229

MVS update

October 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.

Virtual Volume Dataset information

In this article we will discuss some of the basic aspects of VSAM Virtual Volume Datasets (VVDSs).

VVDSs are either allocated explicitly by storage administrators or implicitly by the catalog management component of MVS when there is a requirement for such a dataset. However, whether allocated explicitly or implicitly, it must have a dataset name that is prefixed SYS1.VVDS.Vvolser, where volser is the volume serial number of the volume on which it will exist. It must be a VSAM type dataset, and it will have a control interval size of 4096. The only other attribute that is not system assigned is the actual space it will occupy on the volume where it is to be allocated.

At most sites, people prefer to explicitly allocate their VVDSs, usually to ensure that the space allocated to them is adequate to handle the datasets that will be defined within them and also so that they can allocate them at a specific place on the disk. The latter used to be important, but, to be honest, it is no longer something that should be of concern. The former, however, is valid and where you expect to have many datasets, sizing the VVDS appropriately can be advantageous.

Implicit allocation of a VVDS does not occur until the first dataset requiring a VVDS entry is created. If a volume does not already have a VVDS allocated, catalog management will create one when the first VSAM dataset is allocated on the volume in question, or, if it is an SMS-managed volume, when the first VSAM or non-VSAM dataset is allocated. Catalog management assigns space attributes of TRK(10 10) to the VVDS. This could be deemed small, but will allow for over 2500 datasets to be allocated and the VVDS can also extend into secondary extents. Because it could in essence have up to 123 extents, that allows for quite a lot of datasets to be entered into it. Many people believe that if a VVDS is in more than one extent a performance problem will exist, but this is

not the case. If, however, you still want to allocate a VVDS explicitly, the JCL shown below can be utilized:

```
//JXB7884S   JOB  'J.BRADLEY',CLASS=A,MSGCLASS=H,MSGLEVEL=(1,1)
//STEP1      EXEC PGM=IDCAMS
//SYSPRINT   DD   SYSOUT=*
//SYSIN      DD   *
    DEFINE CLUSTER(NAME(SYS1.VVDS.VSYS001) -
     NONINDEXED -
     VOL(SYS001) -
     CYL(5 1))
/*
//
```

In the example the VVDS being explicitly defined is to be allocated on a volume named SYS001.

In today's data centre, most people are running 3390-type disks, and most devices are actually emulating 3390s and have a specific back-up environment of RAID and cache behind them. This negates any benefits that could be obtained by specifically allocating the VVDS on a specific track/cylinder of the disk in question.

Because the VVDS must have a control interval size of 4096 bytes you can estimate that 20 VVR/NVR records will fit into each control interval. This assumes that an average NVR has 125 bytes of data and an average VVR has 400 bytes of data. Each track on a 3390 disk can hold up to 12 control intervals of 4096 bytes. This means you will get about 240 records per track – which means, if you use the implicit declaration of 10 tracks, the primary extent of the VVDS will have approximately 2400 records in it. The VVDS will acquire secondary extents if that fills and can store a further 2400 records, and so on up to 123 extents. If you think a volume will contain more records than can be held in the maximum 123 extent VVDS, then you should explicitly allocate the VVDS and size it accordingly.

Remember as well that even though the VVDS might be in several extents on RAID type disks, this should not be an issue. In many cases the VVDS itself is held in virtual storage, so actual disk access does not occur that often.

The next item of confusion associated with VVDSs is related to their high-level qualifier. Because a VVDS has a high level qualifier of SYS1, where should it be catalogued? In the master catalog of course! Well, this is not necessarily the case. When the VVDS is needed, it is located without looking in the master catalog. The rule that is detailed by IBM is that a VVDS should only be catalogued in those catalogs to which it usefully relates. So, unless you have system datasets or SMS-managed system volumes, the VVDS need not be allocated in the master catalog.

If you look at a VVDS you may find that it is catalogued in a number of user catalogs or not catalogued at all. It is the only VSAM dataset that can act in this way and still be fully functional in the system. VVDS cataloging is really only for human benefit because the system will handle an uncatalogued VVDS without a problem. The system uses the volume serial number that is part of the VVDS name to locate it. If the system does a dataset locate, the dataset's catalog record will have a volume serial number associated with it. This automatically tells the system the name of the VVDS it will need to search. Catalog management will then open the VTOC on that volume and locate the Format 1 DSCB for the VVDS. It will then read part of the VVDS before a true open of the dataset takes place and find the VVR of the VVDS (which is stored in the VVDS itself) so it can then do a true open of it. The VVDS's self-describing VVR is always in the second record position inside the VVDS, so once the physical VVDS location is known from the VTOC the system routines can easily read this record and perform a true open of the VVDS.

A VVDS should never be defined with a catalog parameter. Any datasets allocated on a volume will place a VVDS catalog entry in their catalog if one does not already exist. However, if that catalog entry is then removed/lost, datasets can still be located. You can recover a VVDS entry using CATALOG REDEFINE, but it is not necessary.

A catalog does have a relationship to the VVDSs at a site. A

special record known as the Virtual Volume Catalog Record (VVCR) will always be the first record in the VVDS. It contains the name of each catalog that the VVDS relates to. When the first dataset on a volume is catalogued, a VVCR is built and the name of the catalog it is being catalogued into is placed in the record. You can then use IDCAMS PRINT to extract this information to find out which catalogs the VVDS relates to. Before being shown how to do this, you do need to have one other piece of information. The VVCR can store up to36 catalog names. If more than this are entered in it, a new record known as a BCS name extension record is created. This is referred to as a VVCN record and is chained from the RBA in the VVCR. If a VVCN exists, it is found by looking at the RBA address value at offset 12 (Word 4 would contain the address) from the front of the VVCR. You would need to extract this value by dumping the VVCR and then running a second command to print the VVCN.

To dump the VVCR use the JCL shown below:

```
//JXB7884S  JOB (7884),'7884,J.BRADLEY',CLASS=K
//*
//*      ************************************************************
//*      *                                                          *
//*      * DUMP A VVDS VVCR                                        *
//*      *                                                          *
//*      ************************************************************
//*
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=*
//SYSIN     DD   *
  PRINT INDATASET(SYS1.VVDS.VSYSØØ1) COUNT(1)
/*
```

The following JCL could be used if you needed to dump a VVCN:

```
//JXB7884S  JOB  (7884),'7884,J.BRADLEY',CLASS=K
//*
//*      ************************************************************
//*      *                                                          *
//*      * DUMP A VVDS VVCN                                        *
//*      *                                                          *
//*      ************************************************************
//*
```

```
//STEP1     EXEC PGM=IDCAMS
//SYSPRINT  DD   SYSOUT=*
//SYSIN     DD   *
  PRINT INDATASET(SYS1.VVDS.VSYSØØ1) FROMADDRESS(rba) COUNT(1)
/*
```

You would have to replace *rba* with the RBA extracted from offset 12.

You can use the information gleaned from these records to tidy up unwanted VVDSs or to build IDCAMS Diagnose commands and then to fix catalog errors. In the ever-changing structure of catalogs and VVDSs, this task can be cumbersome and is of real value normally only if problems occur, rather than in general day-to-day maintenance.

However, understanding the information presented here and also understanding how to extract information from VVDSs is an essential part of the systems programmer's/storage manager's toolkit.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)

Display differences between two datasets

PURPOSE AND APPLICATION

This tool is designed to pick out the differences (inserted/deleted lines) between two datasets that were identified by the COMP macro after its execution to compare the datasets.

This is of use to a person who is interested in finding out what the differences are without having to read the entire contents. It also facilitates faster navigation because the common (unchanged) lines are hidden from the user; only the inserted/deleted lines are shown.

There are three valid parameters that can be passed to this REXX. They are:

- A – to show only lines that are added.
- D – to show only lines that are deleted.
- B – to show both added and deleted lines. This is the default if the parameter is not supplied.

SOURCE CODE

```
/*REXX*/
"ISREDIT MACRO (p1)"
upper p1
if p1 = 'A' | p1 = 'B' | p1 = 'D' | strip(p1) = '' then
   nop
else
  do
    zedsmsg = 'invalid parm'
    Zedlmsg = "Parm must be one of (A,D,B). 'A' for Added "||,
              "lines. 'D' for Deleted lines. 'B' for both."
    "ispexec setmsg msg(isrz001)"
    Return
  end
if p1 = '' then
  do
    p1 = 'B'
    zedsmsg = 'Default parm assumed'
    Zedlmsg = "Parm must be one of (A,D,B). 'A' for Added "||,
              "lines. 'D' for Deleted lines. 'B' for both(default)."
    "ispexec setmsg msg(isrz001)"
  end
"isredit (lastline) = LINENUM .ZLAST"
do i = 1 to lastline
"isredit (xstat) = xstatus "i
if p1 = 'B' | p1 = 'A' then
  if xstat = 'NX' then
     do
       a = ''
       "isredit (a,b) = label "i
       if strip(a) = '' then
         "isredit xstatus "i" = X"
     end
if p1 = 'D' then
    "isredit xstatus "i" = X"
end
if p1 = 'A' then
  "isredit reset special"
```

SET-UP

Copy the source code above into a member of your REXX PDS (which is also allocated to your profile whenever you log-on) and name that member GETDIFF.

EXECUTION

Open your first dataset in edit or view mode.

For example:

```
First dataset
Command ===>                                                  S
****** ***************************** Top of Data **************
000001 first line here
000002 second line here
000003 third line here
000004 fourth line here
000005 fifth line here
000006 sixth line here
000007 tenth line here
****** **************************** Bottom of Data ************
```

Assuming the second dataset looks like the one below:

```
****** ***************************** Top of Data **************
000001 1st   line here
000002 second line here
000003 3rd   line here
000004 fourth line here
000005 5th   line here
000006 sixth line here
000007 7th line here
000008 eighth line here
****** **************************** Bottom of Data ************
```

COMPare the first dataset with the second using the COMP macro:

```
COMP X 'dataset-name'
COMP  'dataset-name'
```

This will result in the following:

```
****** ***************************** Top of Data *********
====== 1st   line here
.OAAAA first line here
000002 second line here
```

```
====== 3rd   line here
.0AAAB third line here
000004 fourth line here
====== 5th   line here
.0AAAC fifth line here
000006 sixth line here
====== 7th line here
====== eighth line here
.0AAAD tenth line here
****** **************************** Bottom of Data *******
```

In the original output, lines that are not changed or are common to the first and second datasets are shown in green.

Lines that are found in the first dataset but not in the second dataset (ie added/new lines when compared with the second dataset) are shown in blue.

Lines that are not found in the first but are found in the second dataset (ie deleted lines when compared with the second dataset) are shown in red.

Now execute the GETDIFF command on this dataset (the dataset highlighting the differences):

```
EDIT       PROD.REXX.TEST(FIRSTDS) - 01.00          Default parm assumed
Command ===>                                            Scroll ===> CSR
****** ************************** Top of Data ***************************
====== 1st   line here
.0AAAA first line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
====== 3rd   line here
.0AAAB third line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
====== 5th   line here
.0AAAC fifth line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
====== 7th line here
====== eighth line here
.0AAAD tenth line here
****** ************************** Bottom of Data ************************
```

Since there is no parameter supplied to this macro, the default parameter was assumed, and hence both added and deleted lines are extracted. Other lines that are not changed or are common to both datasets are hidden from the user. The same results can be seen by executing the macro with the ‘B’

parameter, ie:

```
Getdiff B
```

Watch out for the message in the top right-hand corner when you use the default parameter.

To extract only the lines that were added, use the parameter ‘A’ with the macro GETDIFF after executing the COMP command. This will result in the following:

```
Command ===>
****** ***************************** Top of Data ******
.0AAAA first line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
.0AAAB third line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
.0AAAC fifth line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  -
.0AAAD tenth line here
****** **************************** Bottom of Data ****
```

To extract only the deleted lines, use the parameter ‘D’ with the macro GETDIFF after executing the COMP command:

```
Command ===>
****** ***************************** Top of Data ******
====== 1st   line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  2 Line(s) not Displayed
====== 3rd   line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  2 Line(s) not Displayed
====== 5th   line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  2 Line(s) not Displayed
====== 7th line here
====== eigth line here
- - -  -  -  -  -  -  -  -  -  -  -  -  -  -  -  1 Line(s) not Displayed
**************************** Bottom of Data ****
```

Using any other parameter with GETDIFF will result in the error message shown below. Pressing F1 (F1 should be mapped to HELP) or issuing the HELP command will result in the detailed error message:

```
EDIT       PROD.REXX.TEST(FIRSTDS) - Ø1.ØØ                invalid parm
Command ===> GETDIFF X                                  Scroll ===> CSR
****** ************************* Top of Data ***************************
====== 1st   line here
.0AAAA first line here
```

```
000002 second line here
====== 3rd   line here
.OAAAB third line here
000004 fourth line here
====== 5th   line here
.OAAAC fifth line here
000006 sixth line here
====== 7th line here
====== eighth line here
.OAAAD tenth line here
**************************** Bottom of Data ****************************
 Parm must be one of (A,D,B). 'A' for Added lines. 'D' for Deleted
lines. 'B' for both.
```

Suresh Kumar Murugesan
Systems Analyst
Cognizant Technology Solutions (USA)

Dylakor

Dylakor is a powerful interpreted language that doesn't need to be compiled and is used mainly for generating reports. The language is very similar to Easytrieve. Two Dylakor utility programs are available, DYL260 and DYL280. Originally they were from Sterling Software and later they were taken over by Computer Associates.

They are available with the names Vision:Sixty and Vision:Results/Vision:Eighty. The utility program names remain the same – DYL260 and DYL280.

The current CA release is Vision:Results 5.0.

Key notes:

- The program is written into the SYSIN DD of the DYL280 program.
- Freeform text statements can begin from column 1 to any subsequent column.

- Comments can be added by coding an asterisk (*) or semicolon (;) in column 1.
- STOP is used to terminate the program at any point in time.
- FIN is an optional statement to mark the end of the Dylakor program. It is always the last word in the program. No statements are processed after the FIN statement.
- User-defined labels (label name followed by a colon ':') can be used for structured programming using paragraphs.
- In Release 5.0, several new features are available including DB2 support and Language Environment (LE) enhancement to call subroutines written in other languages like COBOL, PL/I, C, etc.

Sample program structure is shown below:

```
OPTION PRINTERR STRUCTURED2
OPTION QLF
REPORT 8Ø WIDE 6Ø LONG ASA
*************************************************
* HEADER BLOCK OR PROGRAM DESCRIPTION
*************************************************
FILE INFILE   INPUT  FB 451   STATUS  ISTAT
     IN_REC             451    1  CH
     IN_REC_ID            2    1  CH
     IN_REC_MERCH         9   7Ø  CH
     IN_REC_AMT           5   25  PD
FILE OUTFILE  OUTPUT FROM OUTFILE  FB 451  STATUS OSTAT
     OUT_REC            451    1  CH
*
WORKAREA
     SEC_CODE             4       CH
*************************************************
* MAIN LOGIC
*************************************************
START:
ON ONE
     MOVE DYLPARM TO SEC_CODE
ENDONE
SORT INFILE USING IN_REC_ID A IN_REC_MERCH A
IF IN_REC_ID NE 'D1'
   REJECT
ENDIF
```

```
OUT_REC = IN_REC
WRITE OUTFILE
CONTROL IN_REC_MERCH
ON CHANGE IN IN_REC_MERCH
   LIST IN_REC_MERCH ' TOTAL ' SUM IN_REC_AMT
ON FINAL
   LIST ' TOTAL ' AT IN_REC_MERCH SUM IN_REC_AMT
************************************************
* TITLE LINES
************************************************
T1 ' --- PTSLOG REPORT --- ' WITH 1 AFTER
T1+1  DYLDATE
T1+60 DYLPAGE
T2 ' FOR SECURITY CODE ' WITH 2 AFTER
T2+55 SEC_CODE
FIN
```

Sample JCL is shown below:

```
//TSOID JOB (6563),'DYLAKOR',REGION=9M,NOTIFY=&SYSUID,
//   CLASS=F,MSGCLASS=X,TIME=1440
//JOBLIB    DD DSN=IOCP00P.PB.CESPROD.LOADLIB,
//             DISP=(SHR,KEEP,KEEP)
//PSTY0010 EXEC PGM=DYL280,
//             PARM='UA=020'                              user parameter
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYS280R   DD SYSOUT=*                                  default report
//SYSCOPY   DD DSN=PRODENDV.PB.SYSP.COPYLIB,            copybook library
//             DISP=(SHR,KEEP,KEEP)
//ITMPLOG   DD DSN=ISDM00D.CTS.B1152004.TMPLOG,
//             DISP=(SHR,KEEP,KEEP)
//OTMPLOG   DD DSN=APTA00Q.TB.APTD2045.PTSLOG3,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(20,20),RLSE),
//             DCB=(RECFM=FB,LRECL=451,BLKSIZE=0)
//SYS004    DD DSN=APTA00Q.TB.APTD2045.WORK2802,  mandatory work dataset
//             DISP=(NEW,DELETE,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(55,5)),
//             DCB=(SYS1.MDSCB,RECFM=FB,LRECL=80)
//SYSIN     DD DSN=IOCP00P.PB.CESPROD.CARDLIB(KPTCTMPL), Dylakor program
//             DISP=(SHR,KEEP,KEEP)
```

OPTION statements:

- OPTION STRUCTURED – allows you to use structured programming statements like READ, DOWHILE, CASE, and PERFORM.

- OPTION STRUCTURED2 – same as STRUCTURED, but with some additional features like the REJECT verb.
- OPTION QLF – allows you to use duplicate data names within the same REPORT step. Duplicate variables will be identified and prefixed with the file name they are referenced in.
- OPTION VERIFY – checks the syntax without executing it. It can be used when executing a program for the first time.
- The option PRINTERR can be used along with STRUCTURED, though it is the default. It will print errors in SYSPRINT.

A REPORT statement looks like:

```
REPORTnnn 999 WIDE 999 LONG ASA SYS280R#
```

where:

- *999 WIDE* is the width of the printed report.
- *999 LONG* is the number of lines in a single page.
- *ASA* – this parameter is required for the correct formatting of reports by mainframe printers.
- *nnn* is the sequence number of the REPORT statement.
- *SYS280R#* is the report DDname in the JCL. If there is only one report there is no need to give this parameter. It will go to the default SYS280R dataset.
- # is the report identifier suffix for multiple reports, eg SYS280R1, SYS280RS, etc.

An example multiple report is shown below:

```
REPORT1 8Ø WIDE 6Ø LONG SYS28ØR1
FILE INFILE   INPUT  FB 451 RETAIN  STATUS  ISTAT
     IN_REC             451    1  CH
     IN_REC_ID            2    1  CH
     IN_REC_MERCH         9   7Ø  CH
     IN_REC_AMT           5   25  PD
```

```
FILE OUTFILE  OUTPUT FROM OUTFILE  FB 451  STATUS OSTAT
     OUT_REC            451    1  CH
WORKAREA
     SEC_CODE             4       CH
MAINLINE:
ON ONE
---
ENDONE
---
---
ON FINAL
   LIST ' TOTAL ' AT IN_REC_MERCH SUM IN_REC_AMT
T1 ' --- PTSLOG REPORT --- ' WITH 1 AFTER
T1+1  DYLDATE
---
---
************************************
REPORT2 8Ø WIDE 6Ø LONG SYS28ØR2
USE INFILE
WORKAREA
     SEC_CODE1            4       CH
START:
ON ONE
---
ENDONE
---
---
ON FINAL
   LIST ' TOTAL ' AT IN_REC_MERCH SUM IN_REC_AMT
T1 ' --- PTSLOG REPORT --- ' WITH 1 AFTER
T1+1  DYLDATE
---
---
FIN
```

Notes:

- Different suffixes are given to distinguish between reports, ie REPORT1, REPORT2, etc.
- Each report section can have its own set of file, work area, processing logic, and title statements.
- If more than one REPORT section is using the same file, there is no need to declare it in every report section; write USE <file name>. Also the file handling (READ, SORT, etc) can be kept in the first REPORT section only. But the RETAIN option must be coded in the file declaration to

make it available for the next REPORT.

Sample DYL280 step:

```
//PSTY0140 EXEC PGM=DYL280,
//ITRNBILL  DD DSN=KMGA00P.TB.SRTD.BILLING,
//             DISP=(SHR,KEEP,KEEP)
//SYS004    DD DSN=KMGA00P.TB.KMGA0065.WORK001,
//             DISP=(NEW,CATLG,DELETE),
//             UNIT=SYSDA,
//             SPACE=(CYL,(15,10),RLSE)
//SYS280R1  DD SYSOUT=*
//SYS280R2  DD SYSOUT=*
//SYSOUT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//SYSIN     DD DSN=IOCP00P.PB.CESPROD.CARDLIB(KMGARPT1),
//             DISP=(SHR,KEEP,KEEP)
```

Shamik Mondal
Application Programmer (India)

Please note that the correct contact address for Xephon Inc is PO Box 550547, Dallas, TX 75355, USA. The phone number is (214) 340 5690, the fax number is (214) 341 7081, and the e-mail address to use is info@xephon.com.

ISPF DM file tailoring, permanent tables, and more

In a previous article (see 'A basic suite of table subroutines for ISPF DM' in *MVS Update* issue 225, June 2005), I presented a range of REXX subroutines designed to manipulate temporary ISPF tables. In this article, subroutines to carry out similar but more complex operations for permanent tables are given. Depending on the situation, either approach may be the more appropriate. Additionally, an enhanced display subroutine is offered, the ability to scan tables is added, and a mechanism to offload a table to a sequential file is provided. Examining this code might be a fruitful way of extending your ISPF skills. It introduces TBSKIP and TBSCAN, File Tailoring (FT), Library Management (LM), LIBDEF, the use of variable 'pools', and message members. I always advise the use of a strong naming standard and have used one here – XEPH=dialog, XEPHPnn=panel, XEPHSnn=skeleton, XEPHMnn=message, and XEPHTnn=table, etc.

To begin, you must create a small partitioned dataset for the tables and put various members in the panel, skeleton, and message libraries that are defined in your log-on procedure. The following six steps must be satisfactorily completed for successful execution:

1 Create a small catalogued PDS (recfm=FB, lrecl=80, tracks=5, dir=5) to hold the tables. If convenient, use a PDSE, but this is not essential. Use any appropriate name you like. You will be asked to provide the name in step 5.

2 Add the following two ISPF panel definitions to a library in your ISPPLIB log-on concatenation:

   - The execution panel, XEPHP01. Note this panel differs from the previous example.

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

```
)BODY EXPAND(\\)
'Command =>_ZCMD                          'Scroll =>_AMT '
'O Key------------ Name------------
'\-\
)MODEL ROWS(&DISMO)
_O{K1              {N1              '
)INIT
 .HELP = XEPHPØ1H
 VGET (DISMO) SHARED
 &ZTDMARK = ''
 IF (&ACC='EDIT') .ATTRCHAR({)='TYPE(NEF) CAPS(OFF)' /* edit */

)END
```

- The related help panel XEPHP01H:

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

)BODY WINDOW(72,19) EXPAND(\\)
'Command =>_ZCMD
'
'To modify data, simply overtype the row(s). Where keys are edited, make
'sure that the new value is unique or the update will not be permitted.
'
'\-\ Line operators. One or more may be issued simultaneously. \-\
'D = Delete selected row (even if key is changed to indicate another).
'C = Copy row. Note: change key to unique value or command will fail.
'
'\-\ Primary Commands \-\
'A        = Add row. Default values will be inserted and should be
'           subsequently overtyped.
'O        = Output table to sequential file.
'F <text> = Hide all rows not containing <text>
'F        = Restore hidden rows.
'C        = Cancel edit session without saving changes.
'X        = Terminate edit session, saving changes.
'
\ \Hit PF3 to return
)END
```

3 Add the following file tailoring skeleton member to a library in your ISPSLIB log-on concatenation, under the name XEPHS01. Be sure to edit the job card to match your own site specification and check whether the generated output

dataset name is acceptable to your security product and SMS as well:

```
)CM Skeleton for xephon demo dialogue.
)TB 12
//&VUSR.X!JOB   SS,'g17Ø',CLASS=X,MSGCLASS=X,MSGLEVEL=(1,1),REGION=4M
//*  Skeleton=&SKEL  Table=&TAB
//*
)CM -- delete output dataset if already existing (ok if nonexistent)
//DELETE   EXEC PGM=IEFBR14
//DELDD    DD   DISP=(MOD,DELETE,DELETE),UNIT=339Ø,SPACE=(TRK,(Ø)),
//         DSN=&VUSR..&TAB..J&VDAT..TABULAR.DATA
)CM -- generate iebgener input data from table
//STEP1    EXEC PGM=IEBGENER
//SYSIN    DD   DUMMY
//SYSUT2   DD   DISP=(NEW,CATLG,DELETE),UNIT=339Ø,LRECL=8Ø,
//         SPACE=(TRK,(25,25),RLSE),
//         DSN=&VUSR..&TAB..J&VDAT..TABULAR.DATA
//SYSPRINT DD   SYSOUT=*
//SYSUT1   DD   *
)TB 2Ø 4Ø
 !  Keys           !  Data
 !--------------- !---------------
)BLANK
)DOT &TAB
 Data !&K1 !&N1
)ENDDOT
/*
```

Important: if you are running z/OS 1.6 you may wish to replace the line:

```
)DOT &TAB
```

(fourth line from end of skeleton) with:

```
)DOT &TAB &DISMO
```

This new feature enables a table that is being scanned (as the result of an 'F' command) to be copied out as it is currently displayed, ie as a subset. This option was not available before z/OS 1.6 (previously the DOT operation always processed the whole table whether scanning or skipping).

4 Add the following message member definition to a library in your ISPMLIB log-on concatenation, under the name XEPHM00:

```
XEPHMØØØ                                                .W=LN .ALARM=NO
'No rows containing <&FSTR.> were found. Display is unmodified'

XEPHMØØ1                                                .W=LN .ALARM=NO
'Existing table &TAB opened for processing'

XEPHMØØ2                                                .W=LN .ALARM=NO
'New table &TAB open. Use ''A'' primary command to add first row'

XEPHMØØ3                                                .W=LN .ALARM=NO
'Table &TAB has been closed in &MODE mode, r/c=&RC'

XEPHMØØ4 'Reserved'                                     .W=LN .ALARM=NO
'This is message xephmØØ4 and should not appear'
```

5 Add the following EXEC member to a library in your SYSEXEC or SYSPROC concatenation, under the name XEPH (or any other you prefer). Provide the name of your table library in the fourth line. Use regular TSO rules (fully qualified with apostrophes or otherwise).

```
/* rexx */
TRACE("O")
ADDRESS ISPEXEC
"CONTROL ERRORS RETURN"
TABLIB="'MY.XEPH.TABLE.DATA' "       /* Give your table library name  */

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

ARG TAB                              /* get table name if any         */
IF (TAB="") THEN TAB="XEPHTØ1"       /* if none, call it xephtØ1      */
PAN="XEPHPØ1"                        /* panel name is xephpØ1         */
AC=LIBDEFINE(TABLIB)                 /* go acquire my table dataset   */
IF (AC=Ø) THEN DO                    /* OK? open (or create and open) */
 AC=OPENTAB(TAB)                     /* the table                     */
 IF (AC=Ø) THEN DO                   /* OK?                           */
  AC=DISPTAB(TAB,PAN,"EDIT")         /* display, force access to edit */
  CALL CLOSETAB TAB,AC               /* close table with rc from disp */
  END
 AC=LIBFREE()                        /* surrender my table dataset    */
 END
EXIT AC

/* --- LIBDEF's for table library ---------------------------------- */

LIBDEFINE: PROCEDURE                 /* no variables exposed          */
 ARG TABLIB                          /* table name required           */
 "LIBDEF ISPTLIB DATASET ID("||,     /* define the INPUT dataset,     */
            TABLIB||") STACK"        /* from where the tables come.   */
```

```
 AC=RC
 IF (AC=Ø) THEN DO                    /* OK? the define the OUTPUT     */
  "LIBDEF ISPTABL DATASET ID("||,      /* dataset, same as input, to    */
             TABLIB||") STACK"         /* where the tables go.          */
  AC=RC                               /* OK? Everything is cool. So    */
  IF (AC=Ø) THEN RETURN Ø             /* leave at this point...        */
  SAY "Libdef TABL? r/c=" AC          /* Erk. output lib error.        */
  "LIBDEF ISPTLIB"                    /* Backout-free the input lib.   */
  END
 ELSE SAY "Libdef TLIB? r/c=" AC      /* Erk. input lib error          */
 RETURN AC                            /* Give user unhappy news        */

/* --- LIBDEF free's ------------------------------------------------ */

LIBFREE: PROCEDURE                    /* We must liberate the input    */
 "LIBDEF ISPTLIB"                     /* and output table datasets.    */
 IF (RC>Ø) THEN                       /* Note that whatever the result */
    SAY "Free libdef TLIB? r/c=" AC   /* we send back rc Ø 'cos there  */
 "LIBDEF ISPTABL"                     /* is no point in doing anything */
 IF (RC>Ø) THEN                       /* else...                       */
    SAY "Free libdef TABL? r/c=" AC
 RETURN Ø

/* --- Open table (create if nonexistent) --------------------------- */

OPENTAB: PROCEDURE                    /* no variables exposed          */
 ARG TAB                              /* Table name required           */
 "TBEND" TAB                          /* false close - just in case    */
 "TBOPEN" TAB "WRITE"                 /* now we try and open it        */
 AC=RC
 IF (AC=Ø) THEN                       /* opened ok? (existing table)   */
     "SETMSG MSG(XEPHMØØ1) COND"      /* tell user                     */
 IF (AC=8) THEN DO                    /* nonexistent? we must try and  */
  "TBCREATE" TAB "KEYS(K1)",          /* create it instead             */
     "NAMES(N1) WRITE"
  AC=RC
  "SETMSG MSG(XEPHMØØ2) COND"         /* tell user about new table     */
  END
 IF (AC=Ø) THEN DO                    /* Do we have an open table?     */
  "TBSORT" TAB "FIELDS(K1,C,A)"       /* Sorted                        */
  AC=RC                               /*                               */
  IF (AC>Ø) THEN SAY "Sort? r/c=" AC  /* or not                        */
  END                                 /*                               */
 ELSE SAY "Create? r/c=" AC           /* or not created                */
 RETURN AC                            /* Ø=OK, anything else=not OK    */

/* --- Close table subroutine --------------------------------------- */

CLOSETAB: PROCEDURE                   /* no variables exposed          */
 ARG TAB,AC                           /* table? rc from display?       */
```

```
 IF (AC=24) THEN MODE="TBEND"        /* finish without saving         */
 ELSE MODE="TBCLOSE"                 /* finish with save              */
 MODE TAB                            /* close table as per mode       */
 AC=RC
 "SETMSG MSG(XEPHMØØ3) COND"         /* Tell user how it went         */
 RETURN AC

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

DISPTAB: PROCEDURE                   /* no variables exposed          */
 ARG TAB,PAN,ACC                     /* table panel and access mode   */
 O=""
 "TBTOP" TAB                         /* go to top (should be anyway)  */
 AC=Ø
 OLDPOS=Ø
 DISMO="ALL"                         /* show all rows to begin        */
 "VPUT (ACC DISMO) SHARED"           /* save in shared pool           */
 DO WHILE AC<8                       /* here we go...                 */
  "TBDISPL" TAB "PANEL("||PAN||")",  /* display table                 */
   "ROWID(OLDROW) POSITION(OLDPOS)",
   "CSRROW("||OLDPOS||") AUTOSEL(NO)"
  AC=RC
  IF AC>8 THEN RETURN 16             /* Display error? leave r/c 16   */
  POSN=Ø                             /* default position marker       */

           /*** do the line operators first ***/

  DO WHILE ZTDSELS>Ø                 /* Line operator(s)?             */
   "CONTROL DISPLAY SAVE"            /* Save the world                */
   O=TRANSLATE(O)                    /* translate lineop to upper     */
   BC=Ø                              /* prime BC                      */
   SELECT

    WHEN (ACC="BROWSE") THEN NOP     /* No lineops in browse mode     */

    WHEN (O=" ") THEN DO             /* Line update?                  */
     "TBPUT" TAB "ORDER"             /* Save the modified line        */
     BC=RC
     IF (BC>Ø) THEN DO               /* not ok? did we edit a key?    */
      "TBADD" TAB "ORDER"            /* try adding the row then       */
      BC=RC
      IF (BC=Ø) THEN DO              /* ok? it was a key edit so we   */
       "TBSKIP" TAB "ROW("||,        /* MUST skip to the old row and  */
          OLDROW||")"
       "TBDELETE" TAB                /* delete it                     */
       END
      ELSE SAY "Update? r/c=" BC     /* otherwise yuk                 */
      END
     O=""
     END
```

```
  WHEN (O="D") THEN DO              /* Delete row? Can't just delete */
   "TBSKIP" TAB "ROW("||,           /* cos user might have edited a  */
      OLDROW||")"                   /* key so we skip to this row    */
   "TBDELETE" TAB                   /* instead and delete it         */
   IF (RC>Ø) THEN
      SAY "Delete? r/c="*RC         /* Yuk?                          */
   O=""
   END

  WHEN (O="C") THEN DO              /* Copy? User should change the  */
   "TBADD" TAB "ORDER"              /* key (if any) before doing it  */
   IF (RC>Ø) THEN                   /* Else the copy will fail       */
      SAY "Copy row? r/c=" RC
   O=""
   END

  OTHERWISE NOP                     /* No more line operators        */
  END

 "CONTROL DISPLAY RESTORE"          /* Restore the world             */
 O=""                               /* flush that lineop field       */
 IF ((ZTDSELS>1)&(BC=Ø)) THEN       /* More than one lineop given?   */
  "TBDISPL" TAB "ROWID(OLDROW)",    /* display again, but without    */
     "POSITION(OLDPOS)",            /* the panel name, to get the    */
   "CSRROW("||OLDPOS||,             /* next lineop                   */
    ") AUTOSEL(NO)"
 ELSE ZTDSELS=Ø                     /* else set lineops to 'done'    */
 END

         /*** now check primary commands ***/

IF ZCMD<>"" THEN DO                 /* Any primary command?          */
 SELECT

  WHEN (ZCMD="O") THEN DO           /* Output request?               */
   CALL OUTSUB TAB,"XEPHSØ1"        /* Go do it                      */
   POSN=Ø                           /* Ø=Reposition display to top   */
   END

  WHEN (WORDPOS(SUBWORD(ZCMD,1,1), /* Is this a 'F <text>' cmd?     */
           ,"F f")>Ø) THEN DO
   FSTR=TRANSLATE(SUBWORD(ZCMD,2)) /* translate the <text> bit to   */
   CALL FIDSUB TAB,FSTR             /* upper and call Find sub       */
   POSN=1                           /* position result to top        */
   END

  WHEN (ZCMD="X") THEN AC=8         /* Exit? Easy. Set RC to 8       */

  WHEN (ACC="BROWSE") THEN NOP      /* No more for Browse mode       */
```

```
    WHEN (ZCMD="C") THEN AC=24       /* Cancel? Out we go...          */

    WHEN (ZCMD="A") THEN DO          /* Add row?                      */
     "TBVCLEAR" TAB                  /* clear variables               */
     K1=TIME("L")                    /* quasi random value for key    */
     N1="Added"                      /* static value for name field   */
     "TBADD" TAB "ORDER"             /* add the row                   */
     IF (RC>Ø) THEN                  /* OK?  Were we too quick? If so */
        SAY "Add? r/c=" RC           /* key value will duplicate      */
     POSN=2                          /* 2=do not reposition display   */
     END

    OTHERWISE NOP                    /* No more primary commands      */
    END
   END
  ZCMD=""                            /* Clear primary command field   */

        /*** finally, honour simultaneous scroll commands ***/

  IF (POSN<2) THEN DO                /* if 2 do nothing, else...      */
   "TBTOP" TAB                       /*  position at top              */
   IF (POSN<1) THEN DO               /*  if 1 do no more else...      */
    "TBSKIP" TAB "NUMBER("ZTDTOP")"  /*  skip to last position        */
    "VGET (ZVERB ZSCROLLN)"          /*  get scroll request           */
    IF ZVERB=UP   THEN "TBSKIP" TAB, /*  going up?                    */
       "NUMBER(-"ZSCROLLN")"         /*  skip up by scroll amount     */
    IF ZVERB=DOWN THEN "TBSKIP" TAB, /*  going down?                  */
       "NUMBER(+"ZSCROLLN")"         /*  skip down by scroll amount   */
    END
   END

  END                                /*  all done. show display again */
 IF (AC <> 24) THEN AC=Ø
 RETURN AC                           /*  finished altogether. ciao.   */

/*--- Output subroutine ------------------------------------------- */

OUTSUB: PROCEDURE                       /* no variables exposed       */
 ARG TAB,SKEL                           /* get tab/skel names         */
 VDAT=DATE("J")                         /* get Julian date...         */
 VUSR=SYSVAR(SYSUID)                    /*    ...and user ID...       */
 "VPUT (TAB SKEL VDAT VUSR)"            /* save all in shared pool    */
 "FTOPEN TEMP"                          /* open temp skeleton file    */
 "FTINCL" SKEL                          /* include my skeleton        */
 "FTCLOSE"                              /* close the skeleton file    */
 "VGET (ZTEMPF ZTEMPN)"                 /* get file name and number   */
 "LMINIT DATAID(DSID) DDNAME("||,       /* alloc temp file            */
  ZTEMPN||") ENQ(SHR)"                  /*                            */
 IF (RC=Ø) THEN DO                      /* ok?                        */
```

```
  "VIEW DATAID("||DSID||")"             /* view the jcl               */
  "LMFREE DATAID("||DSID||")"           /* and free the file          */
  END                                   /*                            */
 /* ADDRESS TSO "SUBMIT '"||ZTEMPF||"'"    optional: auto submit      */
 RETURN Ø                               /* that's all.                */

/*-- Find subroutine ---------------------------------------------- */

FIDSUB: PROCEDURE                       /* no variables exposed       */
 PARSE ARG TAB,FSTR                     /* receive table and string   */

 IF (FSTR="") THEN DO                   /* no find string? it is a    */
  DISMO="ALL"                           /* reset request then         */
  "VPUT (DISMO) SHARED"                 /* set display to 'all' and   */
  RETURN Ø                              /* out we go...               */
  END

 TS=TIME(S); TSX=TS; Y=Ø                /* initialize these           */

 "TBQUERY" TAB "KEYS(KS) NAMES(NS)"     /* get layout of table. keys  */
 IF (RC>Ø) THEN DO                      /* in KS and others in NS.    */
  SAY "Query table? r/c=" RC
  EXIT RC
  END

 FLDS=STRIP(TRANSLATE(KS NS,"  ",       /* Compress layout into a     */
  ,"()"),"B"," ")                       /* string of names in FLDS    */

 "TBTOP" TAB                            /* Go to top of table         */
 AC=Ø
 DO WHILE(AC=Ø)                         /* whether SKIP or SCAN,      */
  "TBSKIP" TAB                          /* always SKIP through here   */
  AC=RC
  IF (AC=Ø) THEN DO                     /* not last row?  then...     */
   INTERPRET "STR=" FLDS                /* turn fields into concat    */
   BC=POS(FSTR,TRANSLATE(STR),1)        /* string, translate to upper */
   IF (BC>Ø) THEN DO                    /* Found? Put 'found ID' in   */
    TSX=TS                              /* extension field and save   */
    "TBPUT" TAB "SAVE(TSX) ORDER"       /* row + extension field.     */
    Y=1                                 /* set 'found something' flag */
    END
   END
  END

 IF (Y=1) THEN DO                       /* Found one or more rows?    */
  "TBVCLEAR" TAB                        /* Yup, clear the fields...   */
  TSX=TS                                /* Fill just the ID field...  */
  "TBSARG" TAB "ARGLIST(TSX)"           /* Set argument and switch    */
  DISMO="SCAN"                          /* from 'all' to 'scan'.      */
  "VPUT (DISMO) SHARED"                 /* Save DISMO in pool.        */
```

```
  END
ELSE "SETMSG MSG(XEPHM000)"            /* No data, nothing changed   */
RETURN 0                               /* back to display subroutine */
```

6 To execute the dialog, issue the command 'XEPH <table name>', for example 'XEPH BORIS'. Table BORIS will be opened (or created and opened if it does not already exist) and displayed. If you do not provide a table name, 'XEPHT01' will be used.

Notes:

- 'EDIT' is forced in the main code's call to the table display. However, the display table subroutine and panel XEPHP01 can display tables in EDIT or BROWSE mode, as the previous article demonstrated.
- To 'output' the table, use the 'O' primary command. The generated JCL is displayed, but the EXEC shows (in a commented line) how to submit it instead – or as well.
- To use 'Find' use the 'F <text>' primary command, for example 'F fred'. All rows not containing 'fred' will be hidden. Use 'F' on its own to restore the display.
- For a new table, use the 'A' add primary command to get the first row into the table. From there, you may use the 'A' command to add more or the 'C' line operator to copy rows.

Deryck Swatman
System Programmer
HM Land Registry (UK)

A 3.4 COPY command

How frequently do you need to allocate a dataset that is just like another dataset? For this I will usually find a dataset to model on, go to ISPF 3.2, hit the *Enter* key to get the attributes,

then type ‘A’ on the command line and change the dataset name to ‘clone’ it. How many times when you clone a dataset do you want to copy all or most of the contents from the ‘model’ into the new dataset? For this I will go to ISPF 3.3 and copy the dataset (sequential or PDS).

Well, I got tired of doing this and created a ‘COPY’ command that can be used as a line command from ISPF 3.4. COPY will present a pop-up with the original dataset name, allow the entry of a new dataset name (by default, it generates a new name by appending ‘.NEW’ to the end of the original name), and allow the use of a member pattern (defaults to ‘*’).

COPY will allocate a dataset just like the original (using ALLOC LIKE) and copy the contents from the source DSN to the target DSN (using LMCOPY or REPRO). COPY will work with PDSs, PDSEs, sequential, and VSAM datasets. It will also copy empty datasets (gives a message to let you know the source was empty).

In all cases (except LOADLIBs) COPY will optimize the BLKSIZE (or CISIZE) of the new dataset. Since LOADLIBs have an LRECL=0, the original BLKSIZE is used. If COPY is used on a z/OS 1.5 system or higher, it uses the new ALIAS feature of LMCOPY to ensure all member aliases are processed correctly.

To use COPY, simply put the COPY EXEC in your SYSEXEC concatenation. Some shops have a non-existent COPY command defined in the ISPTCM (ISPF TSO command table). If this is the case and you cannot change it, enter the COPY command prefixed with a ‘%’ sign, %COPY, and it will work fine. If you can change ISPTCM to eliminate COPY, you can use it without the ‘%’ sign.

COPY was actually written when we de-installed TSO Superset Utilities (from Applied Software**)**. Another requirement was that it supported being called from a REXX EXEC or CLIST. Therefore, COPY can also be used as a called subroutine or a function from other REXX EXECs:

```
call copy 'SYS1.PARMLIB' 'MY.PARMLIB' '*'
COPYRC = copy('SYS1.IPLPARM' 'MY.IPLPARM' 'LOAD*')
```

COPY EXEC

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: Replacement for TSO Superset COPY command                */
/*-------------------------------------------------------------------*/
/* Syntax:  COPY srcdsn targdsn pattern replace                      */
/*-------------------------------------------------------------------*/
/* Parms: SRCDSN     - The Source DSN                                */
/*        TARGDSN    - The Target DSN                                */
/*        PATTERN    - The optional member pattern (default is '*')  */
/*        REPLACE    - If the TARGDSN exists, overwrite (YES or NO)  */
/*                     (default is NO)                               */
/*                                                                   */
/* Notes: Initially designed for use with ISPF 3.4 DSLIST            */
/*        Supports PDS, PDSE, sequential and VSAM datasets           */
/*        REPLACE=YES will delete and define TARGDSN                 */
/*        Empty datasets are just allocated                          */
/*        RECFM=U LOADLIBs are supported                             */
/*        Callable as a subroutine or function                       */
/*        Online use allows space changes                            */
/*        Can run in batch too.                                      */
/*                                                                   */
/* Sample Batch JCL:                                                 */
/*                                                                   */
/* //COPY     EXEC PGM=IKJEFTØ1,PARM='%COPY srcdsn targdsn'         */
/* //SYSEXEC  DD   DSN=your.rexx.dsn,DISP=SHR                       */
/* //ISPPLIB  DD   DSN=your.system.ISPPLIB,DISP=SHR                 */
/* //ISPSLIB  DD   DSN=your.system.ISPSLIB,DISP=SHR                 */
/* //ISPMLIB  DD   DSN=your.system.ISPMLIB,DISP=SHR                 */
/* //ISPTLIB  DD   DDNAME=ISPTABL                                   */
/* //         DD   DSN=your.system.ISPTLIB,DISP=SHR                 */
/* //ISPTABL  DD   LIKE=your.system.ISPTLIB,UNIT=VIO                */
/* //ISPPROF  DD   LIKE=your.system.ISPTLIB,UNIT=VIO                */
/* //ISPLOG   DD   SYSOUT=*,RECFM=VA,LRECL=125                      */
/* //SYSTSPRT DD   SYSOUT=*                                         */
/* //DIAGMSGS DD   SYSOUT=*                                         */
/* //SYSTSIN  DD   DUMMY                                            */
/*                                                                   */
/* Return Codes:  ØØ - Copy successful                               */
/*                04 - SRCDSN was empty, TARGDSN was allocated       */
/*                12 - TARGDSN allocation error (or other TSO error) */
/*                14 - SRCDSN and TARGDSN can not be the same        */
/*                16 - SRCDSN exists and REPLACE=NO                  */
/*                2Ø - LMCOPY error (usually x37 abend)              */
```

```
/*                                                                    */
/**********************************************************************/
/*                        Change Log                                  */
/**********************************************************************/
/* Accept the input PDS                                               */
/**********************************************************************/
 arg srcdsn targdsn pattern replace .
 if pattern =  '' then pattern = '*'
 if replace =  '' then replace = 'NO'
/**********************************************************************/
/* Ensure we are in ISPF if run from batch                            */
/**********************************************************************/
 call ispfboot srcdsn targdsn pattern replace
/*********** @REFRESH BEGIN START    2005/06/06 03:58:28 *************/
/* 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'
 vardump = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' 0 'Parms:' parms
 if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
 call modtrace 'START' 0
/**********************************************************************/
/* Set local esoteric names                                           */
/**********************************************************************/
 @vio   = 'VIO'
 @sysda = 'SYSDA'
/*********** @REFRESH END   START    2005/06/06 03:58:28 *************/
/* Set flags and variables                                            */
/**********************************************************************/
 version   = '1.3.1'
 allocstat = 'NO'
 srcksds   = 'NO'
 LMCRC     =  0
 REPRC     =  0
/**********************************************************************/
/* Put quotes around the DSNs if supplied                             */
/**********************************************************************/
 if srcdsn  <> '' then srcdsn  = qdsn(srcdsn)
 if targdsn <> '' then targdsn = qdsn(targdsn)
/**********************************************************************/
/* Determine what type of dataset this is                             */
```

```
/*********************************************************************/
 srcdsorg = 'PO'
 srcpodir = 'NO'
 if srcdsn <> '' then
    do
     call listdsi srcdsn 'DIRECTORY'
     srcvol = sysvolume
/*********************************************************************/
/* Force a synchronous RECALL if migrated to ensure success          */
/*********************************************************************/
     if srcvol = '' then
        do
         call tsotrap "HRECALL" srcdsn "WAIT EXTENDRC"
         call listdsi srcdsn 'DIRECTORY'
         srcvol = sysvolume
        end
/*********************************************************************/
/* Save all dataset information required for later decisions         */
/*********************************************************************/
     srcdsorg = sysdsorg
     srcrecfm = sysrecfm
     srcunits = sysunits
     srcprisp = sysprimary
     srcsecsp = sysseconds
     srcblksz = sysblksize
     srcpodir = sysadirblk
    end
/*********************************************************************/
/* If VSAM do a LISTCAT to get space information                     */
/*********************************************************************/
 if srcdsorg = 'VS' then
    do
     call tsotrap "LISTCAT ENT("srcdsn") ALL"
     do v=1 to tsoout.Ø
        lcline = strip(translate(tsoout.v,' ','-'))
        select
          when wordpos('INDEXED',lcline) <> Ø then
               srcksds = 'YES'
          when pos('SPACE TYPE',lcline) <> Ø then
               parse var lcline . . srcunits .
          when pos('SPACE PRI',lcline) <> Ø then
               parse var lcline . . srcprisp .
          when pos('SPACE SEC',lcline) <> Ø then
               do
                parse var lcline . . srcsecsp .
                leave
               end
          otherwise iterate
        end
     end
```

```
    end
/*********************************************************************/
/* Display a pop-up                                                  */
/*********************************************************************/
 if srcdsn = '' | targdsn = '' then
    do
     targpos = 2
     if targdsn = '' then targdsn = qdsn(strip(srcdsn,"B","'")||'.NEW')
     copy.1  = ")ATTR                                                 "
     copy.2  = " @ TYPE(TEXT) COLOR(TURQ)                             "
     copy.3  = " # TYPE(INPUT) CAPS(ON) COLOR(RED)                    "
     copy.4  = " ! TYPE(OUTPUT) CAPS(ON) INTENS(NON)                  "
     copy.5  = ")BODY EXPAND(//) WINDOW(63,8)                         "
     copy.6  = "+Command%===>_ZCMD                                    "
     copy.7  = "%Specify the Source and Target DSN                    "
     copy.8  = "@Source DSN  : #Z                                     "
     copy.9  = "@Target DSN  : #Z                                     "
     select
       when srcdsorg = 'PO' then
            copy.1Ø = "@Member(s)   :  #Z       +(ISPF wildcards valid)"
       when srcdsorg = 'VS' & srcksds = 'YES' then
            do
             text = "Generic key end with '*'"
             copy.1Ø = "@KSDS Filter :  #Z                  +("text")"
            end
       otherwise
            copy.1Ø = "                !Z                             "
     end
     copy.11 = "/-/ Modifiable Allocation Parameters /-/              "
     if srcdsorg = 'PO' & datatype(srcpodir) = 'NUM' then
        copy.12 = "@Units       :  #Z         @Directory    :  #Z      "
     else
        copy.12 = "@Units       :  #Z                                 "
     copy.13 = "@Primary     :  #Z   +     @Secondary    :  #Z   + "
     copy.14 = ")INIT                                                 "
     copy.15 = " .ZVARS = '(SRCDSN TARGDSN PATTERN SRCUNITS +         "
     if srcdsorg = 'PO' & datatype(srcpodir) = 'NUM' then
        copy.16 = "            SRCPODIR SRCPRISP SRCSECSP)'          "
     else
        copy.16 = "            SRCPRISP SRCSECSP)'                   "
     copy.17 = " .CURSOR = TARGDSN                                    "
     copy.18 = " .CSRPOS = &TARGPOS                                   "
     copy.19 = " &PATTERN = *                                         "
     copy.2Ø = ")PROC                                                 "
     copy.21 = " VER (&SRCDSN,NB,DSNAMEQ)                            "
     copy.22 = " VER (&TARGDSN,NB,DSNAMEQ)                            "
     select
       when srcdsorg = 'VS' then
            copy.23 = " VER (&PATTERN,NB)                             "
       otherwise
```

```
            copy.23 = " VER (&PATTERN,NB,NAMEF)                          "
     end
     copy.24 = " VER (&SRCUNITS,NB,LIST,CYLINDER,TRACK,BLOCK)    "
     if srcdsorg = 'PO' & datatype(srcpodir) = 'NUM' then
        copy.25 = " VER (&SRCPODIR,NB,NUM)                          "
     else
        copy.25 = "                                                 "
     copy.26 = " VER (&SRCPRISP,NB,NUM)                          "
     copy.27 = " VER (&SRCSECSP,NB,NUM)                          "
     copy.28 = ")END                                             "
/*********************************************************************/
/* Display the Dynamic Panel                                         */
/*********************************************************************/
     call popdyn 'COPY' 4 'Enter' execname version 'Parameters'
/*********************************************************************/
/* Determine whether TARGDSN exists                                  */
/*********************************************************************/
     if sysdsn(qdsn(targdsn)) = 'OK' then
        do
         replace.1  = ")ATTR                                         "
         replace.2  = " @ TYPE(TEXT) COLOR(TURQ)                     "
         replace.3  = " # TYPE(INPUT) CAPS(ON) COLOR(RED)            "
         replace.4  = ")BODY EXPAND(//) WINDOW(6Ø,1)                 "
         replace.5  = "@Replace    : #Z   +(Type%YES+to overwrite)   "
         replace.6  = ")INIT                                         "
         replace.7  = " .ZVARS = '(REPLACE)'                         "
         replace.8  = ")PROC                                         "
         replace.9  = " VER (&REPLACE,NB,LIST,YES,NO)                "
         replace.1Ø = ")END                                          "
/*********************************************************************/
/* Display the Confirmation pop-up                                   */
/*********************************************************************/
         call popdyn 'REPLACE' 4 'Target DSN' targdsn 'Exists'
        end
/*********************************************************************/
/* Lock the screen with a pop-up                                     */
/*********************************************************************/
     call lock 'Copying' srcdsn 'to' targdsn
    end
/*********************************************************************/
/* Confirm the SRCDSN and TARGDSN are not the same                   */
/*********************************************************************/
 srcdsn = qdsn(srcdsn)
 targdsn = qdsn(targdsn)
 if srcdsn = targdsn then
    call rcexit 14 'SRCDSN and TARGDSN can not be the same'
/*********************************************************************/
/* Determine whether TARGDSN exists and REPLACE is NO                */
/*********************************************************************/
 if sysdsn(targdsn) = 'OK' & replace = 'NO' then
```

```
    call rcexit 16 'TARGDSN' targdsn 'exists and REPLACE =' replace
/*********************************************************************/
/* Delete the TARGDSN if it exists and REPLACE is YES                */
/*********************************************************************/
 if sysdsn(targdsn) = 'OK' & replace = 'YES' then
    call tsoquiet "DELETE" targdsn
/*********************************************************************/
/* Confirm the target DSN exists and allocate if necessary           */
/*********************************************************************/
 if sysdsn(targdsn) <> 'OK' then
    do
/*********************************************************************/
/* Determine whether any space parameters were changed               */
/*********************************************************************/
     if sysunits   <> srcunits | sysprimary <> srcprisp |,
        sysseconds <> srcsecsp then
        if srcunits = 'BLOCK' then
           newparm = srcunits'('srcblksz') SPACE('srcprisp srcsecsp')'
        else
           newparm = srcunits'S SPACE('srcprisp srcsecsp')'
     else
        newparm = ''
/*********************************************************************/
/* If this is a PDS and SRCPODIR changed add directory blocks        */
/*********************************************************************/
     if srcdsorg = 'PO' & srcpodir <> sysadirblk then
        newparm = newparm 'DIR('srcpodir')'
/*********************************************************************/
/* Add BLKSIZE if this is a LOADLIB (RECFM=U)                        */
/*********************************************************************/
     if srcrecfm = 'U' then
        call tsotrap "ALLOC F(TARGDSN) DA("targdsn") NEW" newparm,
                     "LIKE("srcdsn") BLKSIZE("srcblksz")"
     else
        call tsotrap "ALLOC F(TARGDSN) DA("targdsn") NEW" newparm,
                     "LIKE("srcdsn")"
     call tsotrap "FREE F(TARGDSN)"
    end
/*********************************************************************/
/* LMINIT the input and output datasets (if non-VSAM)                */
/*********************************************************************/
 if srcdsorg <> 'VS' then
    do
     call ispwrap "LMINIT DATAID(IN) DATASET("srcdsn") ENQ(SHR)"
     call ispwrap "LMINIT DATAID(OUT) DATASET("targdsn") ENQ(SHR)"
    end
 else
    do
     call tsotrap "ALLOC F(IN) DA("srcdsn") SHR REUSE"
     call tsotrap "ALLOC F(OUT) DA("targdsn") SHR REUSE"
```

```
    end
/*********************************************************************/
/* Set allocation status for shutdown                                */
/*********************************************************************/
 allocstat = 'YES'
/*********************************************************************/
/* Determine ISPF release level for use of ALIAS option on LMCOPY    */
/*********************************************************************/
 call ispwrap "VGET (ZENVIR)"
 parse var zenvir . ispfrel .
 parse var ispfrel ispfrel 'MVS'
 if ispfrel >= 5.5 then
    alias = 'ALIAS'
 else
    alias = ''
/*********************************************************************/
/* Let LMCOPY return error messages                                  */
/*********************************************************************/
 call ispwrap "CONTROL ERRORS RETURN"
/*********************************************************************/
/* Copy the dataset from IN to OUT                                   */
/*********************************************************************/
 select
/*********************************************************************/
/* LMCOPY a PDS                                                      */
/*********************************************************************/
   when srcdsorg = 'PO' then
        LMCRC = ispwrap(4 "LMCOPY FROMID("in") TODATAID("out")",
                          "FROMMEM("pattern") REPLACE" alias)
/*********************************************************************/
/* LMCOPY a sequential dataset                                       */
/*********************************************************************/
   when srcdsorg = 'PS' then
        LMCRC = ispwrap(4 "LMCOPY FROMID("in") TODATAID("out")")
/*********************************************************************/
/* REPRO a VSAM dataset                                              */
/*********************************************************************/
   when srcdsorg = 'VS' then
        do
         if pattern = '*' then
            REPRC = tsotrap(12 "REPRO INFILE(IN) OUTFILE(OUT) REPLACE")
         else
            REPRC = tsotrap(12 "REPRO INFILE(IN) OUTFILE(OUT) REPLACE",
                               "FROMKEY("pattern") TOKEY("pattern")")
        end
 end
/*********************************************************************/
/* Issue a message if the input was empty                            */
/*********************************************************************/
 if LMCRC = 4  then
```

```
    do
     call msg srcdsn 'was empty and so is' targdsn', RC=4'
     LMCRC = 4
    end
 if REPRC = 12 then
    do
     call msg srcdsn 'might have been empty, confirm',
                     targdsn 'is correct, RC=4'
     REPRC = 4
    end
/*********************************************************************/
/* Set EXITRC                                                        */
/*********************************************************************/
 EXITRC = max(LMCRC,REPRC)
/*********************************************************************/
/* Issue a success message if running in batch                       */
/*********************************************************************/
 if EXITRC = Ø & tsoenv = 'BACK' then
    call msg 'Copied' srcdsn 'to' targdsn', RC='EXITRC
/*********************************************************************/
/* Shutdown                                                          */
/*********************************************************************/
 shutdown: nop
/*********************************************************************/
/* Put unique shutdown logic before the call to stdexit              */
/*********************************************************************/
           if allocstat = 'YES' then
/*********************************************************************/
/* LMFREE the datasets                                               */
/*********************************************************************/
              do
               if srcdsorg <> 'VS' then
                  do
                   call ispwrap "LMFREE DATAID("in")"
                   call ispwrap "LMFREE DATAID("out")"
                  end
               else
                  call tsotrap "FREE F(IN OUT)"
              end
/*********** @REFRESH BEGIN STOP     2002/Ø8/Ø3 Ø8:42:33 *************/
/* Shutdown message and terminate                                    */
/*********************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2002/Ø8/Ø3 Ø8:42:33 *************/
/*********** @REFRESH BEGIN SUBBOX   2004/Ø3/1Ø Ø1:25:Ø3 *************/
/*                                                                   */
/* 29 Internal Subroutines provided in COPY                          */
/*                                                                   */
/* Last Subroutine REFRESH was 11 Jun 2005 12:41:32                  */
/*                                                                   */
```

```
/*                                                                   */
/* 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 is only one set of quotes              */
/* UNIQDSN  - Create a unique dataset name                           */
/* TEMPMEM  - EXECIO a stem into a TEMP PDS                          */
/* ISPFBOOT - Include only in modules that must start under ISPF     */
/* ISPWRAP  - Wrapper for ISPF commands                              */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* TSOQUIET - Trap all output from a TSO command and ignore failures */
/* SETBORD  - Set the ISPF Pop-up active frame border colour         */
/* 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               */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* MODTRACE - Module Trace                                           */
/*                                                                   */
/*********** @REFRESH END   SUBBOX   2004/03/10 01:25:03 *************/
/*********** @REFRESH BEGIN RCEXIT   2005/04/21 15:23:32 *************/
/* 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 <> 0 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
/*********************************************************************/
/* Print the VARDUMP values (if present)                             */
/*********************************************************************/
          if vardump <> 'NONE' then
             do
              if tsoenv = 'BACK' then
                 do
                  say
                  say 'Selected variable values:'
                  say
                  do vd=1 to words(vardump)
                     interpret "say word(vardump,vd) '='",
                               word(vardump,vd)
                  end
                  say
                 end
               else
                 do
                  zerrlm = 'Selected variable values:'
                  address ISPEXEC "LOG MSG(ISRZØØ3)"
                  do vd=1 to words(vardump)
                     interpret "zerrlm = word(vardump,vd) '='",
                               word(vardump,vd)
                     address ISPEXEC "LOG MSG(ISRZØØ3)"
                  end
                 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
/*********** @REFRESH END   RCEXIT   2ØØ5/Ø4/21 15:23:32 *************/
```

```
/*********** @REFRESH BEGIN TRAP     2004/12/13 14:00:48 *************/
/* 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
/*********** @REFRESH END   TRAP     2004/12/13 14:00:48 *************/
/*********** @REFRESH BEGIN ERRMSG   2002/08/10 16:53:04 *************/
/* 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
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2005/01/30 07:49:49 *************/
/* 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'
           lower = xrange('a','z')
           upper = xrange('A','Z')
/*********************************************************************/
/* 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
/*********** @REFRESH END   STDENTRY 2005/01/30 07:49:49 *************/
/*********** @REFRESH BEGIN STDEXIT  2004/08/02 06:06:40 *************/
/* STDEXIT  - Standard Exit logic                                    */
/*-------------------------------------------------------------------*/
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
/*********************************************************************/
 stdexit: module = 'STDEXIT'
          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 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 Ø 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)
/*********** @REFRESH END   STDEXIT  2004/08/02 06:06:40 *************/
/*********** @REFRESH BEGIN MSG      2002/09/11 01:35:53 *************/
/* 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) <> 0 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(ISRZ000)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2002/09/11 01:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2004/11/09 22:48:36 *************/
/* DDCHECK  - Determine if 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
/*********** @REFRESH END   DDCHECK  2004/11/09 22:48:36 *************/
/*********** @REFRESH BEGIN DDLIST   2002/12/15 04:54:32 *************/
/* 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
/*********** @REFRESH END   DDLIST   2002/12/15 04:54:32 *************/
/*********** @REFRESH BEGIN DDDSNS   2002/09/11 00:37:36 *************/
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*-------------------------------------------------------------------*/
/* TARGDD   - DD to return DSNs for                                  */
/*********************************************************************/
 dddsns: module = 'DDDSNS'
         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 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 = 0
         ddname = '$DDNAME$'
/*********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/*********************************************************************/
         do ddd=1 to lines.0
            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
/*********** @REFRESH END   DDDSNS   2002/09/11 00:37:36 *************/
/*********** @REFRESH BEGIN QDSN     2002/09/11 01:15:23 *************/
/* QDSN     - Make sure there are only one set of quotes             */
/*-------------------------------------------------------------------*/
/* QDSN     - The DSN                                                */
/*********************************************************************/
 qdsn: module = 'QDSN'
       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 qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return qdsn
/*********** @REFRESH END   QDSN     2002/09/11 01:15:23 *************/
/*********** @REFRESH BEGIN UNIQDSN  2005/04/26 12:45:35 *************/
/* UNIQDSN  - Create a unique dataset name                           */
/*-------------------------------------------------------------------*/
/* UNIQHLQ  - HLQ to use (blank for Userid, PREF for SYSPREF or any) */
/*********************************************************************/
 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
/*********************************************************************/
/* Determine whether there is a special HLQ requirement              */
/*********************************************************************/
          arg uniqhlq
          select
            when uniqhlq = '' then uniqhlq = userid()
            when uniqhlq = 'SYSPREF' then uniqhlq = sysvar('SYSPREF')
            otherwise nop
          end
/*********************************************************************/
/* 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 = uniqhlq'.'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
/*********** @REFRESH END   UNIQDSN  2ØØ5/Ø4/26 12:45:35 *************/
/*********** @REFRESH BEGIN TEMPMEM  2ØØ4/Ø9/Ø1 17:2Ø:19 *************/
/* 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
/*********** @REFRESH END   TEMPMEM  2004/09/01 17:20:19 *************/
/*********** @REFRESH BEGIN ISPFBOOT 2005/06/11 12:41:19 *************/
/* ISPFBOOT - Include only in modules that must start under ISPF     */
/*            MUST BE EXECUTED BEFORE STDENTRY                       */
/*********************************************************************/
 ispfboot: parse arg execparm, ispfparm
           parse upper source . . execname .
/*********************************************************************/
/* Test whether ISPF is available                                    */
/*********************************************************************/
           if sysvar('SYSISPF') = 'NOT ACTIVE' then
              do
/*********************************************************************/
/* If not, queue an ISPSTART command                                 */
/*********************************************************************/
               queue 'ISPSTART CMD(%'execname execparm')' ispfparm
               exit(111)
              end
           else
/*********************************************************************/
```

```
/* If so, just return                                                 */
/*********************************************************************/
             return
/*********** @REFRESH END   ISPFBOOT 2005/06/11 12:41:19 *************/
/*********** @REFRESH BEGIN ISPWRAP  2002/09/11 01:11:43 *************/
/* ISPWRAP  - Wrapper for ISPF commands                               */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC from the ISPF command, defaults to 0 */
/* ISPPARM  - Valid ISPF command                                      */
/*********************************************************************/
 ispwrap: module = 'ISPWRAP'
          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 ispparm
          zerrlm = 'NO ZERRLM'
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume 0  */
/*********************************************************************/
          parse var ispparm valid_rc isp_cmd
          if datatype(valid_rc,'W') = 0 then
             do
              valid_rc = 0
              isp_cmd = ispparm
             end
          address ISPEXEC isp_cmd
          IRC = RC
/*********************************************************************/
/* If RC = 0 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
/*********** @REFRESH END   ISPWRAP  2002/09/11 01:11:43 *************/
/*********** @REFRESH BEGIN TSOTRAP  2002/12/15 05:18:45 *************/
/* TSOTRAP  - Capture the output from a TSO command in a stem         */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                     */
/* TSOPARM  - Valid TSO command                                       */
/*********************************************************************/
```

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

```
           return
/*********** @REFRESH END   TSOQUIET 2003/05/26 10:22:58 *************/
/*********** @REFRESH BEGIN SETBORD  2002/09/11 01:16:41 *************/
/* SETBORD  - Set the ISPF Pop-up active frame border colour         */
/*-------------------------------------------------------------------*/
/* COLOR    - Color for the Active Frame Border                      */
/*********************************************************************/
 setbord: module = 'SETBORD'
          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 color
/*********************************************************************/
/* Parse and validate colour                                         */
/*********************************************************************/
          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
/*********** @REFRESH END   SETBORD  2002/09/11 01:16:41 *************/
/*********** @REFRESH BEGIN LOCK     2004/09/01 18:00:03 *************/
/* 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) <> 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 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 <= 10 then locklen = 10
/*********************************************************************/
/* 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
/*********** @REFRESH END   LOCK     2004/09/01 18:00:03 *************/
/*********** @REFRESH BEGIN UNLOCK   2003/10/18 09:33:19 *************/
/* 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
/*********** @REFRESH END   UNLOCK   2003/10/18 09:33:19 *************/
/*********** @REFRESH BEGIN PANDSN   2004/04/28 00:46:04 *************/
/* 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
/*********** @REFRESH END   PANDSN   2004/04/28 00:46:04 *************/
/*********** @REFRESH BEGIN POPDYN   2002/09/11 01:15:11 *************/
/* 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 Colour                                    */
/*********************************************************************/
         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 Colour                                    */
/*********************************************************************/
         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
/*********** @REFRESH END   POPDYN   2002/09/11 01:15:11 *************/
/*********** @REFRESH BEGIN SAYDD    2004/03/29 23:48:37 *************/
/* SAYDD    - Print messages to the requested DD                     */
/*-------------------------------------------------------------------*/
/* MSGDD    - DDNAME to write messages to                            */
/* MSGLINES - number of blank lines to put before and after          */
/* MESSAGE  - Text to write to the MSGDD                             */
/*********************************************************************/
 saydd: module = 'SAYDD'
        if wordpos(module,probe) <> 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 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,6Ø)
             messseg = substr(message,1,messst)
             msgline.msgm = date() time() strip(messseg)
             message = strip(delstr(message,1,messst))
             do while length(message) > Ø
                msgm = msgm + 1
                if length(message) > 55 then
                   messst = lastpos(' ',message,55)
                if messst > Ø then
                   messseg = substr(message,1,messst)
                else
                   messseg = substr(message,1,length(message))
                msgline.msgm = date() time() 'CONT:' strip(messseg)
                message = strip(delstr(message,1,length(messseg)))
             end
            end
         else
/*********************************************************************/
/* Build print lines. Default strips and prefixes date and timestamp */
/* @BLANK - Blank line, no date and timestamp                        */
/* @      - No stripping, retains leading blanks                     */
/* @@     - No stripping, No date and timestamp                      */
/*********************************************************************/
            do
             select
                when message = '@BLANK@' then msgline.msgm = ' '
                when word(message,1) = '@' then
                     do
                      message = substr(message,2,length(message)-1)
                      msgline.msgm = date() time() message
                     end
                when substr(message,1,2) = '@@' then
                     do
                      message = substr(message,3,length(message)-2)
                      msgline.msgm = message
                     end
                otherwise msgline.msgm = date() time() strip(message)
             end
            end
/*********************************************************************/
/* If a number is provided, add that number of blank lines after     */
/* the message                                                       */
/*********************************************************************/
         if msglines > Ø then
            do msgt=1 to msglines
               msge = msgt + msgm
               msgline.msge = ' '
            end
/*********************************************************************/
```

```
/* Write the contents of the MSGLINE stem to the MSGDD                */
/*********************************************************************/
        call tsotrap "EXECIO * DISKW" msgdd "(STEM MSGLINE. FINIS"
        drop msgline. msgb msgt msge
        pull tracelvl . module . sigl . sparms
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return
/*********** @REFRESH END   SAYDD    2004/03/29 23:48:37 *************/
/*********** @REFRESH BEGIN JOBINFO  2004/11/23 22:11:25 *************/
/* JOBINFO  - Get job-related data from control blocks                */
/*-------------------------------------------------------------------*/
/* ITEM     - Optional item number desired, default is all            */
/*********************************************************************/
 jobinfo: module = 'JOBINFO'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg item
/*********************************************************************/
/* Chase control blocks                                              */
/*********************************************************************/
          tcb      = ptr(540)
          ascb     = ptr(548)
          tiot     = ptr(tcb+12)
          jscb     = ptr(tcb+180)
          ssib     = ptr(jscb+316)
          asid     = c2d(stg(ascb+36,2))
          jobtype  = stg(ssib+12,3)
          jobnum   = strip(stg(ssib+15,5),'L',0)
          stepname = stg(tiot+8,8)
          procstep = stg(tiot+16,8)
          progname = stg(jscb+360,8)
          jobdata  = jobtype jobnum stepname procstep progname asid
/*********************************************************************/
/* Return job data                                                   */
/*********************************************************************/
          if item <> '' & (datatype(item,'W') = 1) then
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return word(jobdata,item)
             end
          else
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
```

```
             return jobdata
            end
/*********** @REFRESH END   JOBINFO  2004/11/23 22:11:25 *************/
/*********** @REFRESH BEGIN PTR      2002/07/13 15:45:36 *************/
/* PTR      - Pointer to a storage location                          */
/*-------------------------------------------------------------------*/
/* ARG(1)   - Storage Address                                        */
/*********************************************************************/
 ptr: return c2d(storage(d2x(arg(1)),4))
/*********** @REFRESH END   PTR      2002/07/13 15:45:36 *************/
/*********** @REFRESH BEGIN STG      2002/07/13 15:49:12 *************/
/* STG      - Return the data from a storage location                */
/*-------------------------------------------------------------------*/
/* ARG(1)   - Location                                               */
/* ARG(2)   - Length                                                 */
/*********************************************************************/
 stg: return storage(d2x(arg(1)),arg(2))
/*********** @REFRESH END   STG      2002/07/13 15:49:12 *************/
/*********** @REFRESH BEGIN MODTRACE 2003/12/31 21:56:54 *************/
/* MODTRACE - Module Trace                                           */
/*-------------------------------------------------------------------*/
/* TRACETYP - Type of trace entry                                    */
/* SIGLINE  - The line number called from                            */
/*********************************************************************/
 modtrace: if modtrace = 'NO' then return
           arg tracetyp sigline
           tracetyp = left(tracetyp,5)
           sigline = left(sigline,5)
/*********************************************************************/
/* Adjust MODSPACE for START                                         */
/*********************************************************************/
           if tracetyp = 'START' then
              modspace = substr(modspace,1,length(modspace)+1)
/*********************************************************************/
/* Set the trace entry                                               */
/*********************************************************************/
          traceline = modspace time('L') tracetyp module sigline sparms
/*********************************************************************/
/* Adjust MODSPACE for STOP                                          */
/*********************************************************************/
           if tracetyp = 'STOP' then
              modspace = substr(modspace,1,length(modspace)-1)
/*********************************************************************/
/* Determine where to write the traceline                            */
/*********************************************************************/
           if ispfenv = 'YES' & tsoenv = 'FORE' then
/*********************************************************************/
/* Write to the ISPF Log, do not use ISPWRAP here                    */
/*********************************************************************/
             do
```

```
               zedlmsg = traceline
               address ISPEXEC "LOG MSG(ISRZ000)"
              end
            else
               say traceline
/*********************************************************************/
/* SAY to SYSTSPRT                                                   */
/*********************************************************************/
            return
/************ @REFRESH END   MODTRACE 2003/12/31 21:56:54 *************/
```

Robert Zenuk
Systems Programmer (USA)

JES2 spool offload report

Every now and then an IPL of the system is required – one of the reasons for this could be that a restart of JES2 with the cold start option is needed. It is true that very few definitions, or redefinitions, of JES2 facilities and resources require that the JES2 system be totally shut down. Should that happen, JES2 must be restarted with a cold start to allow all component systems to be aware of the changed facilities and resources. For example, the use of the FORMAT option causes a cold start. The time taken to restart JES2 in this manner is based on the work in the system and, if not scheduled, causes a disruption to data processing services. All job data previously on the spool volumes is lost with a cold start. Let us remember that the spool is the primary medium for all JES2 data. All input jobs and system output (SYSOUT) are stored on the spool. However, you can avoid data loss by scheduling a spool offload of the JES2 queues. JES2 gives your installation the ability to offload (transmit) data from the spool to a dataset, and later reload (receive) the data from the dataset to the spool. The spool offload facility uses either tape or DASD as the offload medium. In addition to that, spool offload eases the migration from release to release by lessening the impact of a cold start. The spool offload facility provides a JES2 release-

independent means of moving the data on the spool between releases. Besides preserving jobs and SYSOUT across a cold start and migration to another release of JES2, the spool offload facility is useful if you need to convert to another DASD type for spool, archive system jobs and SYSOUT, relieve a full-spool condition during high-use periods, or simply provide a back-up for spool datasets (for disaster recovery purpose, for example).

JES2 offers many selection criteria to limit the spool offload operation. These selection criteria can be changed by operator commands, according to the initial specification in the JES2 initialization statements. A detailed description of how to define and set up, as well as how to use, the spool offload facility, can be obtained from the *JES2 Initialization and Tuning Guide* (SA22-7532) manual. It is quite obvious that it is up to the individual site to decide how best to implement the offload back-up and retrieval functions. Personally, I tend to agree with the notion that spool offload data should be considered temporary – otherwise people will use it for long-term storage and it'll fill all the time – and in a disaster recovery situation, I doubt if I'd bother reloading it. The offload is mostly for people who accidentally purge their job's output and can't readily re-run.

When enabled by the SMFPRM*xx* TYPE parameter of your system parameter library, SMF creates record type 24 whenever a job or SYSOUT dataset is transmitted to or received from an offload dataset. JES2 writes one type 24 record for each pre-processing job that is transmitted to an offload dataset or received back to the spool. Because one type 24 record is written for each SYSOUT dataset header that is transmitted or received, multiple type 24 records can be expected for each post-processing job. This record identifies the name, time, and date of each job that has been transmitted or received. It includes specific information about jobs in a record subtype. For jobs not yet run, it reports job-related information such as job class and system affinity in both the job selection criteria section and in the system affinity section.

For jobs that have already run, it reports information about SYSOUT datasets such as output group ID and forms name in the SYSOUT selection criteria section. Record type 24 never contains both the job selection criteria section and the SYSOUT selection criteria section. A detailed description of layout of an SMF type 24 record and its subtypes can be obtained from the *MVS System Management Facilities (SMF)* (SA22-7630) manual.

Based on the JES2 spool offload type 24 record description obtained from its mapping macro (IAZSMF24, which resides in SYS1.MACLIB) a sample JES2 spool offload event report writer was written. In order to extract JES2 spool offload event information from SMF data, I have constructed a three-part job stream. In the first part (DUMP24), SMF records 24 are extracted from the SMF weekly dataset to a file, which can be used as a base of archived records. In the second step (COPY24), previously extracted records (selection being defined by INCLUDE's condition) are sorted and copied to a file, which is the input to the analysis and reporting J2OFF REXX EXEC invoked in the last (OFF24) step. There is only one report generated by this REXX EXEC; it is called the JES2 Spool offload report. The report provides information about the event/operation that caused the spool offload facility to write to the SMF record.

Sample JCL to execute SMF type 24 data extract and JES2 spool offload reporting:

```
//*-------------------------------------------------------------------*
//* UNLOAD SMF 24 RECORDS FROM VSAM OR VBS TO VB                      *
//* Note: change the DUMPIN DSN=your.smfdata to be the name of        *
//* the dataset where you currently have SMF data being               *
//* recorded. It may be either SMF weekly dataset or an active        *
//* dump dataset. If you chose the latter, then prior to              *
//* executing this job, you need to terminate SMF recording           *
//* of the currently active dump dataset for allow the                *
//* unload of SMF records.                                            *
//* Also, change the DCB reference to match the name of your          *
//* weekly smf dump dataset.                                          *
//*-------------------------------------------------------------------*
//DUMP24   EXEC  PGM=IFASMFDP
//DUMPIN   DD    DISP=SHR,DSN=your.smfdata
```

```
//DUMPOUT  DD    DISP=(NEW,PASS),DSN=&&SMF24OUT,UNIT=SYSDA,
//         SPACE=(CYL,(2,2)),DCB=(your.smfweekly.dataset)
//SYSPRINT DD    SYSOUT=*
//SYSIN    DD    *
    INDD(DUMPIN,OPTIONS(DUMP))
    OUTDD(DUMPOUT,TYPE(24))
/*
//*-------------------------------------------------------------------*
//* COPY VBS TO VB, DROP HEADER/TRAILER RECORDS, SORT ON DATE/TIME *
//* Note: change the SMF24 DSN=hlq.SMF24.DATA to be the name of    *
//* the dataset you'll use in the last step.                        *
//*-------------------------------------------------------------------*
//COPY24   EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=&&SMF24OUT,DISP=SHR
//SMF24    DD DSN=hlq.SMF24.DATA,SPACE=(CYL,(x,y)),UNIT=SYSDA,
//         DISP=(NEW,CATLG,KEEP),
//         DCB=(RECFM=VB,LRECL=32756,BLKSIZE=3276Ø)
//TOOLIN   DD *
   SORT FROM(RAWSMF) TO(SMF24) USING(SMFI)
//SMFICNTL DD *
   OPTION SPANINC=RC4,VLSHRT
   INCLUDE COND=(6,1,BI,EQ,24)
   SORT FIELDS=(11,4,PD,A,7,4,BI,A)
/*
//*-------------------------------------------------------------------*
//* FORMAT JES2 Spool offload TYPE 24 RECORDS                       *
//* Note: change the SYSEXEC DSN=your.rexx.library to be the name  *
//* of the dataset where you have placed the J2OFF REXX EXEC.      *
//* Also, change the SMF24 DSN=hlq.SMF24.DATA to be the name of    *
//* the dataset you have created in previous step.                  *
//*-------------------------------------------------------------------*
//OFF24    EXEC PGM=IKJEFTØ1,REGION=ØM
//SYSEXEC  DD DISP=SHR,DSN=your.rexx.library
//SMF24    DD DISP=SHR,DSN=hlq.SMF24.DATA
//SYSPRINT DD SYSOUT=*
//SYSTSPRT DD SYSOUT=*
//SYSTSIN  DD *
prof nopref
%J2OFF
/*
```

J2OFF reporting EXEC:

```
/* REXX EXEC to read and format JES2 Spool Offload  */
/* SMF 24 records.                                  */
/*--------------------------------------------------*/
Address TSO
userid=SYSVAR(SYSUID)
```

```
r24off = userid||'.offload.rep'
If SYSDSN(r24off) = 'OK'
Then "DELETE "r24off" PURGE"
  "ALLOC FILE(REPORT) DA("r24off")",
  "UNIT(SYSALLDA) NEW TRACKS SPACE(90,50) CATALOG",
  "REUSE LRECL(400) RECFM(F B) "
first = 0; prev = 0; Totrec= 0
eoj = left(' ',5,' '); sut   = left(' ',5,' ')
b80 = left(' ',5,' '); b40   = left(' ',5,' ')
b20 = left(' ',5,' '); esser = left(' ',5,' ')
jf80= left(' ',5,' '); jf40 = left(' ',5,' ')
/*-------------------------------------------------------------*/
/*  Print report header                                        */
/*-------------------------------------------------------------*/
 hdr.1 = left('JES2 Spool offload report',50)
 hdr.2 = left(' ',1,' ')
 hdr.3  =left('Report produced on',18)      ,
         ||left(' ',1,' ')||left(date(),12) ,
         ||left('at',3,' ')||left(time(),10)
 hdr.4 = left(' ',1,' ')
 hdr.5 = left('Jobname',9)||left("Job id.",8)  ,
         ||left('time',7)||left("Job's TOD",22),
         ||left('Class',7)||left('Node',7)     ,
         ||left("# rec",6)||left('ds rec',7)   ,
         ||left('RMT',4)||left('FCB',4)        ,
         ||left('Forms',6)||left('Flash',6)    ,
         ||left('PRM',4)||left('UCS',4)        ,
         ||left('Writer',7)||left('Sysout status',14),
         ||left('Job status',10)
 hdr.6 = left('-',133,'-')
 "EXECIO * DISKW REPORT (STEM hdr.)"
 'EXECIO * DISKR SMF24 (STEM x. FINIS '
     Do i = 1 to x.0
/*-------------------------------------------------------------*/
/* Header/Self-defining Section:                               */
/* This section contains the common SMF record headers         */
/* fields and the triplet fields (offset/length/number) that   */
/* locate the other sections on the record. This triplet       */
/* information should be checked prior to accessing a          */
/* section of the record. All three fields being non-zero      */
/* mean that the section exists on the record; conversely any  */
/* of the fields being zero indicates that the section does    */
/* not exist on the record. The "number" triplet field is the  */
/* primary indication of the existence of the field.           */
/*-------------------------------------------------------------*/
   smf24rty = c2d(substr(x.i,2,1))               /* SMF record type  */
IF smf24rty = '24'  Then
   Do
   smf24tme = smf(c2d(substr(x.i,3,4)))          /* SMF record time  */
```

```
   smf24tm  =     c2d(substr(x.i,3,4))            /* SMF time         */
   smf24dte = substr(c2x(substr(x.i,Ø7,Ø4)),3,5)  /* SMF record date  */
   smf24sid = substr(x.i,11,4)                    /* System id        */
   smf24ssi = substr(x.i,15,4)                    /* Subsystem id     */
   smf24sub = c2d(substr(x.i,19,2))               /* Record subtype   */
   smf24ntr = c2d(substr(x.i,21,2))              /* Number of triplets*/
Select
   when smf24sub = 1 then note ="Job transmitted    (subtype 1)"
   when smf24sub = 2 then note ="Job received       (subtype 2)"
   when smf24sub = 3 then note ="SYSOUT transmitted (subtype 3)"
   when smf24sub = 4 then note ="SYSOUT received    (subtype 4)"
  otherwise note ="error"
end
   smf24ops = c2d(substr(x.i,25,4))       /* Offset to Product section*/
   smf24lps = c2d(substr(x.i,29,2))       /* Length of Product section*/
   smf24nps = c2d(substr(x.i,31,2))       /* Number of Product section*/
   smf24ogn = c2d(substr(x.i,33,4))       /* Offset to General section*/
   smf24lgn = c2d(substr(x.i,37,2))       /* Length of General section*/
   smf24ngn = c2d(substr(x.i,39,2))       /* Number of general section*/
   smf24osp = c2d(substr(x.i,41,4))       /* Offset to SPOF section   */
   smf24lsp = c2d(substr(x.i,45,2))       /* Length of SPOF section   */
   smf24nsp = c2d(substr(x.i,47,2))       /* Number of SPOG sections  */
   smf24osw = c2d(substr(x.i,49,4))       /* Offset to ESS section    */
   smf24lsw = c2d(substr(x.i,53,2))       /* Length of ESS section    */
   smf24nsw = c2d(substr(x.i,55,2))       /* Number of ESS sections   */
   smf24osa = c2d(substr(x.i,57,4))       /* Offset to Sysaff section */
   smf24lsa = c2d(substr(x.i,61,2))       /* Length of Sysaff section */
   smf24nsa = c2d(substr(x.i,63,2))       /* Number of Sysaff sections*/
/*-------------------------------------------------------------------*/
/* Product Section                                                   */
/* This section is located on the record using the following         */
/* triplet fields, which are located in the "Header/self-defining"*/
/* section: offset (smf24ops); length (mf24lps); number (smf24nps)*/
/*-------------------------------------------------------------------*/
IF smf24ops <> Ø AND smf24nps <> Ø Then do
          pof = smf24ops -3
     smf24pvr = substr(x.i,pof,2)        /* Record version number */
     smf24pnm = substr(x.i,pof+2,8)      /* Product name          */
end    /* End of Product Section */
/*-------------------------------------------------------------------*/
/* General Section                                                   */
/* This section contains the statistics for spool offload devices.*/
/* Triplet Information:                                              */
/* Offset (smf24ops); Length (smf24lps); Number (smf24nps)           */
/*-------------------------------------------------------------------*/
IF smf24lgn <> Ø AND smf24ngn <> Ø Then do
     gof = smf24ogn - 3
smf24gln = c2d(substr(x.i,gof,2))        /* Length of general section */
smf24bcf =     substr(x.i,gof+2,1)       /* Buffer continuation flag  */
smf24fst = '80000000'X                   /* First SMF buffer for job  */
```

```
smf24con = '40000000'X                    /* Continuation of SMF buffer*/
smf24lst = '20000000'X                  /* Last SMF buffer - end of job*/
 if bitand(smf24bcf,smf24fst) = smf24fst then,
    b80 = "First SMF buffer for job"
 if bitand(smf24bcf,smf24con) = smf24con then,
    b40 = "Smf buffer continued"
 if bitand(smf24bcf,smf24lst) = smf24lst then,
    b20 = "Last SMF buffer - end of job"
smf24eoj =   substr(x.i,gof+3,1)                    /* End of job flag */
smf24com =  '80000000'x                   /* Job completely offloaded */
smf24sds =  '40000000'x         /* Job completed with skipped datasets*/
smf24inj =  '20000000'x                   /* Incomplete job offloaded */
smf24opr =  '10000000'x                     /* Operator cancelled job */
 if bitand(smf24eoj,smf24com) = smf24com then,
    eoj = 'Completed job offload'
 if bitand(smf24eoj,smf24sds) = smf24sds then,
    eoj = 'Job completed with skipped data sets'
 if bitand(smf24eoj,smf24inj) = smf24inj then,
    eoj = 'Uncompleted job offloaded'
 if bitand(smf24eoj,smf24opr) = smf24opr then,
    eoj = 'Job cancelled by operator'
smf24jbn = substr(x.i,gof+4,8)                    /* Job name          */
smf24jid = substr(x.i,gof+12,8)                   /* Original job id.  */
smf24cjd = substr(x.i,gof+20,8)                   /* Current job id.   */
smf24sys = substr(x.i,gof+28,4)                   /* System id.        */
smf24dsn = substr(x.i,gof+32,44)                  /* Offload ds name   */
smf24cnt = c2d(substr(x.i,gof+76,4))      /* No. of rec. dumped-loaded*/
smf24tds = smf(c2d(substr(x.i,gof+80,4))) /* Time offload ds allocated*/
smf24td  =     c2d(substr(x.i,gof+80,4))          /* Time offload ds   */
smf24dds = substr(c2x(substr(x.i,gof+84,4)),3,5)
                                         /* Date offload ds allocated*/
smf24org = substr(x.i,gof+88,8)                   /* Origin node       */
smf24trd = smf(c2d(substr(x.i,gof+96,4)))      /* Time on reader - job*/
smf24drd = substr(c2x(substr(x.i,gof+100,4)),3,5)
                                          /*Date on reader - job     */
Totrec = Totrec + smf24cnt
/*-----------------------------------------------------------------*/
/* Calculate duration of a single list offloading/reloading        */
/*-----------------------------------------------------------------*/
Select
   when i = 1 Then do
      first = smf24td
      listoff = cross(smf24tm,smf24td)
      prev = smf24tm
     end
   otherwise do
      listoff = cross(smf24tm,prev)
      prev = smf24tm
     end
End
```

```
end     /* End of General Section */
/*-----------------------------------------------------------------*/
/* Job or SYSOUT selection criteria section:                       */
/* Triplet Information:                                            */
/* Offset (smf24osp); Length (smf24lsp); Number (smf24nsp)         */
/* Either the Job section or the SYSOUT section is written,        */
/* not both. The spof triplet refers to whichever one is written   */
/* in the current record.                                          */
/*-----------------------------------------------------------------*/
IF smf24lsp <> Ø AND smf24nsp <> Ø Then do
   sof = smf24osp - 3
/*-----------------------------------------------------------------*/
/*  Job Selection Criteria Section (subtytes 1 & 2)                */
/*-----------------------------------------------------------------*/
   IF smf24sub <= 2 Then do;
smf24ln1 = c2d(substr(x.i,sof,2))        /* Length of job section */
smf24jfg =     substr(x.i,sof+2,1)       /* Job flags:            */
smf24jhl = '80000000'x                   /* Held job              */
smf24aff = '40000000'x                   /* Affinity = any        */
 if bitand(smf24jfg,smf24jhl) = smf24jhl then,
 jf8Ø = "Held job"
 if bitand(smf24jfg,smf24aff) = smf24aff then,
 jf4Ø = "Affinity = any"
smf24jcl = substr(x.i,sof+3,1)           /* Job class             */
smf24jnd = substr(x.i,sof+4,8)           /* Node name             */
smf24jaf = substr(x.i,sof+12,28)         /* Affinity system ids   */
end   /* of Job Selection Criteria Section */
/*-----------------------------------------------------------------*/
/* Sysout selection criteria section  (subtypes 3 & 4)             */
/*-----------------------------------------------------------------*/
 IF smf24sub  >= 3 Then do;
smf24ln2 = c2d(substr(x.i,sof,2))         /* Length of sysout section */
smf24sfg =     substr(x.i,sof+2,1)        /* Sysout flags             */
smf24shl = '80000000'x                    /* Held SYSOUT              */
smf24sbt = '40000000'x                    /* Bursted SYSOUT           */
smf24sjh = '20000000'x                    /* Held job                 */
smf24inc = '10000000'x                    /* Incomplete dataset       */
smf24mul = '08000000'x                    /* Multi-destination ds     */
 if bitand(smf24sfg,smf24shl) = smf24shl then,
    sut = "Held SYSOUT"
 if bitand(smf24sfg,smf24sbt) = smf24sbt then,
    sut = "Bursted SYSOUT"
 if bitand(smf24sfg,smf24sjh) = smf24sjh then,
    sut ="Held job"
 if bitand(smf24sfg,smf24inc) = smf24inc then,
    sut ="Incomplete data set"
 if bitand(smf24sfg,smf24mul) = smf24mul then,
    sut ="Multi-destination ds"
smf24scl = substr(x.i,sof+3,1)            /* SYSOUT class             */
smf24snd = substr(x.i,sof+4,8)            /* Node name                */
```

```
smf24srn = substr(x.i,sof+12,8)           /* Remote name              */
smf24fcb = substr(x.i,sof+20,4)           /* Forms control buffer     */
smf24for = substr(x.i,sof+24,8)           /* Forms overlay name       */
smf24fls = substr(x.i,sof+32,4)           /* Flash cartridge name     */
smf24prm = substr(x.i,sof+36,8)           /* Print dataset (PR) mode  */
smf24ucs = substr(x.i,sof+44,4)           /* Universal character set  */
smf24wid = substr(x.i,sof+48,8)           /* Writer                   */
smf24rec = c2d(substr(x.i,sof+56,4))      /* Dataset record count     */
smf24pry = c2d(substr(x.i,sof+60,1))      /* Output selection priority*/
 end /* of SYSOUT selection criteria section  */
end
/*-----------------------------------------------------------------*/
/* Enhanced SYSOUT Support (ESS) section:                          */
/* This section contains the OUTPUT descriptor (if any) in SWBTU   */
/* format (IEFSJPFX plus text units) for the first offloaded data  */
/* set included in this SMF record. The SWBTU may be processed     */
/* using the SWBTUREC macro or other Scheduler JCL Facility (SJF)  */
/* services. Triplet Information:                                  */
/* offset (smf24osw); length (smf24lsw); number (smf24nsw)         */
/*-----------------------------------------------------------------*/
 IF smf24lsw  > 0 AND smf24nsw  > 0 Then do
    eff = smf24osw - 3
smf24ln3 = c2d(substr(x.i,eff,2))            /* Length of ESS section */
smf24sgt = c2d(substr(x.i,eff+2,4))          /* Segment identifier    */
smf24ind =     substr(x.i,eff+6,1)           /* ESS section indicator */
smf24sjf = '80000000'x             /* Error obtaining scheduler JCL   */
                                   /* facility (SJF) information.     */
                                   /* Scheduler work block text unit  */
                                   /* (SWBTU) data area is not present*/
if bitand(smf24ind,smf24sjf) = smf24sjf then,
   esser = "Error obtaining scheduler JCL facility"
smf24jdt = substr(x.i,eff+8,8)     /* JCL definition table (JDT) name */
                                   /* in JCL definition vector table  */
smf24tul = c2d(substr(x.i,eff+16,2)) /* Text unit (SWBTU) data area   */
                                     /* length                        */
smf24tu = substr(x.i,eff+18,smf24tul) /* Text unit (SWBTU) data area. */
                                    /* The data area can be processed */
                                       /* using the SWBTUREQ macro  */
end /* of Enhanced SYSOUT Support (ESS) Section */
/*-----------------------------------------------------------------*/
/* Enhanced System Affinity support section.                       */
/* This section contains the system names for all the systems      */
/* for which this job has affinity. The one exception is if it     */
/* has an affinity of ANY in which case the flag bit smf24aff is   */
/* on. Triplet Information:                                        */
/* offset (smf24osa) length (smf24lsa) number (smf24nsa)           */
/*-----------------------------------------------------------------*/
 IF smf24lsa  > 0 AND smf24nsa  > 0 Then do
    sas = smf24osa - 3
smf24ls4 = c2d(substr(x.i,sas,2))          /* Length of sysaff section  */
```

```
smf24san = c2d(substr(x.i,sas+2,4))     /* Number of system affinities*/
smf24ln4 = c2d(substr(x.i,sas+6,4))      /* Length of system name     */
smf24sac = substr(x.i,sas+10,smf24ln4)   /* Start of system aff. names*/
                                         /* The length of the smf24sac*/
                                         /* field is variable and     */
                                         /* depends on the amount of  */
                                         /* system affinity names     */
end /* End of Enhanced System Affinity suport section */
/*-----------------------------------------------------------------*/
/* Print offloaded list info                                       */
/*-----------------------------------------------------------------*/
 c24joff.1 = left(smf24jbn,8),                  /* Job name          */
             left(smf24jid,8),                  /* Original job id.  */
             right(listoff,4),                 /* List offload duration*/
             right(Date('N',smf24drd,'J'),11),  /*Date on reader - job*/
             left(smf24trd,12),                /* Time on reader - job */
             left(smf24scl,2),                 /* SYSOUT class        */
             left(smf24org,8),                 /* Origin node         */
             right(smf24cnt,5),                /* # rec. dumped-loaded */
             right(smf24rec,5),                /* Dataset record count */
             left(smf24srn,4),                 /* Remote name         */
             left(smf24fcb,4),                 /* Forms control buffer */
             left(smf24for,4),                 /* Forms overlay name  */
             left(smf24fls,4),                 /* Flash cartridge name */
             left(smf24prm,4),                 /* Print data set mode  */
             left(smf24ucs,4),                 /* Universal char. set  */
             left(smf24wid,6),                 /* Writer              */
             left(sut,12),                     /* Sysout status flag  */
             left(eoj,36)                      /* End of job status   */
 "EXECIO 1 DISKW REPORT(stem c24joff.)"
sut = left(' ',5,' '); eoj = left(' ',5,' ')
b80 = left(' ',5,' '); b40 = left(' ',5,' ')
b20 = left(' ',5,' ')
jf80= left(' ',5,' '); jf40 = left(' ',5,' ')
/*-----------------------------------------------------------------*/
/* Print Enhanced SYSOUT Support section data if there is any      */
/*-----------------------------------------------------------------*/
Select
  when smf24nsw > 0 Then do
   msg = left(' ',9,' ')||left('ESS data: ',10)||smf24tu
   call printline msg
  end
  otherwise nop
End
esser = left(' ',5,' ')
  end /*of  type = 24 */
dur = cross(smf24tm,first); t1 = smf(first); t2 = smf24tme
end /* i - reading file*/
/*-----------------------------------------------------------------*/
/* Print offload stats                                             */
```

```
/*-------------------------------------------------------------------*/
call printline "     "
call printline "Offload started at: "t1
call printline "          ended at: "t2
call printline "        time (sec): "dur
call printline "# lists offloaded : "i-1
call printline "Records offloaded : "Totrec
call printline "Offload data set  : "smf24dsn
/*-------------------------------------------------------------------*/
/* Close & free all allocated files                                  */
/*-------------------------------------------------------------------*/
 "EXECIO Ø DISKR SMF24  (FINIS"
 "EXECIO Ø DISKW REPORT (FINIS"
  say "JES2 Spool Offload report file ....."r24off
 "FREE FILE(SMF24 REPORT)"
exit

Printline:
/*-------------------------------------------------------------------*/
/* Print each report line                                            */
/*-------------------------------------------------------------------*/
  PARSE arg lineout1
   "EXECIO 1 DISKW REPORT (STEM lineout)"
   if rc \= Ø then
     do
       say "printline RC =" RC
       exit rc
     end    /* end of printline */
Return

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

CROSS: procedure
/*-------------------------------------------------------------------*/
/*  Cover the midnight crossover                                     */
/*-------------------------------------------------------------------*/
arg endtime,startime
  select
   when endtime >= startime then nop
```

```
   otherwise  endtime = endtime + 8640000
  end
diftm =  (endtime - startime)*0.01
return diftm
```

Mile Pekic
Systems Programmer (Serbia and Montenegro)

The *Update* family

In addition to *MVS Update*, the Xephon family of *Update* publications now includes *AIX Update*, *CICS Update*, *DB2 Update*, *MQ Update*, *RACF Update*, and *TCP/SNA Update*. Details of all of these can be found on the Xephon Web site at www.xephon.com.

BMC Software has announced new features in its change and configuration management solutions, including new reconciliation and reporting features.

BMC also announced new reconciliation and reporting capabilities for BMC Discovery Express and BMC Configuration Discovery that help companies become more efficient at validating planned changes and discovering unplanned, or rogue, changes. Several new rules have been added that are available out-of-the-box, and new integration between these discovery solutions also optimizes the reconciliation process. Both BMC Discovery Express and BMC Configuation Discovery are part of BMC's IT Discovery suite, which identifies all aspects of the IT environment and keeps the BMC Atrium CMDB continuously updated with actual configuration information.

For further information contact:
URL: www.bmc.com/products/proddocview/0,2832,19052_19429_21288009_116908,00.html.

* * *

Compuware has announced Version 3.2 of Abend-AID Fault Manager, which is designed to help IT organizations better manage the growing complexity of enterprise application environments.

The product combines mainframe and distributed fault diagnostic capabilities into a single solution. Additionally, it introduces integration with both HP OpenView Operations and Compuware Changepoint.

For further information contact:
URL: www.compuware.com/products/abendaid/faultmanager.htm.

* * *

21st Century Software has announced the inclusion of an enhanced audit capability within their DR/VFI product. The new feature aids in compliance with recent legislation such as Sarbanes-Oxley by providing an automated method of ensuring that all critical business data is protected and properly backed up. Audit information for all datasets, or files, monitored by the system can be interrogated to determine current status, identify use across multiple applications or users, as well as additional information, which is tracked and contained within the product's database. The new feature also provides online access to all audit information.

For further information contact:
URL: www.drvfi.com.

* * *

BluePhoenix Solutions has announced DBMSMigrator a fully automated converter for ADABAS/Natural applications to Java accessing a relational database management system. This helps companies migrate applications away from the proprietary ADABAS/Natural environments to modern, open technologies such as Java, J2EE, relational databases, and Web-enabled graphical user-interfaces. The migrated applications can then be extended more easily to meet new business requirements.

For further information contact:
URL: www.bphx.com.

* * *