1 or issuing the HELP command should be immediately followed by your command GETTOKEN.)

```
   VIEW     SOME.TESTPROD.DBRMLIB(SOMEPGM$) - 01.00  X'173CAF180EA757F8'
Command ===> help                                        Scroll ===> CSR
****** ************************** Top of Data ***************************
000001 DBRM  µYOURID4 SOMEPGM$ ±¢J  ^_  K
000002
000003 DBRM  Ç  ¬  <DECLARE X_PLT_TYPES TABLE ( TAGS_ID CHAR ( 2 ) NOT
NULL ,
```

```
000004 , DEAL_PARTY_ID CHAR ( 16 ) NOT NULL , DEAL_NAME CHAR ( 30 ) NOT
NULL ,
000005 UPDT_DT DECIMAL ( 9 , 0 ) NOT NULL , BRTH_STATE CHAR ( 1 ) NOT
NULL , CU
```

After the HELP command or F1 (pointing to the HELP command) is issued, the system will send a long message in the middle of the screen (this is again based on your settings – Option 0 settings from the ISPF Primary Option Menu).

```
VIEW     SOME.TESTPROD.DBRMLIB(SOMEPGM$) - 01.00     X'173CAF180EA757F8'
Command ===>                                           Scroll ===> CSR
****** *************************** Top of Data ***************************
000001 DBRM  µYOURID4 SOMEPGM$ ±¢J  ^_  K
000002
000003 DBRM  Ç  ¬  <DECLARE X_PLT_TYPES TABLE ( TAGS_ID CHAR ( 2 ) NOT
NULL ,
000004 , DEAL_PARTY_ID CHAR ( 16 ) NOT NULL , DEAL_NAME CHAR ( 30 ) NOT
NULL ,
000005 UPDT_DT DECIMAL ( 9 , 0 ) NOT NULL , BRTH_STATE CHAR ( 1 ) NOT
NULL , CU

                              X'0EA757F8173CAF18'
```

## SOME MORE FACTS

If you look at the real timestamp value appearing at the top right-hand corner of the screen and the one appearing in the middle of the screen, you will see that the first 8 bytes and the second 8 bytes have swapped their positions.

The real contoken (the one that appears at the top right corner) might say, for example, X'AAAAAAAABBBBBBBB'.

The other contoken value appearing after F1 or the HELP command, will now be X'BBBBBBBBAAAAAAAA'.

This is because some load modules will have their contoken timestamp in the swapped form.

## REVIEW

You can now verify the result with the load module as follows. Open the corresponding load module, SOMEPGM$, and find

the string X'173CAF180EA757F8' or X'0EA757F8173CAF18'. If the load module and DBRM are to be in sync (ie created during the same compile process), one of the contoken strings (real or swapped) should be found in the load module.

The REAL contoken string can also be found in the SYSPACKAGE table of DB2 system catalog under the CONTOKEN COLUMN for this DBRM member.

*Suresh Kumar Murugesan*
*Systems Analyst*
*Cognizant Technology Solutions (USA)*


## Data compression using the CSRCMPSC service

This short article offers a simple demonstration of the use of the CSRCMPSC service. I was unable to find a working sample program on the Web that showed how to exploit CSRCMPSC. CSRCMPSC is basically a wrapper around the CMPSC (Compression Call) instruction that tests whether the Synchronous Data Compression Hardware is available on this CPU, and then either issues the CMPSC instruction or calls an emulation routine. The compression requires a pair of user-supplied dictionaries in order to work. These dictionaries have to be built using representative sample data from the application. IBM supplies a REXX procedure in SYS1.SAMPLIB(CSRBDICT) to do this. How to use this EXEC is not obvious; the only usage instructions I've found are in the EXEC itself.

This is how I created the dictionaries with CSRBDICT:

```
//ST01Ø   EXEC PGM=IKJEFT1A,DYNAMNBR=32ØØ
//SYSPROC  DD  DSN=SYS1.SAMPLIB,DISP=SHR
//SYSTSPRT DD  SYSOUT=*,OUTLIM=2ØØØØ
//OUTFILE  DD  SYSOUT=*
//SYSTSIN  DD  *
 PROF PREFIX(EXSYAT)
 %CSRBDICT 4 1 EB 'dset.with.sample.data' ('scanfile.specfile'
```

*Scanfile.specfile* is an FB 80 control card dataset with these control cards in it:

```
R       40000    60        3        f 7 2 7  1000     af asm
aam    x   x        h       n        x
FLD 1   sa   dce 4               INT  aeis 1 (40)
FLD end
```

Running the EXEC generates six new datasets.

The datasets with low-level qualifiers ACDICT41 and AEDICT41 contain a pair of newly-generated compression and expansion dictionaries. The dictionaries are generated as a pair of Assembler source code CSECTs that need to be assembled and linked into a load module before use.

In the demo program shown here, I've statically linked the dictionaries in the load module together with the executable code. Regardless of the method you use to get the dictionaries in storage, you must make sure they start on a page boundary or the compression service will not work (CMPSC will result in a 0C4). The demo program reads a variable-blocked input dataset and writes a variable-blocked output dataset. Each input record is individually passed through the CSRCMPSC and written to the output dataset. The record length of the output records is set to the length calculated after compression or expansion. When compressing an input file and expanding it again into another dataset, the original and the expanded datasets appear equal to each other with ISPF Supercompare.

In the assembly and link-edit JCL, notice the datasets holding the dictionary CSECTs are concatenated, and notice how to specify that the module must load on a page boundary.

```
//ST010    EXEC PGM=ASMA90,PARM='NOTEST,OBJECT,RENT,BATCH'
//SYSUT1    DD  UNIT=VIO,SPACE=(CYL,(1,1))
//SYSPRINT DD  SYSOUT=*
//SYSLIB    DD  DSN=SYS1.MACLIB,DISP=SHR
//          DD  DSN=SYS1.MODGEN,DISP=SHR
//          DD  DSN=SYS1.ASM.SASMMAC2,DISP=SHR  assembler toolkit maclib
//SYSLIN    DD  DSN=&&OBJ,DISP=(MOD,PASS),
//          SPACE=(CYL,(1)),UNIT=SYSDA,
//          DCB=(LRECL=80,BLKSIZE=3200,RECFM=FB)
//SYSPUNCH DD  DUMMY
```

```
//SYSIN    DD  *
ESPRESSO CSECT ,              ESPRESSO program, main entry point
ESPRESSO AMODE 31             ESPRESSO may receive control in 31-bitmode
ESPRESSO RMODE ANY            ESPRESSO load module may reside above 16MB
 PRINT NOGEN,NODATA           don't list macro expansions
 COPY ASMMSP                  include the structured macro defs
 YREGS ,                      OS reg equates
 SAVE (14,12),,'ESPRESSO'     save callers regs
 LR R12,R15                   R12 = copy of program epa
 USING ESPRESSO,R12           R12 is base reg in csect
 LH R2,=AL2(BYTES_REQUIRED)  R2 = size of working storage
 STORAGE OBTAIN,LENGTH=(R2),LOC=BELOW  get working storage area
 LR R11,R1                    R11 = base reg for working storage
 USING WORKAREA,R11           establish addressability on work area
 LR RØ,R1                     RØ = copy of working storage address
 LR R1,R2                     R1 = copy of working storage size
 XR R15,R15                   R15 = Ø, length for MVCL
 MVCL RØ,R14                  pad entire storage block with nulls
 USING WORKAREA,R11           establish addressability on work area
 ST R13,MAINSAVE+4            link save areas backwards
 LR R1,R13                    R1 = copy of ptr to old save area
 LA R13,MAINSAVE              R13 = address of new save area
 ST R13,Ø(R1)                 link save areas forward
 L  R1,24(R1)                 R1 = pointer to parameter list
 MVC EYE,=CL4'MEYE'           set eyecatcher in working storage
                                                                SPACE
 L  R1,Ø(R1)                  R1 = address of parm string from jcl
 IF (CLC,=C'COMPRESS',NE,2(R1)),AND,(CLC,=C'EXPAND',NE,2(R1))
     LH R2,=AL2(BYTES_REQUIRED) R2 = size of working storage
     L  R13,MAINSAVE+4        R13 = address of previous save area
     STORAGE RELEASE,ADDR=(R11),LENGTH=(R2) free working storage
     RETURN (14,12),RC=8,T    return to caller with error rc
 ENDIF
 IF (CLC,=C'EXPAND',EQ,2(R1)) if this is an EXPAND request
     MVI  FLAG,CMPSC_EXPAND     set our expand flag
     MVC  DICTIONARY,=V(AEDICT) set our pointer to expansion dict
 ELSE ,                       else it's a compress request
     MVC  DICTIONARY,=V(ACDICT) set our pointer to compression dict
 ENDIF
                                                                SPACE
 MVC OUTFILE,OUTFILE_I        init OUTFILE dcb in dynamic storage
 MVC INFILE,INFILE_I          init INFILE dcb in dynamic storage
 MVC OPEN_L,OPEN_I            init OPEN macro in dynamic storage
 OPEN (INFILE,(INPUT),OUTFILE,(OUTPUT)),MODE=31,MF=(E,OPEN_L)
                                                                SPACE
 DO INF                       do until end of input data (EODAD)
    GET INFILE                read a record from INFILE
    LR  R2,R1                 R2 = copy of pointer to INFILE buffer
    PUT OUTFILE               get new output buffer from qsam
    LR  R3,R1                 R3 = copy of pointer to OUTFILE buffer
```

```
    XC  CMPSC(CMPSC_LEN),CMPSC             clear parm block
    MVI CMPSC_FLAGS_BYTE2,CMPSC_SYMSIZE_4  Set symbol size 4
    OC  CMPSC_FLAGS_BYTE2,FLAG             put in compress/expand flag
    MVC CMPSC_DICTADDR,DICTIONARY  Set dictionary address
    L   R4,=F'32710'          R4 = length of target area, outfile buffer
    ST  R4,CMPSC_TARGETLEN    Set length of target area
    LA  R0,4(R3)              R0 = address of target area
    ST  R0,CMPSC_TARGETADDR   Set target area address, OUTFILE buffer
    LA  R0,4(R2)              R0 = address of text to compress
    ST  R0,CMPSC_SOURCEADDR   Set source area, INFILE buffer
    LH  R0,0(R2)              R0 = length of source area input rec RDW
    SL  R0,=F'4'              R0 = length of source area minus rdw
    ST  R0,CMPSC_SOURCELEN    Set length of text to process
    LA  R0,WA                 R0 = address of work area for service rtn
    ST  R0,CMPSC_WORKAREAADDR  Set workarea address
                                                                   SPACE
    CSRCMPSC CBLOCK=CMPSC     call the compression service routine
    IF (CL,R15,NE,=A(CMPSC_RETCODE_OK))  if non-zero return code
        LR R2,R15             R2 = copy of return code from CSRCMPSC
        ABEND 100,REASON=(2) program abend 100 with reason in R2
    ENDIF
                                                                   SPACE
    SL  R4,CMPSC_TARGETLEN    R4 = target length before minus after cmp
    LA  R4,4(R4)              R4 = record length including 4 bytes rdw
    IF (CLI,FLAG,EQ,0),AND,  if doing compress  AND  BITNUM non-zero   C
              (TM,CMPSC_DICTADDR_BYTE3,CMPSC_BITNUM,NZ)
        LA  R4,1(R4)          add one extra byte to the record length
    ENDIF
    STH R4,0(R3)              store record length in output RDW
 ENDDO
EODAD DS 0H                   31 bit eond of data address in DCBE
 MVC CLOSE_L,CLOSE_I          init CLOSE macro in dynamic storage
 CLOSE (INFILE,,OUTFILE),MODE=31,MF=(E,CLOSE_L)
                                                                   SPACE
 LH R2,=AL2(BYTES_REQUIRED)  R2 = size of working storage
 L  R13,MAINSAVE+4           R13 = address of previous save area
 STORAGE RELEASE,ADDR=(R11),LENGTH=(R2) free working storage
 RETURN (14,12),RC=0,T       return to caller with rc zero
 LTORG ,                     literals here
OUTFILE_I DCB DDNAME=OUTFILE,  skeleton dcb to init dynamic storage    C
               LRECL=32710,                                            C
               RECFM=VB,                                               C
               DSORG=PS,                                               C
               MACRF=PL
INFILE_I   DCB DDNAME=INFILE,  skeleton dcb to init dynamic storage    C
               LRECL=32710,                                            C
               RECFM=VB,                                               C
               DSORG=PS,                                               C
               MACRF=GL,                                               C
               DCBE=INFILE_DCBE
```

```
INFILE_DCBE DCBE EODAD=EODAD   dcbe to enable 31 bit EODAD
OPEN_I  OPEN  (,(INPUT),,(OUTPUT)),MODE=31,MF=L skeleton OPEN list
CLOSE_I CLOSE (,,,,),MODE=31,MF=L               skeleton CLOSE list
                                                          SPACE
WORKAREA DSECT ,             map of working storage area
MAINSAVE DS 18F              save area for main program
EYE      DS CL4              eyecatcher in case dump
         CSRYCMPS DSECT=NO   inline compression parameter block
         DS ØD               align workarea on doubleword bdy
WA       DS CL192            Work area
FLAG     DS C                compress/expand flag
DICTIONARY DS A              ptr to compression or expansion dictionary
OUTFILE_ DCB DDNAME=OUTFILE,DSORG=PS,MACRF=PL
OUTFILE  EQU OUTFILE_,*-OUTFILE_   label to init dcb in dynamic storage
INFILE_  DCB DDNAME=INFILE,DSORG=PS,MACRF=GL,DCBE=INFILE_DCBE
INFILE   EQU INFILE_,*-INFILE_     label to init dcb in dynamic storage
OPEN_    OPEN (,,,,),MODE=31,MF=L
OPEN_L   EQU OPEN_,*-OPEN_   parm list for OPEN macro
CLOSE_   CLOSE (,,,,),MODE=31,MF=L
CLOSE_L  EQU CLOSE_,*-CLOSE_ parm list for CLOSE macro
BYTES_REQUIRED EQU *-WORKAREA size of working storage
 END ESPRESSO
 PRINT OFF                   don't list assembly of dictonaries
//         DD  DSN=EXSYAT.CSRCMPSC.TEST1ØK.ACDICT41,DISP=SHR
//         DD  DSN=EXSYAT.CSRCMPSC.TEST1ØK.AEDICT41,DISP=SHR
//*
//ST02Ø   EXEC PGM=IEWL,PARM='NOTEST,XREF,LIST,XREF,RENT'
//SYSPRINT DD  SYSOUT=*
//SYSUT1   DD  UNIT=VIO
//SYSLIB   DD  DSN=EXSYAT.TSO.JCL,DISP=SHR
//SYSLMOD  DD  DSN=EXSYAT.ESPRESSO.LOADLIB(ESPRESSO),DISP=SHR
//SYSLIN   DD  DSN=&&OBJ,DISP=OLD
//         DD  *
    ORDER ACDICT,AEDICT,ESPRESSO
    PAGE  ACDICT
    ENTRY ESPRESSO
    NAME  ESPRESSO(R)
//*
//ST03Ø   EXEC PGM=ESPRESSO,PARM=COMPRESS
//STEPLIB  DD  DSN=EXSYAT.ESPRESSO.LOADLIB,DISP=SHR
//SYSUDUMP DD  SYSOUT=*
//INFILE   DD  DSN=EXSYAT.INFILE,DISP=SHR
//OUTFILE  DD  DSN=EXSYAT.INTER,DISP=SHR
//*
//ST04Ø   EXEC PGM=ESPRESSO,PARM=EXPAND
//STEPLIB  DD  DSN=EXSYAT.ESPRESSO.LOADLIB,DISP=SHR
//SYSUDUMP DD  SYSOUT=*
//INFILE   DD  DSN=EXSYAT.INTER,DISP=SHR
//OUTFILE  DD  DSN=EXSYAT.OUTFILE,DISP=SHR
```

We ran the demo program on one of our 2084 machines and did a number of tests to measure CPU consumption. The tests have shown that data compression with CMPSC is very CPU intensive – only slightly less so than 100% software compression with CSRCESRV. This indicates that CMPSC is little more than a simple call to the microcode in the HSA of the processor, rather than dedicated silicon doing the hard work. The main advantage to this pseudo-hardware compression is the better compression ratio that can be attained, but we also noticed that CMPSC eats more CPU cycles when it compresses better.

*Tim Alpaerts*
*Systems Engineer*
*Euroclear (Belgium)*


## DFSORT performance tuning aid – part 2

*This month we conclude our look at examining DFSORT performance data.*

### LOOKING AT SMF TYPE 30 RECORDS WITH ICETOOL

As already mentioned, two SMF record types are particularly useful for analysing the performance of DFSORT – type 30 and type16. For type 30 records, make sure that subtype 4 (step total) records are collected. If device connect time and EXCP statistics are desired, the DETAIL option must be in effect. SMF type 30 (subtype 4) records contain useful information on the resource consumption of a particular job step. These include CPU time (total, and broken down by field), elapsed time, EXCPs (total, and broken down by device), and device connect time.

These SMF records provide a wealth of performance data, which is often difficult to analyse in its raw form. You need to

convert this raw data into processed data that can be used for trend analysis and management reports. If you do not have access to a performance analysis and reporting product, you can use the DFSORT editing functions in combination with the reporting capabilities of ICETOOL or OUTFIL to generate performance data reports. The INCLUDE/OMIT, OUTFIL, and INREC/OUTREC control statements are helpful in selecting the specific fields of the particular SMF, RMF, or other data records to be analysed. ICETOOL or OUTFIL can then be used to generate printable reports from the resulting raw data. See *z/OS DFSORT Application Programming Guide* (SC26-7523-00) for more information about ICETOOL, OUTFIL, and editing functions.

ICETOOL was used to select and display SMF type 30 records. Run this job to determine what offsets to use for the next job:

```
//*-----------------------------------------
//* OFFSET30 JCL: run this job to
//* find out what SMF record 30/4 offsets
//* to use for the next job.
//* Identification section  (SMF30IOF)
//* I/O Activity section    (SMF30UOF)
//* Processor section       (SMF30COF)
//* Storage section         (SMF30COF)
//* Performance section     (SMF30POF)
//*-----------------------------------------
//TOOL1    EXEC PGM=ICETOOL
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=your.smf.weekly.ds,DISP=SHR
//SMFOFF   DD DSN=&&SMFOFF,DISP=(,PASS),SPACE=(CYL,(50,25)),
//         UNIT=SYSDA,DCB=(RECFM=VB,LRECL=32756,BLKSIZE=0)
//SRT1CNTL DD *
  OPTION DYNALLOC,VLSHRT,STOPAFT=10,SPANINC=RC4
  INCLUDE COND=(6,1,BI,EQ,X'1E',AND,23,2,BI,EQ,X'0004')
  SORT FIELDS=(7,4,FI,A)
/*
//TOOLMSG DD SYSOUT=*
//REPORT  DD SYSOUT=*
//TOOLIN  DD *
  SORT FROM(RAWSMF) TO(SMFOFF) USING(SRT1)
  DISPLAY FROM(SMFOFF) LIST(REPORT)           -
  TITLE('SMF Record 30/4 Offsets display')    -
  PAGE DATE TIME BLANK                        -
```

```
* Use values displayed for a, b, c, d, e in the next ICETOOL job
  HEADER('ID. Sec. (a)')      ON(33,4,FI)  -
  HEADER('I/O Sec.(b)')       ON(41,4,FI)  -
  HEADER('Proc Sec.(c)')      ON(57,4,FI)  -
  HEADER('Storage Sec.(d)')   ON(73,4,FI)  -
  HEADER('Perf. Sec. (e)')    ON(81,4,FI)
/*
//*
```

Output from this job looks like this:

```
SMF Record 30/4 Offsets display

ID. Sec.(a)   I/O Sec.(b)   Proc Sec.(c)   Storage Sec(d)   Perf Sec.(e)
----------   ----------   ------------   --------------   ------------
        214           398            470              554            754
```

Use a=214, b=398, c=470, d=554, and e=754 in the next ICETOOL job:

```
//*-------------------------------------------------------------------
//* Extract type 30 smf records from a dumped smf dataset
//*-------------------------------------------------------------------
//TOOL2    EXEC PGM=ICETOOL,REGION=0M
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=your.smf.weekly.ds,DISP=SHR
//SMFEXTR  DD DSN=&&ASID30,DISP=(,PASS),
//         SPACE=(CYL,(50,50)),UNIT=SYSALLDA
//COPYOUT  DD DSN=&&ASID,DISP=(,PASS),
//         SPACE=(TRK,(30,30)),UNIT=SYSALLDA
//DFSPARM  DD *
  OPTION SMF=FULL
//SRT1CNTL DD *
  OPTION DYNALLOC,VLSHRT,SPANINC=RC4
*Only these records that comply to:
*id offset = 214, excp > 0 and pgm = icetool/iceman
  INCLUDE COND=(33,4,BI,EQ,X'000000D6',
  AND,403,4,BI,GT,X'00000000',AND,
  (223,6,CH,EQ,C'ICETOO',OR,223,5,CH,EQ,C'ICEMA'))
  SORT FIELDS=(11,4,PD,A,7,4,BI,A)
/*
//SRT2CNTL DD *
 OPTION COPY,VLSHRT
* Run previous icetool job (OFFSET30) to get values for a, b, c, d, e
* and substitute with the appropriate values
* SMF30LEN  (RDW)  0+1=1 start of record
* SMF30TME  (TIME) 6+1=7 start of record
* SMF30DTE  SMF record write date: 10+1
*
* Identification Section offset (a): 214 + 1 = 215
```

```
*        off len
*  0+215: 215,8 SMF30JBN - Job or session name.
*  8+215: 223,8 SMF30PGM - Program name
* 32+215: 247,8 SMF30JNM - JES job identifier.
*
* I/O Activity Section offset (b): 398 + 1 = 399
*  4+399: 403,4 SMF30TEP - EXCP count
* 18+399: 417,4 SMF30TCN - Total device connect time for this ASID.
* 32+399: 431,4 SMF30AIC - DASD I/O connect time
* 36+399: 435,4 SMF30AID - DASD I/O disconnect time
* 40+399: 439,4 SMF30AIW - DASD I/O pending + control unit queue time
*
* Processor Accounting Section offset (c): 470 + 1 = 471
*  4+471: 475,4 SMF30CPT - CPU time under the TCB
*  8+471: 479,4 SMF30CPS - CPU time under the SRB
* 44+471: 515,4 SMF30IIP - CPU time used to process I/O interrupts
* 48+471: 519,4 SMF30RCT - CPU time used by the region control task
* 52+471: 523,4 SMF30HPT - CPU time used to transfer data:HSP <-> ASID
*
* Storage and Paging Section offset (d): 554 + 1 = 555
*  4+555: 559,2 SMF30PRV - Largest stg used from bottom of priv.
*  6+555: 561,2 SMF30SYS - Largest stg used from top of priv.
*  8+555: 563,4 SMF30PGI - Paged in from AUX.
* 12+555: 567,4 SMF30PGO - Paged out to AUX
* 20+555: 575,4 SMF30NSW - # of ASID swap sequences.
* 24+555: 579,4 SMF30PSI - # of pages swapped in from AUX storage to CS.
* 28+555: 583,4 SMF30PSO - # of pages swapped out from CS to AUX.
* 32+555: 587,4 SMF30VPI - # of VIO page-ins from AUX to CS.
* 36+555: 591,4 SMF30VPO - # of VIO page-outs from CS to AUX.
* 44+555: 599,4 SMF30CPI - # of Comm area page-ins from AUX to CS.
* 48+555: 603,4 SMF30HPI - # of HSP page-ins from AUX to PROC stg.
* 52+555: 607,4 SMF30LPI - # of LPA page-ins from AUX to CS.
* 56+555: 611,4 SMF30HPO - # of HSP page-outs from PROC stg. to AUX.
* 72+555: 627,4 SMF30RGB - Private size in bytes (< 16 megabytes).
* 76+555: 631,4 SMF30ERG - Private size in bytes (> 16 megabytes).
* 80+555: 635,4 SMF30ARB - Maxvirt alloc from LSQA and the
*                                                  SWA subpools  (<16M)
* 84+555: 639,4 SMF30EAR - Maxvirt alloc from LSQA and the
*                                                  SWA subpools  (>16M)
* 88+555: 643,4 SMF30URB - Maxvirt alloc from the user
*                                                   subpools (< 16 M B)
* 92+555: 647,4 SMF30EUR - Maxvirt alloc from the user
*                                                   subpools (> 16 M B)
* 96+555: 651,4 SMF30RGN - Region size established
*100+555: 655,4 SMF30DSV - Amount of dsp and hsp VS used
*
* Performance Section offset (e): 754 + 1 = 755
*  0+755: 755,4 SMF30SRV - Total service units.
*  4+755: 759,4 SMF30CSU - CPU service units.
*  8+755: 763,4 SMF30SRB - Service request block (SRB) service units.
```

```
* 12+755: 767,4 SMF3ØIO  - I/O service units.
* 16+755: 771,4 SMF3ØMSO - Main storage occupancy (MSO) service units.
*
 OUTREC FIELDS=(1,4,7,4,11,4,215,8,223,8,247,8,4Ø3,4,417,4,431,4,
        435,4,439,4,475,4,479,4,515,4,519,4,523,4,559,2,561,2,
        563,4,567,4,575,4,579,4,583,4,587,4,591,4,599,4,6Ø3,4,
        6Ø7,4,611,4,627,4,631,4,635,4,639,4,643,4,647,4,651,4,
        655,4,755,4,759,4,763,4,767,4,771,4)
*
//TOOLMSG  DD SYSOUT=*
//SERVICE  DD SYSOUT=*            <-- DFSORT Service performance data
//CPURPT   DD SYSOUT=*            <-- DFSORT CPU performance data
//SUMMARY  DD SYSOUT=*            <-- DFSORT Summary report
//STORAGE  DD SYSOUT=*            <-- DFSORT Storage report
//DASD     DD SYSOUT=*            <-- DFSORT I/O report
//PAGE     DD SYSOUT=*            <-- DFSORT Paging report
//SWAP     DD SYSOUT=*            <-- DFSORT Swap report
//HSP      DD SYSOUT=*            <-- DFSORT Hiperspace report
//TOOLIN   DD *
  SORT FROM(RAWSMF)  TO(SMFEXTR) USING(SRT1)
  COPY FROM(SMFEXTR) TO(COPYOUT) USING(SRT2)
  DISPLAY FROM(COPYOUT) LIST(STORAGE)          -
    TITLE('DFSORT Storage report')             -
    PAGE DATE TIME BLANK                       -
    HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
    HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
    HEADER('Jes2 #')        ON(29,8,CH)                 -
    HEADER('Job')           ON(13,8,CH)                 -
    HEADER('Pgm')           ON(21,8,CH)                 -
    HEADER('Region - kb ')  ON(149,4,FI)                -
    HEADER('Dsp Hsp mb')    ON(153,4,FI)                -
    HEADER('Bott stg KB')   ON(77,2,FI)                 -
    HEADER('Top  stg KB')   ON(79,2,FI)                 -
    HEADER('p < 16 kb')     ON(125,4,FI,/KB)            -
    HEADER('p > 16 kb')     ON(129,4,FI,/KB)            -
    HEADER('VS< 16 kb')     ON(133,4,FI,/KB)            -
    HEADER('VS> 16 kb')     ON(137,4,FI,/KB)            -
    HEADER('Max VS<16 mb')  ON(141,4,FI,/KB)            -
    HEADER('Max VS>16 mb')  ON(145,4,FI,/KB)
  DISPLAY FROM(COPYOUT) LIST(PAGE)             -
    TITLE('DFSORT Paging report')              -
    PAGE DATE TIME BLANK                       -
    HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
    HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
    HEADER('Jes2 #')        ON(29,8,CH)                 -
    HEADER('Job')           ON(13,8,CH)                 -
    HEADER('Pgm')           ON(21,8,CH)                 -
    HEADER('Page in')       ON(81,4,FI)                 -
    HEADER('Page out')      ON(85,4,FI)                 -
    HEADER('VIO page in')   ON(1Ø1,4,FI)                -
```

```
  HEADER('VIO page out')  ON(1Ø5,4,FI)                -
  HEADER('Comm page in')  ON(1Ø9,4,FI)                -
  HEADER('LPA page in')   ON(117,4,FI)                -
  HEADER('HSP page in')   ON(113,4,FI)                -
  HEADER('HSP page out')  ON(121,4,FI)
DISPLAY FROM(COPYOUT) LIST(SWAP)            -
  TITLE('DFSORT Swap report')               -
  PAGE DATE TIME BLANK                      -
  HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
  HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
  HEADER('Jes2 #')        ON(29,8,CH)                 -
  HEADER('Job')           ON(13,8,CH)                 -
  HEADER('Pgm')           ON(21,8,CH)                 -
  HEADER('Swap seq.')     ON(89,4,FI)                 -
  HEADER('Swap in')       ON(93,4,FI)                 -
  HEADER('Swap out')      ON(97,4,FI)
DISPLAY FROM(COPYOUT) LIST(DASD)            -
  TITLE('DFSORT I/O report')                -
  PAGE DATE TIME BLANK                      -
  HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
  HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
  HEADER('Jes2 #')        ON(29,8,CH)                 -
  HEADER('Job')           ON(13,8,CH)                 -
  HEADER('Pgm')           ON(21,8,CH)                 -
  HEADER('Excp')          ON(37,4,FI,A2)              -
  HEADER('Tot.conn')      ON(41,4,FI)                 -
  HEADER('Dasd conn')     ON(45,4,FI)                 -
  HEADER('Dasd disconn')  ON(49,4,FI)                 -
  HEADER('Dasd pend.')    ON(53,4,FI)
DISPLAY FROM(COPYOUT) LIST(CPURPT)          -
  TITLE('DFSORT CPU performance data')      -
  PAGE DATE TIME BLANK                      -
  HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
  HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
  HEADER('Jes2 #')        ON(29,8,CH)                 -
  HEADER('Job')           ON(13,8,CH)                 -
  HEADER('Pgm')           ON(21,8,CH)                 -
  HEADER('Excp')          ON(37,4,FI,A2)              -
  HEADER('tcb')           ON(57,4,FI,F1)              -
  HEADER('srb')           ON(61,4,FI,F1)              -
  HEADER('iip')           ON(65,4,FI,F1)              -
  HEADER('rct')           ON(69,4,FI,F1)              -
  HEADER('hpt')           ON(73,4,FI,F1)
DISPLAY FROM(COPYOUT) LIST(SERVICE)         -
  TITLE('DFSORT Service performance data')  -
  PAGE DATE TIME BLANK                      -
  HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
  HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
  HEADER('Jes2 #')        ON(29,8,CH)                 -
  HEADER('Job')           ON(13,8,CH)                 -
```

```
    HEADER('Pgm')           ON(21,8,CH)                 -
    HEADER('Total su')      ON(157,4,FI,A2)             -
    HEADER('Tcb su')        ON(161,4,FI,A2)             -
    HEADER('Srb su')        ON(165,4,FI,A2)             -
    HEADER('Io su')         ON(169,4,FI,A2)             -
    HEADER('MSO su')        ON(173,4,FI,A2)
  DISPLAY FROM(COPYOUT) LIST(SUMMARY)          -
    TITLE('DFSORT Summary report')             -
    PAGE DATE TIME BLANK                       -
    HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
    HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
    HEADER('Jes2 #')        ON(29,8,CH)                 -
    HEADER('Job')           ON(13,8,CH)                 -
    HEADER('Pgm')           ON(21,8,CH)                 -
    HEADER('Excp')          ON(37,4,FI,A2)              -
    HEADER('Conn.')         ON(41,4,FI)                 -
    HEADER('tcb')           ON(57,4,FI,F1)              -
    HEADER('srb')           ON(61,4,FI,F1)              -
    HEADER('Serv')          ON(157,4,FI,A2)             -
    HEADER('Region - kb ')  ON(149,4,FI)                -
    HEADER('Dsp Hsp mb')    ON(153,4,FI)
  DISPLAY FROM(COPYOUT) LIST(HSP)              -
    TITLE('DFSORT Hiperspace report')          -
    PAGE DATE TIME BLANK                       -
    HEADER('Date')          ON(Ø9,4,DT1,E'9999/99/99')  -
    HEADER('Time')          ON(Ø5,4,TM1,E'99:99:99')    -
    HEADER('Jes2 #')        ON(29,8,CH)                 -
    HEADER('Job')           ON(13,8,CH)                 -
    HEADER('Pgm')           ON(21,8,CH)                 -
    HEADER('Dsp Hsp mb')    ON(153,4,FI)                -
    HEADER('hpt')           ON(73,4,FI,F1)              -
    HEADER('HSP page in')   ON(113,4,FI)                -
    HEADER('HSP page out')  ON(121,4,FI)
/*
```

## CONCLUDING REMARKS

Investigating DFSORT performance does not need to take a lot of effort or a long time, and I have provided two simple tools that might help you in setting up a DFSORT performance yardstick. It should be noted that I have used SMF records as a data source, but these sources do not supply all the data needed to perform a thorough analysis of DFSORT performance and resource consumption. Thus, to tune DFSORT thoroughly, you may need extensive data about all DFSORT applications run at your site, including elapsed time,

CPU time, number of EXCPs, and types and amounts of virtual storage used. DFSORT enables you to provide an installation termination exit (ICETEXIT) routine, which can be used to collect extensive performance-related information about all DFSORT applications at a site. The advantage of using an ICETEXIT routine is that all the information about each DFSORT application is available to the routine. You do not need to use information from a number of other sources.

*Mile Pekic*
*Systems Programmer (Serbia and Montenegro)* 

## DCOLLECT – an in-depth introduction

The Data Collection Facility of DFSMS is actually a function of Access Method Services and is commonly referred to as DCOLLECT. The Data Collection facility is used to collect data in a sequential file that can then be interpreted using applications and programs in a variety of languages.

The main functionality of the Data Collection Facility and the data you can extract is shown below:

- Active datasets – data about space usage and dataset attributes can be extracted for selected disk volumes or SMS storage groups.
- VSAM datasets – detailed information that relates to VSAM datasets can be extracted for selected disk volumes and storage groups.
- Disk volumes – statistics and information for selected disk volumes can be collected.
- Inactive data – DFSMS HSM-managed data relating to inactive data management can be collected for both backed up and migrated datasets.

 

- Migrated datasets – information about space utilization and the dataset attributes of migrated datasets can be extracted.
- Backed up datasets – information about space utilization and the dataset's attributes can be extracted for every backed up dataset.
- Capacity planning for DFHSM-managed data – the collection of both tape and disk capacity planning is possible for DFHSM-managed volumes for ML0 and ML1. In addition, tape volume statistics and data can be extracted if managed by DFHSM.
- SMS configuration data – DCOLLECT provides information about SMS configurations. Information can be collected from an active control dataset (ACDS) or a source control dataset (SCDS), or from the active configuration itself.

  DCOLLECT provides attributes that are in the selected configuration for the following areas:

  – data class constructs
  – storage class constructs
  – management class constructs
  – storage group constructs
  – SMS volume information
  – SMS base configuration information
  – aggregate group construct information
  – optical drive information
  – optical library information
  – cache names
  – accounting information for the ACS.

## SMS DCOLLECT EXTRACTION

Information on active and inactive dataset space utilization and capacity planning can be obtained through the ISMF Data Collection application to produce a DCOLLECT job. DCOLLECT provides measurement data in a sequential dataset. This can then be used to produce billing information or reports to show what is being utilized and where it is being utilized.

DCOLLECT can be used to produce measurement data on active datasets (ie datasets that have not been backed up or migrated), inactive datasets (ie datasets that have been backed up or migrated), capacity planning (which concerns volume capacity and usage). Information can be obtained on the following:

- Active datasets
- Volumes
- Migrated datasets (DFSMShsm)
- Back-up datasets (DFSMShsm)
- DASD capacity planning (DFSMShsm)
- Tape capacity planning (DFSMShsm).

You can use on-line ISMF panels to generate JCL for the IDCAMS DCOLLECT command. The job/jobs created will need to run while the DFSMS environment is active. To use these panels, select the Data Collection Application option on the ISMF Primary Option Menu for Storage Administrators. This panel is illustrated below:

```
                ISMF PRIMARY OPTION MENU - z/OS DFSMS V1 R3
Enter Selection or Command ===>
Select one of the following options and press Enter:
Ø  ISMF Profile                    - Specify ISMF User Profile
1  Data Set                        - Perform Functions Against Data Sets
2  Volume                          - Perform Functions Against Volumes
3  Management Class      - Specify Data Set Backup and Migration Criteria
4  Data Class                  - Specify Data Set Allocation Parameters
5  Storage Class          - Specify Data Set Performance and Availability
```

```
6  Storage Group        - Specify Volume Names and Free Space Thresholds
7  Automatic Class Selection    - Specify ACS Routines and Test Criteria
8  Control Data Set          - Specify System Names and Default Criteria
9  Aggregate Group              - Specify Data Set Recovery Parameters
1Ø Library Management        - Specify Library and Drive Configurations
11 Enhanced ACS Management       - Perform Enhanced Test/Configuration
                                                          Management
C  Data Collection                - Process Data Collection Function
L  List                      - Perform Functions Against Saved ISMF Lists
R  Removable Media Manager   - Perform Functions Against Removable Media
X  Exit                           - Terminate ISMF
Use HELP Command for Help; Use END Command or X to Exit.
```

To enter the Data Collection panel, type C as your selection and press *Enter*. You will then be presented with the panel shown below:

```
                      DATA COLLECTION ENTRY PANEL            Page 1 of 3
Command ===>

Select Data Collection options:
  Data Set Information  . . . N         (Y or N; Y requires volume(s) or
  Volume Information  . . . . Y           storage group(s) on next page)
  Migration Data  . . . . . . N                       (Y or N)
  Backup Data . . . . . . . . N                       (Y or N)
  Capacity Planning Data  . . N                       (Y or N)
  SMS Data  . . . . . . . . . N                       (Y or N)
Specify Output Data Set:                        (1 to 44 Characters)
  Data Set Name . . . . . . . 'EXB7884.SMS'
  Optional Password . . . . .          (Ignored if SMS-managed data set)
  Replace Contents  . . . . . N                (Y or N)
  Number of Data Sets . . . . 1       (1 to 99999999; new data set only)
Specify Input Data Set:                         (1 to 44 Characters)
  Migration Data Set Name . .
  Backup Data Set Name  . . .
  CDS Name  . . . . . . . . . 'ACTIVE'

Use ENTER to Perform Selection; Use DOWN Command to View next Entry
Panel;
Use HELP Command for Help; Use END Command to Exit.
```

This is the first panel of three that exist.

The fields on the entry panel for the Data Collection application are initialized to collect data only for active datasets and volumes. You can also collect information on migration data, back-up data, and capacity planning data. You can specify Y (yes) in any combination of the fields under SELECT DATA

COLLECTION OPTIONS.

The following are details of the fields on the entry panel, what they mean, and how they can be used:

- DATASET INFORMATION – if you want information collected on active datasets selected by storage group or volume, specify Y (yes) in this field. Otherwise, specify N (no). The default is Y.
- VOLUME INFORMATION – if you want to collect information on active volumes selected by storage group or volume, specify Y in this field. Otherwise, specify N. The default is Y.
- MIGRATION DATA – if you want to collect information on migration data, specify Y in this field. Otherwise specify N. If you specify Y, you must specify a migration dataset name in the MIGRATION DATA SET NAME field. The default is N.
- BACKUP DATA – if you want to collect information on back-up data, specify Y in this field. Otherwise specify N. If you specify Y, you must specify a back-up dataset name in the BACKUP DATA SET NAME field. The default is N.
- CAPACITY PLANNING DATA – if you want to collect information on capacity planning, specify Y in this field. Otherwise specify N. If you specify Y, you must specify both a migration dataset name and a back-up dataset name. The default is N.
- DATA SET NAME – you must enter the name of an output dataset in this field. DCOLLECT puts the information it collects in the dataset you specify. The dataset can be either an existing dataset or a dataset that is allocated during the data collection job while DFSMS is active.
- OPTIONAL PASSWORD – if the dataset is not system-managed and it has a password, you must specify the password in the OPTIONAL PASSWORD field. Password protection is less secure than RACF protection. The

dataset must be catalogued. This field is ignored for system-managed datasets.

- REPLACE CONTENTS – if you want DCOLLECT to overlay any data, which is already in the output dataset, specify Y in the REPLACE CONTENTS field. If you want DCOLLECT to append newly-collected data to the data already contained in the output dataset, specify N in the REPLACE CONTENTS field. N is the default, that is, data already contained in the output dataset is not erased when the job is submitted. If the dataset has never been written in, it doesn't matter whether you specify Y or N.
- NUMBER OF DATA SETS – if you are specifying an output dataset that is to be allocated during the data collection job, you should estimate the total number of datasets that reside on the volumes about which you are collecting data. Specify this number in the NUMBER OF DATA SETS field. This number is used to estimate the size of the output dataset. If you save the JCL generated, instead of submitting it directly from this application, you can edit the DD statement for the output dataset to change the space calculation.
- MIGRATION DATA SET NAME – if you are collecting migration data or information for capacity planning, you must specify a dataset name in the MIGRATION DATA SET NAME field. This dataset must contain migration information. For DFSMShsm, this dataset is the Migration Control Data Set (MCDS) for either the system you are running on or the system about which you want information. This dataset must be accessible to the system running the DCOLLECT job.
- BACKUP DATA SET NAME – if you are collecting back-up data or information for capacity planning, you must specify a dataset name in the BACKUP DATA SET NAME field. This dataset must contain back-up information. For DFSMShsm, this dataset is the Backup Control Data Set (BCDS) for either the system you are running on or the

system about which you want information. This dataset must be accessible to the system running the DCOLLECT job.

When you are collecting capacity planning information, you must specify migration and back-up datasets that are on the same DFSMShsm system.

If you are collecting information on active datasets or volumes, use the DOWN command to reach the next page of the Data Collection Entry panel.

On the second panel, you must specify one or more volumes or storage groups about which DCOLLECT is to collect information. If you specified Y in the DATA SET INFORMATION or VOLUME INFORMATION fields on the first page of the DCOLLECT entry panel, you must specify at least one volume or storage group on the second panel.

If you are collecting information about active datasets, DCOLLECT collects information on the active datasets on the volumes and in the storage groups you specify. It will not collect information about other datasets on those volumes and in those storage groups. If you are collecting information on active volumes, DCOLLECT collects information about the volumes and in the storage groups you specify. It ignores information about the datasets on those volumes.

When you have finished entering values on the Data Collection Entry panel, press *Enter* to display the DCOLLECT Job Submission Entry panel. On this panel, you can choose to submit the job you just specified, or save the job to a dataset, which you can then edit. You can edit the job statement on this screen before submitting the job.

## MONITORING SPACE USAGE WITH DCOLLECT

By monitoring storage activities at an installation, user requirements can be discovered easily. By using the IDCAMS DCOLLECT command, it is possible to track changing space

trends, identify ineffective use of space, and detect space-related errors. This information can then be utilized to make decisions on how to improve overall system performance.

To ensure efficient use of DASD space, you need to make sure that space is allocated and used according to storage management standards and practices that pertain to your environment. By using the information DCOLLECT produces on active and inactive data, and on DASD and tape volumes, you can monitor space usage for your whole installation.

DCOLLECT can be used to gather information on active and inactive data. The information it collects on active data is

| *Record type* | *Type ID* | *Record contents* |
|---|---|---|
| Active dataset | D | Dataset name<br>Dataset Organization<br>RACF-defined dataset indicator<br>System-managed dataset indicator<br>Temporary data set indicator<br>Partitioned dataset extended (PDSE) indicator<br>Hierarchical file system (HFS) dataset indicator<br>Generation data group dataset indicator<br>System-reblockable dataset indicator<br>Catalogued dataset indicator<br>Integrated catalog indicator (if dataset is an integrated catalog)<br>Dataset VVR indicator<br>Volume serial number<br>Dataset record format<br>Dataset record length<br>Dataset block length<br>Dataset size before compression<br>Dataset size after compression<br>Dataset size validation indicator<br>Amount of space allocated (kilobytes)<br>Amount of space actually used (kilobytes) |

*Figure 1: DCOLLECT record types (cont.)*

| *Record type* | *Type ID* | *Record contents* |
|---|---|---|
| | | Number of extents used |
| | | Secondary allocation |
| | | Number of unusable bytes in blocks |
| | | Creation date |
| | | Expiration date |
| | | Last reference date |
| | | Dataset serial number |
| | | Volume sequence number |
| | | Date and time of last back-up |
| | | Data class name |
| | | Management class |
| | | Storage class name |
| | | Storage group name |
| | | Error information |
| Volume information | V | Volume serial number |
| | | VTOC index status |
| | | Volume usage (Private/Public/Storage) |
| | | Shared device status |
| | | Percentage of free space |
| | | Amount of free space (kilobytes) |
| | | Amount of allocated space (kilobytes) |
| | | Total capacity of the volume (kilobytes) |
| | | Fragmentation index of the volume |
| | | Largest extent (kilobytes) |
| | | Number of free extents |
| | | Number of free DSCBs in the VTOC |
| | | Number of VIRs |
| | | Volume device type |
| | | Volume device number |
| | | Storage group name |
| | | Error information |
| VSAM association | A | Dataset component name |
| | | Dataset cluster name |
| | | High used relative byte address |
| | | High allocated relative byte address |
| | | Number of logical records |
| | | Number of deleted records |
| | | Number of inserted records |
| | | Number of updated records |
| | | Number of retrieved records |
| | | Bytes of free space in the dataset |

*Figure 1: DCOLLECT record types (cont.)*

| *Record type* | *Type ID* | *Record contents* |
|---|---|---|
| | | Number of CI splits |
| | | Number of CA splits |
| | | Number of EXCPs |
| | | Relative key position |
| | | Key length |
| | | Dataset size before compression |
| | | Dataset size after compression |
| Migration dataset | M | Dataset name |
| | | Dataset organization |
| | | Dataset record format |
| | | Dataset block length |
| | | RACF-defined dataset indicator |
| | | System-managed generation data group dataset indicator |
| | | System-reblockable dataset indicator |
| | | Partitioned dataset extended (PDSE) indicator |
| | | Space allocated (kilobytes) |
| | | Space used (kilobytes) |
| | | Migration level where the dataset currently resides |
| | | Number of migrations |
| | | Recall space estimate |
| | | Changed-since-last-back-up indicator |
| | | Device class of the volume |
| | | Size of migrated copy (kilobytes) |
| | | Date and time of the dataset migration |
| | | Creation date |
| | | Expiration date |
| | | Date of last back-up |
| | | Date last referenced |
| | | Data class name |
| | | Management class name |
| | | Storage class name |
| | | Storage group name |
| Dataset back-up version | B | Dataset name |
| | | Dataset record format |
| | | Dataset block length |
| | | RACF-defined dataset indicator |

*Figure 1: DCOLLECT record types (cont.)*

| *Record type* | *Type ID* | *Record contents* |
|---|---|---|
| | | System-managed generation data group dataset indicator<br>System-reblockable dataset indicator<br>Partitioned dataset extended (PDSE) indicator<br>Space allocated (in kilobytes)<br>Space used (kilobytes)<br>Recovery space estimate<br>Device class (tape or DASD) of back-up volume<br>Back-up version size in kilobytes<br>Date and time of latest back-up version<br>Data class name<br>Management class name<br>Storage class name |
| DASD Capacity Planning | C | DASD volume serial number<br>Type of volume (primary or migration level 1)<br>Total capacity of the volume (in tracks)<br>Date DASD capacity planning data was collected<br>Specified target occupancy of primary volume (low threshold)<br>Specified trigger occupancy of primary or migration level 1 volume, (high threshold for primary, threshold for migration level 1)<br>Percentage occupancy of the volume before HSM space management processing<br>Percentage occupancy of the volume after HSM space management processing<br>Percentage of the volume's data not migrated but eligible to migrate |

*Figure 1: DCOLLECT record types (cont.)*

| *Record type* | *Type ID* | *Record contents* |
|---|---|---|
| | | Number of times the HSM interval migration function was run against the volume |
| | | Number of times the target occupancy was met for the volume during interval migration |
| Type Capacity Planning | T | Type of HSM tape (back-up, dump, migration) |
| | | Number of full tape volumes |
| | | Number of partially filled tape volumes |
| | | Number of empty tape volumes |

*Figure 1: DCOLLECT record types (cont.)*

placed in the DCOLLECT active dataset records. The inactive data information, which will consist of backed up data and migration data information, will be kept in DCOLLECT back-up records and migration records. DCOLLECT also gathers VSAM dataset space use (HURBA) information. Figure 1 gives a summary of these record types and their contents.

The information in these records can:

- Identify wasted space in primary storage. Using the information from the active dataset records, you can determine the amount of allocated but unused space in your installation. This can then be used to put into place a plan to reclaim unused space, or to assign datasets a management class that releases allocated but unused space for specific datasets.
- Track wasted storage because of inactive datasets that are not migrated, or datasets that can reside on tape but are kept on DASD. You can use the information in migration or back-up records to help you track wasted storage and then act on that information.
- Assess the overall space savings from compressing VSAM

KSDS or sequential datasets. DCOLLECT collects information on the size of datasets before and after compression.

DCOLLECT will gather information on volumes and storage groups. This information is kept in the DCOLLECT volume records. You can use the information to monitor free space capacity and storage group thresholds. Monitoring free space capacity and storage group thresholds can then be used to help meet the expected amount of free space in the storage groups. It also helps prevent space-related abends if acted on and monitored regularly.

Some example JCL is shown below for batch jobs that invoke DCOLLECT via IDCAMS to produce certain data.

The DCOLLECT JCL to extract migrated and backed up dataset information:

```
//EXB7884A  JOB  (7884),CLASS=A,MSGCLASS=H,MSGLEVEL(1,1)
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=*
//MCDS      DD   DSN=CUM1222.MCDS,DISP=SHR
//BCDS      DD   DSN=CUM1222.BCDS,DISP=SHR
//OUTDS     DD   DSN=EXB7884.OPUT,
//         SPACE=(CYL,(100,25),RLSE),
//         DISP=(NEW,CATALG,DELETE),
//         STORCLASS=BIGDSET
//SYSIN    DD      *
   DCOLLECT OFILE(OUTDS) -
                      MIGRATEDATA -
                      BACKUPDATA
```

The DCOLLECT JCL to extract generic volume data:

```
//EXB7884A  JOB  (7884),CLASS=A,MSGCLASS=H,MSGLEVEL(1,1)
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=*
//OUTDS     DD   DSN=EXB7884.OPUT,
//         SPACE=(CYL,(100,25),RLSE),
//         DISP=(NEW,CATALG,DELETE),
//         STORCLASS=BIGDSET
//SYSIN     DD      *
      DCOLLECT -
              OUTFILE(DCOUT) -
              VOLUMES( -
              PRD*)
```

The amount of DCOLLECT output generated will depend on the size of the DASD subsystem at your site and the number of datasets (active and inactive) you have. Because a large amount of raw data can be collected over a period of time, you obviously need to put into place management routines to ensure that data can be summarized and manipulated.

The data itself can be manipulated using SQL, QMF, ICETOOL, or standard programming languages. Or it can be downloaded to a PC platform and manipulated using products such as spreadsheets or PC-based programs.

You can measure DASD and DFSMShsm-owned tape capacity so that capacity planners can identify current requirements and evaluate future needs.

This data is useful in helping to predict additional space requirements before the demand becomes critical.

You can use information from any of the DCOLLECT records to help track volume capacity. The information in the volume records can be used to determine whether space use is increasing or decreasing – thus highlighting whether you can meet your space requirements with current disk capacity or whether you require more. Below is an example of JCL that can be used to collect capacity information:

```
//EXB7884A   JOB  (7884),CLASS=A,MSGCLASS=H,MSGLEVEL(1,1)
//STEP1      EXEC PGM=IDCAMS
//SYSPRINT   DD   SYSOUT=*
//OUTDS      DD   DSN=EXB7884.OPUT,
//          SPACE=(CYL,(100,25),RLSE),
//          DISP=(NEW,CATALG,DELETE),
//          STORCLASS=BIGDSET
//SYSIN      DD      *
       DCOLLECT OFILE(OUTDS) -
                VOL(PRD*) -
                STOG(STG1,STG2) -
                CAPD
/*
//
```

The syntax of the DCOLLECT command as issued via IDCAMS is shown below:

```
DCOLLECT       {OUTFILE(ddname[/password])|
       OUTDATASET(entryname[/password])}
       {[VOLUMES(volser[ volser...])]
       [STORAGEGROUP(sgname[ sgname...])]
       [MIGRATEDATA]
       [BACKUPDATA]
       [CAPPLANDATA]
       [SMSDATA(SCDSNAME(entryname)|ACTIVE)]}
       [REPLACE|APPEND]
       [NODATAINFO]
       [NOVOLUMEINFO]
       [ERRORLIMIT(value)]
       [EXITNAME(entrypoint)]
       [MIGRSNAPALL|MIGRSNAPERR]
```

DCOLLECT itself can be abbreviated to DCOL when written in the SYSIN datastream. Also note that although BACKUPDATA, CAPPLANDATA, MIGRATEDATA, STORAGEGROUP, VOLUMES, and SMSDATA are designated as optional parameters, you must use at least one of them during processing. There is also a DCOLLECT user exit that can be invoked to manipulate the records that are produced. This is not covered here but details about it can be found in the relevant DFSMS manuals.

To control access to the DCOLLECT function, a RACF check for authorization is made for a facility class profile of STGADMIN.IDC.DCOLLECT. If this profile exists, then read authority is necessary. The command will not be successful if the user is not authorized against this facility class.

The parameters of the DCOLLECT command that are required are shown below:

```
OUTDATASET(entryname[ /password]) -
```

*Outdataset* identifies the target dataset. You must use a physical sequential dataset with a record format of V or VB. Use an LRECL that is at least the size of the longest DCOLLECT record to be collected. Changes to the JCL are not necessary if you use an LRECL larger than the longest record to be collected. The LRECL should be at least as large as the longest record DCOLLECT generates but not larger than 32756.

Enter the name of a DD statement that identifies the target dataset:

```
OUTFILE(ddname[/password]) -
```

Abbreviation is OFILE.

*Outdataset* is the name of the target dataset. If you use this, the entryname is dynamically allocated with a status of either OLD or MOD, as required by the REPLACE parameter:

```
OUTDATASET(entryname[/password]) -
```

Abbreviation is ODS.

*Password* is the update or higher-level password for a password-protected target dataset or path. Passwords are ignored for SMS-managed datasets and catalogs:

```
password -
```

DCOLLECT optional parameters are:

```
BACKUPDATA -
```

*BACKUPDATA* requires that information on backed up datasets be collected from the given back-up control dataset (BCDS). The desired BCDS must be allocated to the DDname BCDS.

The abbreviation is BACD.

To include capacity planning information in the output dataset use:

```
CAPPLANDATA -
```

Allocate the MCDS to the DDname MCDS and the BCDS to the DDname BCDS.

The abbreviation is CAPD.

*ERRORLIMIT* is the maximum number of errors for which detailed DCOLLECT error messages can print during program run:

```
ERRORLIMIT(value) -
```

*ERRORLIMIT* prevents runaway message output. The default

for ERRORLIMIT is 2,147,483,647 errors, but any number between 1 and 2,147,483,647 can be given. Processing continues even though the error limit has been reached.

The abbreviation is ELIMIT.

*EXITNAME* is the 1 to 8 character entrypoint name for an external DCOLLECT user exit module:

```
EXITNAME(entrypoint) -
```

Load it to an APF-authorized library for access at the time of DCOLLECT invocation. If you do not use it, the default DCOLLECT user exit, IDCDCX1, is used.

The abbreviation is EXIT.

*MIGRATEDATA* collects information about migrated datasets from the specified MCDS (Migration Control Data Set):

```
MIGRATEDATA -
```

The desired MCDS must be allocated to the DDname MCDS.

The abbreviation is MIGD.

*MIGRSNAPALL* asks ARCUTIL to do SNAP processing, and is used for diagnostic reasons only:

```
MIGRSNAPALL -
```

Do not use it with MIGRSNAPERR. It is ignored if you do not use MIGRATEDATA, BACKUPDATA, or CAPPLANDATA.

The abbreviation is MSALL.

*MIGRSNAPERR* requires ARCUTIL to run SNAP processing when an error occurs during ARCUTIL processing:

```
MIGRSNAPERR -
```

Use it for diagnostic purposes only. Do not use it with MIGRSNAPALL. It is ignored if you do not use MIGRATEDATA, BACKUPDATA, or CAPPLANDATA.

The abbreviation is MSERR.

*NODATAINFO* says that no dataset information records are generated or written to the output dataset:

```
NODATAINFO -
```

Use this parameter if you want only volume information generated for the given volumes or storage groups.

The abbreviation is NOD.

*NOVOLUMEINFO* says that no volume information records are generated or written to the output dataset:

```
NOVOLUMEINFO -
```

Use this parameter if you want only dataset information generated for the given volumes or storage groups.

The abbreviation is NOV.

*REPLACE|APPEND* specifies whether the output data is to replace existing data or whether it is to be added at the end of the existing dataset:

```
REPLACE|APPEND -
```

*REPLACE|APPEND* applies when OUTDATASET is used. If you use OUTFILE, dataset processing is controlled by the JCL DISP parameter – OLD replaces the current contents of the dataset, and MOD appends new records to the end of the dataset.

*REPLACE* asks that the contents of the output dataset are overwritten with new data. All existing data in the output dataset is lost when this parameter is selected.

The abbreviation is REPL.

*APPEND* writes new records starting at the end of the existing data, if any exists. All existing data is preserved when this parameter is selected.

The abbreviation is APP

SMS configuration data can be included in the DCOLLECT output dataset by:

```
SMSDATA(SCDSNAME(entryname)|ACTIVE) -
```

This parameter can include either an SCDS name or the keyword ACTIVE. One or more of the following record types is created when you use SMSDATA:

- DC – Data Class construct information
- SC – Storage Class construct information
- MC – Management Class construct information
- BC – Base Configuration information
- SG – Storage Group construct information
- VL – Storage Group volume information
- AG – Aggregate Group information
- DR – OAM Drive Record information
- LB – OAM Library Record information
- CN – Cache Names from the base configuration information
- AI – Accounting Information for the ACS.

The abbreviation is SMS.

The subparameters of SMSDATA are:

- SCDSNAME(entryname[/password]) – the source of the SMS control data that is to be collected.
- entryname – used to specify the name of an existing catalogued SCDS. An enqueue with a major name of IGDCDS is issued to serialize access to the control dataset. The enqueue is held for the duration of SMSDATA processing.

  The abbreviation is SCDS
- ACTIVE – takes the SMS information from the configuration that is currently active on the system.
- STORAGEGROUP(sgname[ sgname...]) – lists the storage

groups from which information is to be collected. For each storage group listed, a list of on-line volume serials is generated. Information is collected for all datasets residing on those volumes unless you use NODATAINFO. Volume information is collected unless you use NOVOLUMEINFO. A maximum of 255 storage groups can be selected. Although several storage groups can be specified, and the volume list might have duplicates, each volume's information is processed only once.

The abbreviation is STOG.

- VOLUMES(volser[ volser...]) – lists the volumes from which information is to be collected. For each on-line volume serial listed (or resolved from generic specifications), information is collected for all datasets residing on those volumes unless you use NODATAINFO. Volume information is collected unless you use NOVOLUMEINFO. You can use a maximum of 255 volume serials. Although the same volumes can be specified several times, each volume's information is processed only once.

  The abbreviation is VOL.

*Elizabeth Bradley*
*Systems Programmer*
*Meerkat Computer Services Ltd (UK)*


## Search a PDS for a specific string

This EXEC invokes SuperC SRCHFOR from an ISPF dialog that can be launched from anywhere. It can be launched from any TSO/ISPF command line or Option 3.4 for any PDS. The output can be either the SuperC report or a usable memberlist showing the number of hits and allowing you to edit or delete the members.

## FINDIT REXX EXEC

```
/*********************************************************************/
/*                                REXX                               */
/* Purpose: Search a PDS for the provided search string              */
/* Syntax:  FINDIT anystring                                         */
/* Parms: PDS        - The PDS to search (optional)                  */
/*        OUTOPT     - The output option (optional)                  */
/* Notes: OUTOPT is used to select the format for the results.  The  */
/*        default is REPORT which presents the ISPF SRCHFOR report   */
/*        in ISPF Browse.  The MEMLIST option will present a member  */
/*        list with all members containing the string.               */
/* Standard housekeeping activities                                  */
call time 'r'
parse arg parms
signal on syntax name trap
signal on failure name trap
signal on novalue name trap
probe = 'NONE'
modtrace = 'NO'
modspace = ''
call stdentry 'DIAGMSGS'
module = 'MAINLINE'
push trace() time('L') module 'From:' Ø 'Parms:' parms
if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
call modtrace 'START' Ø
/* Set local estoeric names                                          */
@vio   = 'VIO'
@sysda = 'SYSDA'
/* Accept the string and PDS                                         */
arg ippds string
if ippds = '' then
    call ispwrap 8 "VGET (IPPDS OUTOPT) PROFILE"
else
    ippds = strip(ippds,"B","'")
/* If STRING or IPPDS is missing, display the dynamic panel          */
if string = '' | ippds = '' then
    do
     search.1  = ")ATTR                                          "
     search.2  = " @ TYPE(TEXT) COLOR(TURQ)                      "
     search.3  = " # TYPE(INPUT) CAPS(ON) COLOR(RED)             "
     search.4  = ")BODY EXPAND(//) WINDOW(6Ø,5)                  "
     search.5  = "+Command%===>_ZCMD                             "
     search.6  = "%Specify the string and the PDS to search      "
     search.7  = "@Search for  : #Z                              "
     search.8  = "@Search PDS  : #Z                              "
     search.9  = "@Output Style: #Z        +(%REPORT+or%MEMLIST+)"
     search.1Ø = ")INIT                                          "
     search.11 = " .ZVARS = '(STRING IPPDS OUTOPT)'              "
     search.12 = " .CURSOR = STRING                              "
```

```
     search.13 = " IF (&OUTOPT = &Z)                                    "
     search.14 = "     &OUTOPT = REPORT                                 "
     search.15 = ")PROC                                                 "
     search.16 = " VER (&STRING,NB)                                     "
     search.17 = " VER (&IPPDS,NB,DSNAMEF)                              "
     search.18 = " VER (&OUTOPT,NB,LIST,REPORT,MEMLIST)                 "
     search.19 = ")END                                                  "
/* Display the Dynamic Panel                                           */
     call popdyn 'SEARCH' 4 'Enter' execname 'Parameters'
/* VPUT the DSN specified for future use                               */
     call ispwrap "VPUT (IPPDS OUTOPT) PROFILE"
     call ispwrap "REMPOP"
    end
/* Allocate the PDS to NEWDD                                           */
call tsotrap "ALLOC F(NEWDD) DA("qdsn(ippds)") SHR REU"
/* Determine the desired output option                                 */
call lock 'Searching for' string 'in' ippds
if outopt = 'REPORT' then
    do
/* Allocate the OUTDD and SYSIN required by SUPERC                     */
     call viodd 'OUTDD' 132
     sysin.1 = "SRCHFOR '"string"'"
     sysin.2 = "LPSFV 1"
     sysin.3 = "NTITLE '" execname "Search:" string"'"
     call viodd 'SYSIN'
/* Invoke SUPERC ISPF Search                                           */
     srchparm = 'SRCHCMP,ANYC,LPSF,NOPRTCC,SDUPM,NOSUMS'
     address ATTCHMVS "ISRSUPC" "SRCHPARM"
     SUPERRC = RC
     call ispwrap "REMPOP"
/* Browse the output                                                   */
     if SUPERRC = 1 then
        call brwsdd 'OUTDD'
     else
        call msg '"'string'" was not found in' ippds
    end
else
    do
/* Allocate the OUTDD and SYSIN required by SUPERC                     */
     call viodd 'OUTDD' 132
     sysin.1 = "SRCHFOR '"string"'"
     call viodd 'SYSIN'
/* Invoke SUPERC ISPF Search                                           */
     srchparm = 'SRCHCMP,ANYC,NOPRTCC,SDUPM,NOSUMS'
     address ATTCHMVS "ISRSUPC" "SRCHPARM"
     SUPERRC = RC
/* Generate a member list                                              */
     if SUPERRC = 1 then
        do
/* Create an ISPF table to hold the member list                        */
```

```
         call ispwrap "VGET (ZSCREEN)"
         table = 'FINDIT'||zscreen
         call ispwrap 4 "TBCREATE" table "KEYS(MEMBER) NAMES(HITCOUNT)",
                        "REPLACE"
/* Read in the SRCHFOR output                                          */
         call tsotrap "EXECIO * DISKR OUTDD (STEM MEMLIST. FINIS"
         memcount = 0
/* Add members to the table                                            */
         do m=1 to memlist.0
            parse var memlist.m word1 word2 .
            if substr(memlist.m,2,1) <> ' ' then
               do
                parse var memlist.m member .
                if member <> 'LINE-#' then
                   do
                    memcount = memcount + 1
                    hitcount = 0
                    call ispwrap "TBADD" table
                   end
               end
            if datatype(word1) = 'NUM' then
               do
                hitcount = hitcount + 1
                call ispwrap "TBMOD" table
               end
         end
/* Display the member list                                             */
         call ispwrap "REMPOP"
         call ispwrap "TBTOP" table
         call setbord 'YELLOW'
         call ispwrap "ADDPOP"
         zwinttl = execname 'Member Selection List'
/* Build the TBDISPL                                                   */
         call ispwrap "VGET (ZSCREEND)"
         scrdepth = zscreend - 4
         tbdisp.1  = ")ATTR                                           "
         tbdisp.2  = " ~ AREA(DYNAMIC)                                "
         tbdisp.3  = " | TYPE(DATAOUT) COLOR(TURQ)                    "
         tbdisp.4  = " ^ TYPE(CHAR) COLOR(PINK)                       "
         tbdisp.5  = " $ TYPE(INPUT) PAD('_') CAPS(ON)                "
         tbdisp.6  = " # TYPE(OUTPUT) CAPS(OFF) COLOR(GREEN)          "
         tbdisp.7  = " @ TYPE(OUTPUT) COLOR(GREEN) JUST(RIGHT)        "
         tbdisp.8  = ")BODY EXPAND(//) WINDOW(46,"scrdepth")          "
         tbdisp.9  = "%Command ===>_ZCMD     %Scroll ===>_Z   +       "
         tbdisp.10 = "~CMDAREA,SHADOW                                ~  "
         tbdisp.11 = "+CMD Member          Hits                       "
         tbdisp.12 = ")MODEL                                          "
         tbdisp.13 = " $Z+#MEMBER    @HITCOUNT+                       "
         tbdisp.14 = ")INIT                                           "
         tbdisp.15 = " .ZVARS = '(FCSR FCMD)'                         "
```

```
         tbdisp.16 = "  IF (&FCSR = &Z)                                  "
         tbdisp.17 = "      &FCSR = 'CSR'                                "
         tbdisp.18 = " &CMDAREA = '|Line Commands: Select Edit Delete'"
         tbdisp.19 = " &SHADOW  = '                ^|     ^|   ^|    '"
         tbdisp.2Ø = ")PROC                                              "
         tbdisp.21 = " VER (&@TCMD,LIST,S,/,D,E)                         "
         tbdisp.22 = ")END                                               "
         tbdisp    = ispflib('TBDISP' 'ISPPLIB')
/* Display the member list                                              */
         do until PF3 = 8
            PF3 = ispwrap(8 "TBDISPL" table "PANEL(TBDISP)")
/* Process line commands                                               */
            if ztdsels > Ø then
               do
/* Select and Edit line commands                                       */
                if fcmd = 'S' | fcmd = 'E'  then
                   do
                    findmac.1 = "/* REXX */"
                    findmac.2 = "address ISREDIT"
                    findmac.3 = "MACRO"
                    findmac.4 = "FIND ALL '"string"'"
                    execddn   = execlib('FINDMAC')
                    dsnmem = qdsn(ippds('member')')
                    call ispwrap "REMPOP"
                    call ispwrap 4 "EDIT DATASET("dsnmem")",
                                   "MACRO(FINDMAC)"
                    call ispwrap "ADDPOP"
                    call execfree execddn
                   trace 'o'
                   end
/* delete line command                                                 */
                if fcmd = 'D' then
                   do
                    call tsotrap "FREE F(NEWDD)"
                    call ispwrap "LMINIT DATAID(PDS) ENQ(EXCLU)",
                                 "DATASET("qdsn(ippds)")"
                    call ispwrap "LMOPEN DATAID("PDS") OPTION(OUTPUT)"
                    call ispwrap "LMMDEL DATAID("PDS") MEMBER("member")"
                    call ispwrap "TBDELETE" table
                    call ispwrap "LMCLOSE DATAID("PDS")"
                    call ispwrap "LMFREE DATAID("PDS")"
                   end
               end
              fcmd = ''
         end
/* Clean up the ISPF resources                                         */
         call ispffree tbdisp
         call ispwrap "TBEND" table
         call setbord 'BLUE'
        end
```

```
     else
        call msg '"'string'" was not found in' ippds
    end
/* FREE everything                                                   */
call tsoquiet "FREE F(NEWDD)"
call tsotrap "FREE F(SYSIN)"
/* Shutdown                                                          */
shutdown: nop
/* Put unique shutdown logic before the call to stdexit              */
/* Shutdown message and terminate                                    */
          call stdexit time('e')
/* 37 Internal Subroutines provided in FINDIT                        */
/* Last Subroutine REFRESH was 26 Jan 2005 11:51:44                  */
/* RCEXIT   - Exit on non-zero return codes                          */
/* TRAP     - Issue a common trap error message using rcexit         */
/* ERRMSG   - Build common error message with failing line number    */
/* STDENTRY - Standard Entry logic                                   */
/* STDEXIT  - Standard Exit logic                                    */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine whether a required DD is allocated           */
/* DDLIST   - Returns number of DDs and populates DDLIST variable    */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* QDSN     - Make sure there are only one set of quotes             */
/* UNIQDSN  - Create a unique dataset name                           */
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                          */
/* ISPWRAP  - Wrapper for ISPF commands                              */
/* ISRWRAP  - Wrapper for ISPF Edit commands                         */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* WAIT     - Wait for a specified number of seconds                 */
/* SETBORD  - Set the ISPF Pop-up active frame border color          */
/* LOCK     - Put up a pop-up under foreground ISPF during long waits*/
/* UNLOCK   - Unlock from a pop-up under foreground ISPF             */
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel      */
/* POPDYN   - Addpop a Dynamic Panel                                 */
/* SAYDD    - Print messages to the requested DD                     */
/* JOBINFO  - Get job related data from control blocks               */
/* TSOQUIET - Trap all output from a TSO command and ignore failures */
/* BROWSE   - Capture TSO command output and BROWSE                  */
/* BRWSDD   - Invoke ISPF Browse on any DD                           */
/* VIODD    - EXECIO a stem into a sequential dataset                */
/* ISPFLIB  - Generate a Dynamic ISPF object                         */
/* ISPFFREE - Free a Dynamic ISPF object                             */
/* EXECLIB  - Generate a Dynamic REXX EXEC                           */
/* EXECFREE - Free a Dynamic REXX object                             */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* MODTRACE - Module Trace                                           */
/* RCEXIT   - Exit on non-zero return codes                          */
/* EXITRC   - Return code to exit with (if non-zero)                 */
/* ZEDLMSG  - Message text for it with for non-zero EXITRCs          */
```

```
rcexit: parse arg EXITRC zedlmsg
         EXITRC = abs(EXITRC)
         if EXITRC <> Ø then
            do
             trace 'o'
/* If execution environment is ISPF then VPUT ZISPFRC                  */
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/* Does not call ISPWRAP to avoid obscuring error message modules      */
                      address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/* If a message is provided, wrap it in date, time and EXITRC          */
             if zedlmsg <> '' then
                do
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                 call msg zedlmsg
                end
/* Write the contents of the Parentage Stack                           */
              stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/* Write to MSGDD if background and MSGDD exists                       */
             if tsoenv = 'BACK' then
                do
                 if subword(zedlmsg,9,1) = msgdd then
                    do
                     say zedlmsg
                     signal shutdown
                    end
                 else
                    do
                     call saydd msgdd 1 zedlmsg
                     call saydd msgdd 1 stacktitle
                    end
                end
              else
/* Write to the ISPF Log if foreground                                 */
                do
                 zerrlm = zedlmsg
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/* Unload the Parentage Stack                                          */
             do queued()
                pull stackinfo
                if tsoenv = 'BACK' then
                   do
```

```
                    call saydd msgdd Ø stackinfo
                   end
                else
                   do
                    zerrlm = stackinfo
                    address ISPEXEC "LOG MSG(ISRZØØ3)"
                   end
             end
/* Put a terminator in the ISPF Log for the Parentage Stack          */
             if tsoenv = 'FORE' then
                do
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
             signal shutdown
            end
         else
            return
/* TRAP     - Issue a common trap error message using rcexit         */
/* PARM     - N/A                                                    */
trap: trace 'off'
       traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       if trap = 'YES' & tsoenv = 'BACK' then
          do
           trap = 'NO'
           traplinemsg = msg
           say traplinemsg
           signal on syntax name trap
           signal on failure name trap
           signal on novalue name trap
           say
           say center(' Trace of failing instruction ',78,'-')
           trace 'i'
           interpret trapline
          end
       if trap = 'NO' & tsoenv = 'BACK' then
          do
           say center(' Trace of failing instruction ',78,'-')
           say
          end
       if tsoenv = 'FORE' then
          call rcexit 666 msg
       else
```

```
          call rcexit 666 traplinemsg
/* ERRMSG   - Build common error message with failing line number    */
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/* STDENTRY - Standard Entry logic                                   */
/* MSGDD    - Optional MSGDD used only in background                 */
stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/* Start up values                                                   */
           EXITRC = Ø
           MAXRC = Ø
           trap = 'YES'
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           headoff = 'NO'
           hcreator = 'NO'
           keepstack = 'NO'
           lpar = mvsvar('SYSNAME')
           jobname = mvsvar('SYMDEF','JOBNAME')
           zedlmsg = 'Default shutdown message'
/* Determine environment                                             */
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
               signal off failure
              "ISPQRY"
               ISPRC = RC
               if ISPRC = Ø then
                  do
                   ispfenv = 'YES'
/* Check if HEADING ISPF table exists already, if so set HEADOFF=YES */
                   call ispwrap "VGET (ZSCREEN)"
                   if tsoenv = 'BACK' then
                      htable = jobinfo(1)||jobinfo(2)
                   else
                      htable = userid()||zscreen
                   TBCRC = ispwrap(8 "TBCREATE" htable "KEYS(HEAD)")
                   if TBCRC = Ø then
                      do
                       headoff = 'NO'
```

```
                    hcreator = 'YES'
                   end
                else
                   do
                    headoff = 'YES'
                   end
               end
             signal on failure name trap
            end
/* MODTRACE must occur after the setting of ISPFENV                    */
          call modtrace 'START' sigl
/* Start-up message (if batch)                                        */
          startmsg = execname 'started' date() time() 'on' lpar
          if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
             headoff = 'NO' then
             do
              jobinfo = jobinfo()
              parse var jobinfo jobtype jobnum .
              say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
              say
              if ISPRC = -3 then
                 do
                  call saydd msgdd 1 'ISPF ISPQRY module not found,',
                                     'ISPQRY is usually in the LINKLST'
                  call rcexit 2Ø 'ISPF ISPQRY module is missing'
                 end
/* If MSGDD is provided, write the STARTMSG and SYSEXEC DSN to MSGDD */
              if msgdd <> '' then
                 do
                  call ddcheck msgdd
                  call saydd msgdd 1 startmsg
                  call ddcheck 'SYSEXEC'
                  call saydd msgdd Ø execname 'loaded from' sysdsname
/* If there are PARMS, write them to the MSGDD                        */
                  if parms <> '' then
                     call saydd msgdd Ø 'Parms:' parms
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD                 */
                  if listdsi('STEPLIB' 'FILE') = Ø then
                     do
                      steplibs = dddsns('STEPLIB')
                      call saydd msgdd Ø 'STEPLIB executables loaded',
                          'from' word(dddsns,1)
                      if dddsns('STEPLIB') > 1 then
                         do
                          do stl=2 to steplibs
                             call saydd msgdd Ø copies(' ',31),
                                  word(dddsns,stl)
                          end
                         end
                     end
```

```
                end
             end
/* If foreground, save ZFKA and turn off the FKA display             */
          else
             do
              fkaset = 'OFF'
              call ispwrap "VGET (ZFKA) PROFILE"
              if zfka <> 'OFF' & tsoenv = 'FORE' then
                 do
                  fkaset = zfka
                  fkacmd = 'FKA OFF'
                  call ispwrap "CONTROL DISPLAY SAVE"
                 call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
                  call ispwrap "CONTROL DISPLAY RESTORE"
                 end
             end
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/* STDEXIT  - Standard Exit logic                                    */
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
stdexit: module = 'STDEXIT'
          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 endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC          */
          EXITRC = max(EXITRC,MAXRC)
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
             headoff = 'NO' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
/* Make sure this isn't a MSGDD missing error then log to MSGDD      */
              if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                 do
                  call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                  call saydd msgdd 0 endmsg
                 end
             end
/* If foreground, reset the FKA if necessary                         */
          else
             do
              if fkaset <> 'OFF' then
                 do
```

```
                fkafix = 'FKA'
                call ispwrap "CONTROL DISPLAY SAVE"
                call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                if fkaset = 'SHORT' then
                   call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                "COMMAND(FKAFIX)"
                call ispwrap "CONTROL DISPLAY RESTORE"
               end
            end
/* Clean up the temporary HEADING table                              */
         if ispfenv = 'YES' & hcreator = 'YES' then
            call ispwrap "TBEND" htable
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there     */
         call modtrace 'STOP' sigl
         if queued() > Ø then pull . . module . sigl . sparms
         if queued() > Ø then pull . . module . sigl . sparms
         if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
            pull . . module . sigl . sparms
/* if the Parentage Stack is not empty, display its contents         */
         if queued() > Ø & keepstack = 'NO' then
            do
             say queued() 'Leftover Parentage Stack Entries:'
             say
             do queued()
                pull stackundo
                say stackundo
             end
             EXITRC = 1
            end
/* Exit                                                              */
         exit(EXITRC)
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* ZEDLMSG  - The long message variable                              */
msg: module = 'MSG'
      parse arg zedlmsg
      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
/* If this is background or OMVS use SAY                             */
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/* If this is foreground and ISPF is available, use SETMSG           */
         do
          if ispfenv = 'YES' then
/* Does not call ISPWRAP to avoid obscuring error message modules    */
             address ISPEXEC "SETMSG MSG(ISRZØØØ)"
          else
             say zedlmsg
```

```
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/* DDCHECK  - Determine whether a required DD is allocated          */
/* DD       - DDNAME to confirm                                     */
ddcheck: module = 'DDCHECK'
          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 dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/* Allow sysreason=3 & 22 to verify SYSOUT and tape DD statements   */
          if LRC <> Ø & sysreason <> 3 & sysreason <> 22 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/* DDLIST   - Returns number of DDs and populates DDLIST variable   */
/* N/A      - None                                                  */
ddlist: module = 'DDLIST'
         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
/* Trap the output from the LISTA STATUS command                    */
         call outtrap 'lines.'
         address TSO "LISTALC STATUS"
         call outtrap 'off'
         ddnum = Ø
/* Parse out the DDNAMEs and concatenate into a list                */
         ddlist = ''
         do ddl=1 to lines.Ø
            if words(lines.ddl) = 2 then
               do
                parse upper var lines.ddl ddname .
                ddlist = ddlist ddname
                ddnum = ddnum + 1
               end
            else
               do
                iterate
               end
```

```
         end
/* Return the number of DDs                                          */
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return ddnum
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* TARGDD   - DD to return DSNs for                                  */
dddsns: module = 'DDDSNS'
         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 targdd
         if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/* Trap the output from the LISTA STATUS command                     */
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
         dsnnum = Ø
         ddname = '$DDNAME$'
/* Parse out the DDNAMEs, locate the target DD and concatenate DSNs  */
         do ddd=1 to lines.Ø
            select
               when words(lines.ddd) = 1 & targdd = ddname &,
                    lines.ddd <> 'KEEP' then
                    dddsns = dddsns strip(lines.ddd)
               when words(lines.ddd) = 1 & strip(lines.ddd),
                    <> 'KEEP' then
                    dddsn.ddd = strip(lines.ddd)
               when words(lines.ddd) = 2 then
                    do
                     parse upper var lines.ddd ddname .
                     if targdd = ddname then
                        do
                         fdsn = ddd - 1
                         dddsns = lines.fdsn
                        end
                    end
               otherwise iterate
            end
         end
/* Get the last DD                                                   */
         ddnum = ddlist()
         lastdd = word(ddlist,ddnum)
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
```

```
/* Return the number of DSNs in the DD                                  */
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/* QDSN     - Make sure there are only one set of quotes                */
/* QDSN     - The DSN                                                   */
qdsn: module = 'QDSN'
       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 qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return qdsn
/* UNIQDSN  - Create a unique dataset name                              */
/* PARM     - N/A                                                       */
uniqdsn: module = 'UNIQDSN'
          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
/* Compose a DSN using userid, exec name, job number, date and time  */
          jnum = jobinfo(1) || jobinfo(2)
          udate = 'D'space(translate(date('O'),'','/'),Ø)
          utime = 'T'left(space(translate(time('L'),'',':.'),Ø),7)
          uniqdsn = userid()'.'execname'.'jnum'.'udate'.'utime
          if sysdsn(qdsn(uniqdsn)) = 'OK' then
             do
/* Wait 1 seconds to ensure a unique dataset (necessary on z99Ø)     */
              RC = syscalls('ON')
              address SYSCALL "SLEEP 1"
              RC = syscalls('OFF')
              uniqdsn = uniqdsn()
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return uniqdsn
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                             */
/* TEMPMEM  - The member to create                                      */
tempmem: module = 'TEMPMEM'
          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 tempmem
```

```
/* Create a unique DD name                                          */
          if length(tempmem) < 7 then
             tempdd = '$'tempmem'$'
          else
             tempdd = '$'substr(tempmem,2,6)'$'
/* If TEMPDD exists, then FREE it                                   */
          if listdsi(tempdd 'FILE') = Ø then "FREE F("tempdd")"
/* Generate the TEMPDSN                                             */
          tempdsn = uniqdsn()'('tempmem')'
/* ALLOCATE the TEMP DD and member                                  */
          call tsotrap "ALLOC F("tempdd") DA("qdsn(tempdsn)") NEW",
                       "LRECL(8Ø) BLKS(Ø) DIR(1) SPACE(1) CATALOG",
                       "UNIT("@sysda") RECFM(F B)"
/* Write the STEM to the TEMP DD                                    */
          call tsotrap 'EXECIO * DISKW' tempdd '(STEM' tempmem'. FINIS'
/* DROP the stem variable                                           */
          interpret 'drop' tempmem'.'
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return tempdd
/* ISPWRAP  - Wrapper for ISPF commands                             */
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to Ø */
/* ISPPARM  - Valid ISPF command                                    */
ispwrap: module = 'ISPWRAP'
          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 ispparm
          zerrlm = 'NO ZERRLM'
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          parse var ispparm valid_rc isp_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isp_cmd = ispparm
             end
          address ISPEXEC isp_cmd
          IRC = RC
/* If RC = Ø then return                                            */
          if IRC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return IRC
             end
          else
             do
```

```
             perrmsg = errmsg(sigl 'ISPF Command:')
             call rcexit IRC perrmsg isp_cmd strip(zerrlm)
            end
/* ISRWRAP  - Wrapper for ISPF Edit commands                          */
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to Ø */
/* ISRPARM  - Valid ISPF Edit command                                 */
isrwrap: module = 'ISRWRAP'
          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 isrparm
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          parse var isrparm valid_rc isr_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              isr_cmd = isrparm
             end
          parse var isr_cmd isr_verb .
          address ISREDIT isr_cmd
          ERC = RC
/* If RC = Ø then return                                              */
          if ERC <= valid_rc then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return ERC
             end
          else
             do
              if isr_verb = 'MACRO' & ERC = 2Ø then
                 do
                  call rcexit ERC 'is an Edit Macro and not valid to',
                              'run outside of ISPF Edit'
                 end
              else
                 do
                  rerrmsg = errmsg(sigl 'ISPF Edit Command:')
                  call rcexit ERC rerrmsg isr_cmd
                 end
             end
/* 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) <> 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 tsoparm
/* If the optional valid_rc parm is present use it, if not assume Ø  */
          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(ISRZØØØ)"
                      end
                      zedlmsg = trapmsg
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
```

```
                    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.Ø
                      say tsoout.c
                   end
                   say trapmsg
                   call rcexit CRC terrmsg tso_cmd
                  end
               end
             end
/* WAIT     - Wait for a specified number of seconds                  */
/* SECONDS  - Number of seconds to wait                               */
/* WMODE    - Use any value to stop printing batch wait messages      */
wait: module = 'WAIT'
       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 seconds wmode
       if datatype(seconds,'W') = Ø then seconds = 1Ø
       RC = syscalls('ON')
/* If foreground ISPF lock the screen                                 */
       if tsoenv = 'FORE' & ispfenv = 'YES' then
          call lock seconds 'second wait was requested'
/* If background, report the wait time                                */
       if tsoenv = 'BACK' & wmode = '' then
          call saydd msgdd Ø seconds 'second wait was requested'
/* Call USS SLEEP                                                     */
       address SYSCALL "SLEEP" seconds
/* If foreground ISPF lock the screen                                 */
       if tsoenv = 'FORE' & ispfenv = 'YES' then
          call unlock
       RC = syscalls('OFF')
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return
/* SETBORD  - Set the ISPF Pop-up active frame border color           */
/* COLOR    - Color for the Active Frame Border                       */
setbord: module = 'SETBORD'
         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 color
/* Parse and validate color                                           */
```

```
          if color = '' then color = 'YELLOW'
/* Build a temporary panel                                              */
          ispopt11.1=")BODY                        "
          ispopt11.2="%Command ===>_ZCMD        + "
          ispopt11.3=")INIT                        "
          ispopt11.4="&ZCMD = ' '                  "
          ispopt11.5="VGET (COLOR) SHARED          "
          ispopt11.6="&ZCOLOR = &COLOR             "
          ispopt11.7=".RESP = END                  "
          ispopt11.8=")END                         "
/* Allocate and load the Dynamic Panel                                  */
          setdd = tempmem('ISPOPT11')
/* LIBDEF the DSN, VPUT @TFCOLOR and run CUAATTR                        */
          call ispwrap "LIBDEF ISPPLIB LIBRARY ID("setdd") STACK"
          call ispwrap "VPUT (COLOR) SHARED"
          call ispwrap "SELECT PGM(ISPOPT) PARM(ISPOPT11)"
          call ispwrap "LIBDEF ISPPLIB"
          call tsotrap "FREE F("setdd") DELETE"
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/* LOCK     - Put up a pop-up under foreground ISPF during long waits*/
/* LOCKMSG  - Message for the pop-up screen                             */
lock: module = 'LOCK'
       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 lockmsg
       if lockmsg = '' then lockmsg = 'Please be patient'
       if tsoenv = 'FORE' then
          do
/* Use the length of the lockmsg to determine the pop-up size          */
           if length(lockmsg) < 76 then
              locklen = length(lockmsg) + 2
           else
              locklen = 77
           if locklen <= 1Ø then locklen = 1Ø
/* Build a temporary panel                                              */
           lock.1 = ")BODY EXPAND(//) WINDOW("locklen",1)"
           lock.2 = "%&LOCKMSG                    "
           lock.3 = ")END                         "
/* Lock the screen and put up a pop-up                                  */
           call ispwrap "CONTROL DISPLAY LOCK"
           call popdyn 'LOCK' 8 execname 'Please be patient'
           lockpop = 'YES'
          end
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
```

```
       interpret 'trace' tracelvl
       return
/* UNLOCK   - Unlock from a pop-up under foreground ISPF             */
/* PARM     - N/A                                                    */
unlock: module = 'UNLOCK'
         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
         if tsoenv = 'FORE' then
             do
              if lockpop = 'YES' then
                 do
                  call ispwrap "REMPOP"
                  lockpop = 'NO'
                 end
              if popup = 'YES' then
                 do
                  call setbord 'BLUE'
                  call ispwrap "REMPOP"
                  popup = 'NO'
                 end
             end
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return
/* PANDSN   - Create a unique PDS(MEM) name for a dynamic panel      */
/* PANEL    - Dynamic panel name                                     */
pandsn: module = 'PANDSN'
         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 panel
         pandsn = uniqdsn()'('panel')'
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return pandsn
/* POPDYN   - Addpop a Dynamic Panel                                 */
/* DYN      - Dynamic panel name                                     */
/* DYNROW   - Default row for ADDPOP, defaults to 1                  */
/* DYNMSG   - ADDPOP Window title                                    */
popdyn: module = 'POPDYN'
         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 dyn dynrow dynmsg
```

```
/* Set the default ADDPOP row location                                  */
        if dynrow = '' then dynrow = 1
/* Set the default ADDPOP window title to the current exec name         */
        if dynmsg = '' then dynmsg = execname
/* Check whether the RETURN option is specified in the DYNMSG           */
        dynreturn = 'NO'
        if word(dynmsg,1) = 'RETURN' then
           parse var dynmsg dynreturn dynmsg
/* Allocate and load the Dynamic Panel                                  */
        dyndd = tempmem(dyn)
/* LIBDEF the POPDYN panel                                              */
        call ispwrap "LIBDEF ISPPLIB LIBRARY ID("dyndd") STACK"
/* Change the Active Frame Color                                        */
        call setbord 'YELLOW'
/* set the POPUP variable if this is not a LOCK request                 */
        if dyn = 'LOCK' & popup = 'NO' then
           popup = 'NO'
        else
           popup = 'YES'
/* Put up the pop-up                                                    */
        zwinttl = dynmsg
        call ispwrap "ADDPOP ROW("dynrow")"
        DRC = ispwrap(8 "DISPLAY PANEL("dyn")")
        call ispwrap "LIBDEF ISPPLIB"
        call tsotrap "FREE F("dyndd") DELETE"
/* Change the Active Frame Color                                        */
        call setbord 'BLUE'
/* Determine how to return                                              */
        if dynreturn = 'NO' then
           call rcexit DRC 'terminated by user'
        if dynreturn = 'RETURN' & DRC = 8 then
           do
            call ispwrap "REMPOP"
            popup = 'NO'
           end
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return DRC
/* 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 > 0 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) > 60 & substr(message,1,2) <> '@@' then
            do
             messst = lastpos(' ',message,60)
             messseg = substr(message,1,messst)
             msgline.msgm = date() time() strip(messseg)
             message = strip(delstr(message,1,messst))
             do while length(message) > 0
                msgm = msgm + 1
                if length(message) > 55 then
                   messst = lastpos(' ',message,55)
                if messst > 0 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
/* JOBINFO  - Get job related data from control blocks               */
/* ITEM     - Optional item number desired, default is all           */
jobinfo: module = 'JOBINFO'
          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 item
/* Chase control blocks                                              */
          tcb      = ptr(54Ø)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+18Ø)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',Ø)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          progname = stg(jscb+36Ø,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
/* TSOQUIET - Trap all output from a TSO command and ignore failures */
/* TSOCMD   - TSO command to execute                                 */
tsoquiet: module = 'TSOQUIET'
           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 command to execute and throw away results                  */
           arg tsocmd
           call outtrap 'garbage.' Ø
           address TSO tsocmd
           call outtrap 'off'
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/* BROWSE   - Capture TSO command output and BROWSE                  */
/* BRWSCMD  - TSO Command to capture                                 */
browse: module = 'BROWSE'
         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 brwscmd
         if datatype(word(brwscmd,1)) = 'NUM' then
            parse var brwscmd start brwscmd
         else
            start = 1
/* Capture the command output                                        */
         call outtrap 'brout.'
         address TSO brwscmd
/* If no records captured                                            */
         if brout.Ø = Ø then
            do
             call msg 'No output from command:' brwscmd
             pull tracelvl . module . sigl . sparms
             call modtrace 'STOP' sigl
             interpret 'trace' tracelvl
```

```
            return
           end
        else
           do
            call msg 'Command executed:' brwscmd
           end
/* If a start location was provided, load stem with the subset       */
        if start > 1 then
           do i=1 to brout.Ø
              brout.i = substr(brout.i,start,71)
           end
/* Build a VIO dataset to hold the output                            */
        call viodd 'BROUT' 255 'V B'
/* Browse the VIO dataset                                            */
        call brwsdd 'BROUT'
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/* BRWSDD   - Invoke ISPF Browse on any DD                           */
/* BRWSDD   - Any DD to browse                                       */
brwsdd: module = 'BRWSDD'
        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 brwsdd
        if brwsdd = '' then call rcexit 9Ø 'Browse DD missing'
        call ispwrap "LMINIT DATAID(DATAID) DDNAME("brwsdd")"
/* Browse the VIO dataset                                            */
        call ispwrap "BROWSE DATAID("dataid")"
/* FREE and DELETE the VIO dataset                                   */
        call ispwrap "LMFREE DATAID("dataid")"
        call tsotrap "FREE F("brwsdd")"
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/* 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
/* ISPFLIB  - Generate a Dynamic ISPF object                         */
/* ISPFMEM  - The Dynamic ISPF Member                                */
/* ISPFDD   - The ISPF DDNAME                                        */
ispflib: module = 'ISPFLIB'
          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 ispfmem ispfdd
/* Allocate and load the Dynamic ISPF object                         */
          ispfddn = tempmem(ispfmem)
          interpret 'drop' ispfmem'.'
```

```
/* LIBDEF the ISPFLIB DD                                              */
          call ispwrap "LIBDEF" ispfdd "LIBRARY ID("ispfddn") STACK"
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return ispfdd ispfddn
/* ISPFFREE - Free a Dynamic ISPF object                              */
/* ISPFDD   - The ISPF DDNAME                                         */
/* ISPFMEM  - The Dynamic ISPF Member                                 */
ispffree: module = 'ISPFFREE'
           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 ispffdd ispffddn
/* LIBDEF FREE and TSO FREE the ISPF resources                        */
           call ispwrap "LIBDEF" ispffdd
           call tsotrap "FREE F("ispffddn") DELETE"
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return Ø
/* EXECLIB  - Generate a Dynamic REXX EXEC                            */
/* EXECMEM  - The REXX EXEC Member                                    */
execlib: module = 'EXECLIB'
          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 execmem
/* Allocate and load the Dynamic exec object                          */
          execddn = tempmem(execmem)
          interpret 'drop' execmem'.'
/* Re-allocate without the member name                                */
          LRC = listdsi(execddn 'FILE')
          call tsotrap "ALLOC F("execddn") DA('"sysdsname"') OLD REUSE"
/* ALTLIB the EXECLIB DD                                              */
          call tsotrap "ALTLIB ACTIVATE APPLICATION(EXEC)",
                       "FILE("execddn") UNCOND"
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return execddn
/* EXECFREE - Free a Dynamic REXX object                              */
/* EXECDDN  - The Dynamic REXX DDNAME                                 */
execfree: module = 'EXECFREE'
           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 execddn
/* ALTLIB DEACT and TSO FREE the exec resources                      */
           call tsotrap "ALTLIB DEACTIVATE APPLICATION(EXEC)"
           call tsotrap "FREE F("execddn") DELETE"
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return Ø
/* PTR      - Pointer to a storage location                          */
/* ARG(1)   - Storage Address                                        */
ptr: return c2d(storage(d2x(arg(1)),4))
/* STG      - Return the data from a storage location                */
/* ARG(1)   - Location                                               */
/* ARG(2)   - Length                                                 */
stg: return storage(d2x(arg(1)),arg(2))
/* 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(ISRZØØØ)"
             end
           else
              say traceline
/* SAY to SYSTSPRT                                                   */
           return
```

*Robert Zenuk*
*Systems Programmer (USA)*

# z/OS 1.6 enhancements – SMF buffers constraint relief

## HOW SMF MANAGED ITS BUFFERS BEFORE Z/OS 1.6

Initially, SMF allocates 8MB in its own high private storage for its buffers.

These buffers are used to store temporary records before SMF asynchronously writes these records to the SMF datasets (SYS1.MAN*x*) on DASD.

If, for any reason, the data in the buffers cannot be copied to the SMF datasets, SMF continues writing to its buffers the data it generates. When the initial allocation of 8MB is filled, SMF increases its buffers in 8MB increments up to a maximum of 128MB. When the 128MB buffer is filled, loss of SMF data will result.

In huge MVS configurations, when an RMF interval occurs, it generates 'spikes' in SMF activity.

During these SMF 'spikes' several messages can be issued:

- The IEE986E message is issued when the allocation of buffers in the SMF address space exceeds the 25% warning level (this default cannot be changed before z/OS 1.6). The IEE986E message is an indication that SMF is producing records faster than it can write them to the SYS1.MAN*x* datasets:

```
*IEE986E SMF HAS USED    25% OF AVAILABLE BUFFER SPACE
*IEE986E SMF HAS USED    31% OF AVAILABLE BUFFER SPACE
*IEE986E SMF HAS USED    37% OF AVAILABLE BUFFER SPACE
```

- When the system reaches the situation where the last buffer is used, then SMF produce the IEE979W message to indicate that no more space is available for buffers:

```
*IEE979W SMF DATA LOST - NO BUFFER SPACE AVAILABLE TIME=12.00.05
```

## Z/OS 1.6 ENHANCEMENTS

z/OS 1.6 provides two new SMFPRM*xx* parmlib parameters to allow customization of:

- The maximum buffer size – BUFSIZMAX
- The buffer usage warning level – BUFUSEWARN.

### BUFSIZMAX parameter

The BUFSIZMAX(nnnnM) parameter specifies the maximum amount of storage that SMF can use for SMF record data buffering.

The BUFSIZMAX value can be defined from a minimum of 128MB to a maximum of 1GB. The default value is 128MB.

### BUFUSEWARN parameter

The BUFUSEWARN (dd) parameter specifies the buffer warning level percentage when SMF will start issuing message IEE986E. The value that can be specified is from 10 to 90 (10% to 90%). The default is 25%.

### SMFPRM*xx* member

The SYS1.PARMLIB(SMFPRM*xx*) member now looks like:

```
ACTIVE
 DSNAME(SYS1.&SYSNAME..MAN1,SYS1.&SYSNAME..MAN2,SYS1.&SYSNAME..MAN3)
BUFUSEWARN(10)
BUFSIZMAX(0512M)
NOPROMPT
DDCONS(NO)
INTVAL(15)
REC(PERM)
MAXDORM(3000)
STATUS(SMF,SYNC)
JWT(0100)
SID(&SYSNAME(1:4))
 SYS(NOTYPE(19,92,99,231),INTERVAL(SMF,SYNC),NOEXITS,DETAIL)
 SUBSYS(JES2,EXITS(IEFU83,IEFUJI,IEFACTRT,IEFUSI,IEFUJV,IEFUTL))
SUBSYS(STC,EXITS(IEFU29))
SUBSYS(TSO,EXITS(IEFUSI))
```

## D SMF,O

The new BUFSIZMAX and BUFUSEWARN parameters that are in effect will be displayed on the result of the D SMF,O command:

```
D SMF,O
IEE967I 15.00.26 SMF PARAMETERS 631
        MEMBER = SMFPRM00
        MULCFUNC - DEFAULT
        BUFSIZMAX(0512M) - PARMLIB
        BUFUSEWARN(10) - PARMLIB
        MEMLIMIT(00000M) - DEFAULT
        LISTDSN - DEFAULT
        DSNAME(SYS1.SMVS.MAN3) - PARMLIB
        DSNAME(SYS1.SMVS.MAN2) - PARMLIB
        DSNAME(SYS1.SMVS.MAN1) - PARMLIB
        ...
```

*Alexandre Goupil*
*Systems Programmer (France)*

SyncSort has announced Release 1.2 of SyncSort for z/OS, a sort/merge/copy utility designed to exploit the advanced facilities of the z/OS operating system and zSeries systems.

The new version is claimed to be 25 percent faster than the previous release. The performance improvements are achieved by enhancements to SyncSort's ZSPACE technique, exploitation of Parallel Access Volume (PAV) technology, and advances in elapsed time performance for many merge and copy applications.

SyncSort's proprietary sorting algorithms and optimization techniques also reduce consumption of CPU time and EXCPs and improve overall system throughput, they claim.

For further information contact:
URL: www.syncsort.com/press/zospress2.htm.

* * *

NEON Systems has announced that its Shadow z/Events now include capabilities for non-invasive capture, enrichment, and publication of events occurring within native VSAM and Adabas mainframe databases.

Shadow z/Events, part of Shadow RTE, provides organizations with a single, common interface for managing the refinement, delivery, and monitoring of business events across multiple mainframe data sources to multiple messaging protocols – all within a zero-latency or real-time execution model.

Shadow z/Events supports DB2, IMS/DB, CICS VSAM, native VSAM, and Adabas.

For further information contact your local IBM representative:
URL: www.neonsys.com/newsroom/press_releases/2005/20050503.asp.

* * *

Computer Associates has announced r11 of its Unicenter Database Management tool for IMS, which can handle facilities introduced with Version 9 of IMS.

New in IMS 9 is online reorganizations of High Availability Large Databases (HALDB). IMS ships with native tools that support this feature, but CA claims the latest release of Unicenter Database Management for IMS lets users more easily support sophisticated reorg scenarios.

R11 also provides support for other IMS 9-specific capabilities, including compression for Data Entry Databases (DEDB) when they are opened by multiple TCBs.

For further information contact:
URL: www3.ca.com/press/PressRelease.aspx?CID=68764.

* * *

Select Business Solutions has announced Version 4.5 of UltraQuest, its Web-based, business intelligence software. The new release has job scheduling facilities.

UltraQuest servers provide end users with Web access to S/390 data sources, including DB2 for z/OS or z/VM, VSAM, IMS, IDMS, Teradata, NOMAD, and QSAM.

Users submit their report requests for batch processing as part of a site's production process. Jobs can be scheduled to run at particular times or days of the week.

For further information contact:
URL: www.selectbs.com/products/ultraquest.htm.