March 2001

In this issue

3  Tidying generated HTML using REXX
9  Extracting system hold data from SMP/E SYSMODs
15 Dynamically allocating cartridges
23 Query Common Storage Utilization
35 On-line messages and codes revisited
64 Documenting dataset usage in jobs
72 MVS news

© Xephon plc 2001
Editor
Jaime Kaminski

Disclaimer
Readers are cautioned that, although the information in this journal is presented in good faith, neither Xephon nor the organizations or individuals that supplied information in this journal give any warranty or make any representations as to the accuracy of the material it contains. Neither Xephon nor the contributing organizations or individuals accept any liability of any kind howsoever arising out of the use of such material. Readers should satisfy themselves as to the correctness and relevance to their circumstances of all advice, information, code, JCL, EXECs, and other contents of this journal before making any use of it.

MVS Update on-line
Code from MVS Update can be downloaded from our Web site at http://www.xephon.com/mvsupdate.html; you will need the user-id shown on your address label.

Subscriptions and back-issues
A year’s subscription to MVS Update, comprising twelve monthly issues, costs £340.00 in the UK; $505.00 in the USA and Canada; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 1992 issue, are available separately to subscribers for £29.00 ($43.00) each including postage.
Tidying generated HTML using REXX

INTRODUCTION

While testing various functions in the Webserver environment, I put together a mini intranet. All was well at the start, I used simple HTML pages and generated these with a normal PC-based tool. The HTML pages were then exported to our OS/390 machine and all ran well.

As I progressed, I introduced various other elements into the intranet site. Important for us was access to DB2 tables and dynamic selection on an HTML page. One solution was for us to use IBM’s Net.Data product. Net.Data is a macro processor that executes as middleware on a Webserver machine including OS/390. Net.Data uses macros to create dynamic Web pages based on various factors including DB2 databases. A macro consists of two major parts – the declaration and the presentation. The declaration part contains the definition of functions and variables, the presentation part contains HTML blocks specifying the layout of the Web pages. Net.Data builds a Web page based on the functions and the HTML blocks. Each HTML block contains 95% normal HTML code – code that we can generate and then modify as required by hand. The problem that I faced was that the HTML code generated, that I needed to change, was very poorly formatted.

Therefore I wrote this REXX routine to format the generated HTML. Basically, leading spaces are stripped from each line and then newly indented. For each new HTML open tag, indentation is increased by one and the tag is placed on a new line. When a HTML close tag is encountered, the corresponding open tag indentation is used.

EXAMPLE

"DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 FINAL//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-8859-1">
<meta name="Generator" content="NetObjects Fusion 4.0.1 for Windows">
<title>
I developed the routine under TSO and then transferred and modified it to be called from the Shell (Unix System Services). The version here is the latest version (htmlneat.cmd), and when called needs the inputfile and outputfile as parameters. With a few modifications and use of JCL (example code in REXBATCH) it can be run in TSO (JES). The subroutine openandread needs the following code:

```
openandread:
  "EXECIO * DISKR filein (STEM htmlin. " /* read file into array */
return
```

And the subroutine writefile should then read:

```
writefile:
  "EXECIO * DISKW fileout (STEM htmlout. FINIS "
return
```

HTMLNEAT
/* REXX */
/* ******* Name:  HTMLNEAT                                            */
/* Procedure to reformat generated HTML code to aid readability      */
/* ************************************************************************/  
/* OS/390 Unix System Services Shell version */
/* ************************************************************************/  
/* Parameters: Inputfile Outputfile */
/* ************************************************************************/  
call openandread
if htmlin.Ø = Ø then /* empty input? */
do
  say "**** EMPTY INPUT FILE ****"
  say "**** PROCEDURE ABENDED ****"
return
end
call initialise
do i = 1 to htmlin.Ø / * line by line */
  do forever
    blankln = COMPARE(  , ,htmlin.i, ' ' ) / * skip blank lines */
    if blankln = Ø then i = i + 1
    else leave
  end
  if i > htmlin.Ø then leave / * if index exceeded */
  htmlin.i = STRIP(htmlin.i,"B") / * remove dead space */
  workline = htmlin.i
  posA = Ø
  posB = Ø
  posC = Ø
  do forever
    posA = POS("<",workline)
    posB = POS(">",workline)
    posC = POS("/",workline)
    if posA = Ø & posB = Ø then / * no tags */
      do
        outline = outline  workline
        outline = STRIP(outline,"B")
        leave
      end
    if posA = Ø then / * no left tag-bracket */
      do
        uptoB = SUBSTR(workline,1,posB) / * to output and written out */
        uptoB = STRIP(uptoB,"B") / * rest moved to workline */
        outline = outline  uptoB
        workline = SUBSTR(workline,posB)
        call writeline
      end
    if posB = Ø then / * no right tag-bracket */
      do
        afterA = SUBSTR(workline,posA) / * output */
        afterA = STRIP(afterA,"B")
        outline = outline  afterA
        workline = 
        leave
      end
    if posA < posB then / * New tag */
      do
        if posA = 1 then / * Tag in start position */
          do / * Complete tag on new line */
            call writeline
            uptoA = SUBSTR(workline,1,posB)
            afterA = SUBSTR(workline,posB+1)
            workline = afterA / * rest of line passed on */
            outline = outline  uptoA
            call writeline
          end
        else / * Tag not in start position */
          ...
do
    uptoA = SUBSTR(workline,1,posA - 1)
    afterA = SUBSTR(workline,posA)
    workline = afterA             /* New tag now in start pos. */
    outline = outline  uptoA      /* up to tag added to output */
    call writeline
end

else                                /* Left bracket of tag on      */
    do                                 /* previous line. Close tag */
        uptoB = SUBSTR(workline,1,posB) /* and write out            */
        afterB = SUBSTR(workline,posA)
        workline = afterB
        outline = outline  uptoB
        call writeline
    end
end

say "**** PROCEDURE ENDED ****"
"FREE DD(OUTPUT)"
return rc

/******************************************************************
/*                    subroutines                                */
/******************************************************************

/* initialize */
**************/
initialise:
    indcnt = Ø
    pnt = Ø
    j = Ø
    outline = '
    workline = '
    indent = '
    dummy = '
    switchleft = Ø
return

/* writeline */
**************
writeline:
    posAw = Ø
    posBw = Ø
    posCw = Ø
    posAw = POS("<",outline)
    posBw = POS(">",outline)
    posCw = POS("/",outline)
    blanko = COMPARE(,,outline,'')
if blanko ≠ Ø then
do
  if posAw > Ø then
    if posCw = posAw + 1 then call indentleft
    else call indentright
  indent = LEFT(dummy, indcnt, " ")
  j = j + 1
  htmlout.j = indent outline /* write indented line to */
  htmlout.j = outline        /* output array */
end
outline = ','
return

/* indentleft */
/************************/
indentleft:
switchleft = 1
posleft = POS("/", outline)
posright = POS(">", outline, posleft)
posblank = POS(" ", outline, posleft)
if posblank < posright & posblank > Ø then posright = posblank
searchword = SUBSTR(outline, posleft + 1, (posright - posleft) - 1)
do forever
  if pnt < 1 then
    do
      say "**** ERROR IN SEARCH ****"
      leave
    end
  if searcharray.pnt.1 = searchword then
    do
      indcnt = searcharray.pnt.2
      searcharray.pnt.1 = ','
      searcharray.pnt.2 = ','
      pnt = pnt - 1
      leave
    end
  searcharray.pnt.1 = ','
  searcharray.pnt.2 = ','
  pnt = pnt - 1
end
return

/* indentright */
/************************/
indentright:
/* store position of */
pnt = pnt + 1 /* open-tag */
if switchleft = Ø then indcnt = indcnt + 1
posleft = POS("<", outline)
posright = POS(">", outline, posleft)
posblank = POS(" ", outline, posleft)
if posright = Ø then posright = posblank
if posblank < posright & posblank > Ø then posright = posblank
searchword = SUBSTR(outline, posleft + 1, (posright - posleft) -1)
searcharray.pnt.1 = searchword
searcharray.pnt.2 = indcnt
switchleft = Ø
return

/* openandread */
/****************/
openandread:
parse arg inpath outpath .       /* path for input and output */
/* including filename       */
address syscall
"open (inpath)" o_rdonly ØØ Ø    /* open input readonly     */
fdin=retval
if fdin = -1 then
do
    say "unable to open file " inpath
    exit
end

"open (outpath)" o_creat+o trunc+o_wronly 755 /* open output file     */
/* create if needed     */
/* overwrite if exists  */
/* write-only          */
/* file attributes 755 */
fdout=retval
if fdout = -1 then
do
    say "unable to open file " outpath
    exit
end

address mvs "execio * diskr" fdin "(stem htmlin.)"
/* read file into array */
return

/* writefile */
/****************/
writefile:
address mvs "execio * diskw" fdout "(stem htmlout.)"
Return

//REXBATCH JOB , 'USERID'.
// MSGLEVEL=(1,1), MSGCLASS=X.
// CLASS=A, REGION=4M.
// NOTIFY=&SYSUID.
Extracting system hold data from SMP/E SYSMODs

INTRODUCTION
When applying SMP/E maintenance to MVS, OS/390 (or z/OS system as it soon will be) it is almost inevitable that you will encounter some ‘Exception SYSMODs (HOLD)’. SMP/E supports three types of exception data ie HOLDERROR, HOLDSYSTEM (internal and external) and HOLDUSER. This article refers only to HOLDSYSTEM exception data. If you are interested in learning about HOLDERROR and HOLDUSER exception data please refer to Chapter 2.8 HOLD MCS of OS/390 SMP/E Reference (SC28-1806).

The SYSTEM Reason IDs that are used by IBM are ACTION, AO, DELETE, DEP, DOC, EC, EXRF, FULLGEN, IOGEN, MSGSKEL, and MVSCP. As part of our maintenance strategy, we first run a mass APPLY CHECK for the required SOURCEIDs specifying
BYPASS(HOLDSYS) to bypass all of the above reason IDs. When the APPLY CHECK job has completed, SMP/E will produce GIM42001W messages in the SMPOUT output to indicate which SYSMODs have had their reason IDs bypassed. At this point we had to go through the somewhat laborious task of using the online SMP/E dialogs to look at the system hold data for each of the SYSMODs listed in the GIM42001W messages. We could then decide which of the HOLDs were important and had to be actioned and which could be ignored. It soon became obvious that there had to be a simpler way to process these HOLDs. This led to the evolution of a simple COBOL program, HOLDPRNT, to do the hard work for us. The HOLDPRNT program, compile/link JCL, and execution JCL are detailed below:

HOLDPRNT PROGRAM
IDENTIFICATION DIVISION.
PROGRAM-ID. HOLDPRNT.
INSTALLATION.
DATE-COMPILED.

* THIS PROGRAM READS IN A PTF NUMBER AND EXTRACTS AND
* PRINTS ITS SYSTEM HOLD DATA FROM THE SMPPTS.

ENVIRONMENT DIVISION.

CONFIGURATION SECTION.

SOURCE-COMPUTER. IBM-37Ø.
OBJECT-COMPUTER. IBM-37Ø.

INPUT-OUTPUT SECTION.

FILE-CONTROL.

SELECT INFILE ASSIGN TO UT-S-INFILE.

SELECT OUTFILE ASSIGN TO UT-S-OUTFILE.

DATA DIVISION.
FILE SECTION.
FD INFILE LABEL RECORDS ARE STANDARD RECORD CONTAINS 8Ø CHARACTERS BLOCK CONTAINS 0 RECORDS.
Ø1 INFILE-REC PIC X(8Ø).
FD OUTFILE LABEL RECORDS ARE STANDARD RECORD CONTAINS 8Ø CHARACTERS BLOCK CONTAINS 0 RECORDS.
WORKING-STORAGE SECTION.

OUTFILE-REC PIC X(80).

WS-INFILE-REC.
  WS-PLUS PIC XX.
  WS-PLUSTYPE.
    WS-PLUSTYPE-FIRST4 PIC X(4).
    FILLER PIC X.
  FILLER PIC X.
  WS-SYSMOD PIC X(7).
  FILLER PIC X(65).

WS-READ PIC X VALUE 'Y'.
  WS-FIRST-READ VALUE 'Y'.

WS-TYPE PIC X VALUE 'N'.
  WS-NOT-HOLDDATA VALUE 'N'.

WS-OPEN PIC X VALUE 'N'.
  WS-NOT-OPENED VALUE 'N'.
  WS-OPENED VALUE 'Y'.

WP-PRINT-LINE1.
  FILLER PIC X(21) VALUE 'HOLD DATA FOR SYSMOD'.
  WP-SYSMOD PIC X(7).
  FILLER PIC X(52) VALUE '.'.

WP-PRINT-LINE2 PIC X(80) VALUE SPACES.

PROCEDURE DIVISION.

OPENIT.

* OPEN INPUT DATASET
  OPEN INPUT INFILE.
  READIT.

* READ INPUT RECORD
  READ INFILE INTO WS-INFILE-REC
  AT END
  GO CLOSEIT.

* SET FIRST TIME FLAG
  IF WS-FIRST-READ
MOVE 'N' TO WS-READ
MOVE WS-SYSMOD TO WP-SYSMOD
GO READIT.

* SET SYSTEM HOLD RECORD FLAG

    IF  WS-PLUS = '++'
        IF  WS-PLUSTYPE = ' HOLD'
            OR  WS-PLUSTYPE-FIRST4 = 'HOLD'
                MOVE 'Y' TO WS-TYPE
        ELSE
            MOVE 'N' TO WS-TYPE.

* IGNORE IF NOT SYSTEM HOLD RECORD(S)

    IF  WS-NOT-HOLDDATA
        GO READIT.

* OPEN OUTPUT DATASET
* SET FLAG TO INDICATE THIS
* WRITE HEADER RECORD

    IF  WS-NOT-OPENED
        MOVE 'Y' TO WS-OPEN
        OPEN OUTPUT OUTFILE
        WRITE OUTFILE-REC FROM WP-PRINT-LINE1 AFTER PAGE
        WRITE OUTFILE-REC FROM WP-PRINT-LINE2 AFTER 1.

* WRITE SYSTEM HOLD RECORD
    WRITE OUTFILE-REC FROM WS-INFILE-REC.
        GO READIT.
    CLOSEIT.

* CLOSE INPUT DATASET

    CLOSE INFILE.

* CLOSE OUTPUT DATASET
    IF  WS-OPENED
        CLOSE OUTFILE.
    STOP RUN.

* END OF HOLDPRNT PROGRAM

**COMPIL**E/LINK JCL

    //U1300MP JOB 060103010,'COBOL COMP/LINK',MSGLEVEL=(1,1),MSGCLASS=X,
    //         CLASS=X,NOTIFY=U1300,REGION=8M
    //*
    //COBOL EXEC PGM=IGYCRCTL,PARM='DATA(24)'
    //STEPLIB DD DSN=COBOL370.SIGYCOMP,DISP=SHR
PROCESS

Simply repeat the //S1 EXEC PRINTIT,PTF=UW75232 as often as necessary to accommodate all the SYSMODs for which GIM42001W messages were issued. We find that the quickest way to do this is to use the following procedure:
1. Issue the SDSF ? action character against the APPLY CHECK job output to display a list of datasets for the output group.

2. Issue the SDSF SE action character against the SMPOUT dataset to invoke ISPF Edit.

3. Issue the EXC ALL primary command to hide all the lines.

4. Issue the F GIM42001W ALL primary command to find all occurrences of the GIM42001W message.

5. Issue the DEL ALL X primary command to delete all the excluded lines.

6. Cut and paste the SYSMOD IDs from the GIM42001W messages into the HOLDPRNT JCL.

7. SUB the job.

Do not worry, it is not really as time-consuming as it looks. Try it and see for yourself. When the job completes it will have produced each SYSMOD on a fresh page in the following format:

HOLD DATA FOR SYSMOD UW75232.

++ HOLD(UW75232) SYS FMID(HRM66Ø7) REASON(ACTION) DATE(ØØ315)
COMMENT
(****************************** A T T E N T I O N ****************************
* An IPL with CLPA must be performed to get this PTF active . *
******************************************************************************).

CONCLUSION

We certainly think it is much easier to read the exception data from a hardcopy rather than on-line. It also allows the hardcopies to be filed away for future reference. This can be useful when you wonder why some function is not ‘working as designed’ in the way it is documented in the manuals. Simply trawl through your hardcopies and you will probably find that one of the PTFs you applied has altered the processing! Hopefully you will have found this article useful and it will speed up the time-consuming process of keeping your system maintenance at a reasonably up-to-date level.
Dynamically allocating cartridges

INTRODUCTION
We have an application that archives records to cartridge which have been offloaded from an online system. As these records may later be retrieved, all records are catalogued in a KSDS, along with a number that forms the dataset name for the cartridge on which they are held. For example, record 01234567 may be located on the dataset ‘OFFL.CARSTO.CART123’, and so it would be held on the KSDS with ‘123’ associated with it. I wrote the following program, which can be called to enable the system to:

1. Allocate the current cartridge ‘DISP=MOD’, so that the latest records can be appended to it (the number of records that have been written to the cartridge is also held in the KSDS, so that we can determine when the cart is ‘full’, and we will then be able to switch to a new one).

2. Switch to a new cartridge when the current one is full, by calling the program to deallocate the current cartridge, followed by a further call for a new cartridge ‘DISP=(NEW,CATLG)’.

3. Allocate a particular cartridge when retrieving records (after having determined which cartridge is required by reading the KSDS). Following this the cart can be de-allocated followed by another allocate request for a different cartridge; this process can therefore be continued for any number of retrieval requests.

SOURCE
CARTDYN TITLE 'DYNAMICALLY ALLOCATE/DEALLOCATE CARTRIDGES'
***********************************************************************
* MODULE: CARTDYN *
* DESC: ACCORDING TO THE PARMS PASSED, USE SVC 99 TO *
* DYNAMICALLY ALLOCATE/DEALLOCATE DATASETS. *
* PARMS: AS FOLLOWS - *
* -NAME- -LENGTH- ————DESCRIPTION———
* DDNAME 8 DDNAME TO BE USED FOR ALLOC/DEALLOC. *
* CARSTNUM 3 CARSTO CARTRIDGE NUMBER. THIS WILL BE USED TO *

* CREATE A DATASET NAME IN THE FORM OF: * 'OFFL.CARSTO.CARTNNN'. * * FUNCTION 3 TYPE OF CALL: * * NEW - ALLOCATE THE DATASET NAME TO DDNAME * AS PASSED WITH 'DISP=(NEW,CATLG)'. * THIS CALL WOULD BE USED TO ALLOCATE * A NEW CART WHEN THE CURRENT ONE HAS * FILLED UP (WE WOULD MAKE A CALL TO * THIS PROGRAM FIRST, WITH A DEALLOCATE * REQUEST). * * OLD - ALLOCATE THE DATASET NAME TO DDNAME * AS PASSED WITH 'DISP=OLD'. THIS CALL * WOULD BE TO ALLOCATE A SPECIFIC CART * SO THAT WE CAN EXTRACT HISTORY INFO. * * MOD - ALLOCATE THE DATASET NAME TO DDNAME * AS PASSED WITH 'DISP=MOD'. THIS CALL * WOULD BE TO ENABLE US TO APPEND DATA * TO THE END OF THE CURRENT CARTRIDGE. * * DLC - DE-ALLOCATE THE DDNAME AS PASSED. * THIS CALL WOULD BE USED AFTER A 'MOD' * CALL, BUT BEFORE A 'NEW' CALL, SO THAT * WE CAN CHANGE TO A NEW CART IN THE * MIDDLE OF A JOB WHEN THE CURRENT ONE * HAS FILLED UP. * * ——————————————————————————————————————————————————————————————————— * * NOTES: * * THE S99FLAG2 FIELD VALUE S99WTUNT X'10' WILL ISSUE THE WAIT * OPTION IF NO DEVICES ARE AVAILABLE. * * PROGRAM IS AUTHORIZED. * * *********************************************************************** PRINT NOGEN * ——————————————————————————————————————————————————————————————————— * HOUSEKEEPING, ETC ... * ——————————————————————————————————————————————————————————————————— * CARTDYN CSECT BAKR R14,Ø SAVE CALLER DATA ON STACK LR R12,R15 GET BASE USING CARTDYN,R12 ADDRESSABILITY * ——————————————————————————————————————————————————————————————————— * LOCATE AND VERIFY PARMS PASSED TO US... * * L R2,Ø(R1) GET PARMS ADDRESS USING PARMS,R2 ADDRESSABILITY TO PARMS CLC FUNCTION,DLC DEALLOC REQUEST? BE DEALLOC YES..DON'T NEED CART NUMBER LA R1,3 LOOP COUNT LA R3,CART_NO POINT TO CART NUMBER LOOP1 DS ØH
CLI Ø(R3),C'Ø'  ENSURE CART_NO IS NUMERIC
KL BADCART
CLI Ø(R3),C'9'
BH BADCART
LA R3,1(R3)  POINT TO NEXT CHARACTER
CCT R1,LOOP1
MVC ATXT2DSN+16(3),CART_NO  MOVE CART NUM TO ALLOC PARMS
MVC ATXT1DDN,DDNAME  MOVE DDNAME TO ALLOC PARMS
CLC FUNCTION,NEW  DISP=(NEW,CATLG) REQUEST?
BE DISPNW  YES..
CLC FUNCTION,OLD  DISP=OLD REQUEST?
BE DISPOLD  YES..
CLC FUNCTION,MOD  DISP=MOD REQUEST?
BE DISPMOD  YES..
B INVFUNC  NO...INVALID REQUEST

* ———————————————————————————————————————————————————————————————————— *
* SET UP PARMS FOR SVC99 "DEALLOC" REQUEST... *
* ———————————————————————————————————————————————————————————————————— *

DEALLOC  DS ØH
WTO 'CARTDYN: DEALLOC',ROUTCDE=11  TRACKING
MVC DTXT1DDN,DDNAME  DDNAME FOR DEALLOCATION
LA R1,W99DPARM  R1 -> DEALLOCATION PARMS
B DOSVC99  GO AND DO SVC99...

* ———————————————————————————————————————————————————————————————————— *
* SET UP PARMS FOR SVC99 "(NEW,CATLG,DELETE)" REQUEST, BUT FIRST WE *
* CHECK TO SEE IF THE 'NEW' DATASET ALREADY EXISTS... IF SO, ABEND. *
* FOR RETURN CODES FROM THE 'LOCATE' MACRO, SEE "MVS/DFP SYSTEM PROG- *
* PROGRAMMING REFERENCE" UNDER "CATALOG MANAGEMENT MACROS". *
* ———————————————————————————————————————————————————————————————————— *

DISPNW  DS ØH
WTO 'CARTDYN: DISP=NEW',ROUTCDE=11  TRACKING
MVC DSN,ATXT2DSN  SET UP NAME FOR 'LOCATE' MACRO
LOCATE NAME  SEE IF ITS ALREADY CATALOGED
C R15,=F'Ø'  FOUND OK?
BE EXISTS  YES..CAN'T ALLOCATE ANOTHER ONE
C R15,=F'8'  NOT FOUND?
BNE BADLOC  NO...ERROR WITH 'LOCATE'
MVI WATXT4ST,X'Ø4'  PARM FOR 'DISP=(NEW'
MVI WATXT5DI,X'Ø2'  PARM FOR ',CATLG'
MVI WATXT6DI,X'Ø4'  PARM FOR ',DELETE)'
LA R1,W99APARM  R1 -> ALLOCATION PARMS
B DOSVC99  GO AND ISSUE SVC99

* ———————————————————————————————————————————————————————————————————— *
* SET UP PARMS FOR SVC99 "(OLD,KEEP,KEEP)" REQUEST... *
* ———————————————————————————————————————————————————————————————————— *

DISPOLD  DS ØH
WTO 'CARTDYN: DISP=OLD',ROUTCDE=11  TRACKING
MVI WATXT4ST,X'Ø1'  PARM FOR 'DISP=(OLD'
MVI WATXT5DI,X'Ø8'  PARM FOR ',KEEP'
MVI WATXT6DI,X'Ø8'  PARM FOR ',KEEP)'
LA R1,W99APARM  R1 -> ALLOCATION PARMS
B DOSVC99 GO AND ISSUE SVC99

* SET UP PARMS FOR SVC99 "(MOD,KEEP,KEEP)" REQUEST, BUT FIRST WE WILL *
* CHECK TO ENSURE THAT THE DATASET WE ARE MODDING EXISTS - IF NOT *
* THEN WE WILL ABEND... *

* ——————————————————————————————————————————————————————————————————————*

DISPMOD DS ØH

WTO 'CARTDYN: DISP=MOD',ROUTCDE=11 TRACKING
MVC DSN,ATXT2DSN SET UP NAME FOR 'LOCATE' MACRO

* LOCATE NAME SEE IF ITS ALREADY CATALOGED *

C R15,=F'8' DID WE GET RC=8? (IE NOT FOUND)
BE NOTFOUND YES..THAT'S AN ERROR
C R15,=F'Ø' FOUND OK?
BNE BADLOC NO...ERROR WITH 'LOCATE'
MVI WATXT4ST,X'Ø2' PARM FOR 'DISP=(MOD'
MVI WATXT5DI,X'Ø8' PARM FOR ',KEEP'
MVI WATXT6DI,X'Ø8' PARM FOR ',KEEP)'
LA R1,W99APARM R1 -> ALLOCATION PARMS
B DOSVC99 GO AND ISSUE SVC99

* ——————————————————————————————————————————————————————————————————————*

* ISSUE SVC99 TO PERFORM ALLOC/DEALLOC. FOR SVC99 ERROR CODES SEE THE *
* MANUAL "MVS/ESA AUTHORIZED ASSEMBLER PROGRAMMING GUIDE" UNDER THE *
* SECTION "REQUESTING DYNAMIC ALLOCATION FUNCTIONS".

* ——————————————————————————————————————————————————————————————————————*

DOSVC99 DS ØH
LR R3,R1 SAVE POINTER TO SVC99 PARMS
SVC 99 DO IT...
LTR R15,R15 ALLOC/DEALLOC OK?
BNZ BADSVC99 NO...

* ——————————————————————————————————————————————————————————————————————*

* RETURN TO CALLER... *

* ——————————————————————————————————————————————————————————————————————*

RETURN DS ØH
PR .

* ——————————————————————————————————————————————————————————————————————*

* ERROR ROUTINES...

* ——————————————————————————————————————————————————————————————————————*

INVFUNC DS ØH
MVC ERR1+39(3),FUNCTION MOVE FUNCTION TO MSG
WTO MF=(E,HEADER) DISPLAY HEADER
WTO MF=(E,ERR1) DISPLAY ERROR MESSAGE
WTO MF=(E,HEADER) DISPLAY HEADER
ABEND 1,DUMP *

BADCART DS ØH
MVC ERR2+54(3),CART_NO MOVE CART NUMBER TO MSG
WTO MF=(E,HEADER) DISPLAY HEADER
WTO MF=(E,ERR2) DISPLAY ERROR MESSAGE
WTO MF=(E,HEADER) DISPLAY HEADER
ABEND 2,DUMP

BADSVC99 DS ØH

* WE CHECK FOR SOME WELL-KNOWN ERRORS AND DISPLAY EXPLICIT MSGS
  MVC ERR3+31(3),FUNCTION MOVE FUNCTION TO MSG
  UNPK UNPKFLD(5),8(3,R3) UNPACK ERROR CODE + 1 BYTE
  TR UNPKFLD(4),TRTAB-24Ø TRANSLATE TO PRINTABLE HEX
  MVC ERR3+53(4),UNPKFLD MOVE HEX ERROR CODE TO MSG
  UNPK UNPKFLD(5),1Ø(3,R3) UNPACK INFO CODE + 1 BYTE
  TR UNPKFLD(4),TRTAB-24Ø TRANSLATE TO PRINTABLE HEX
  MVC ERR3+67(4),UNPKFLD MOVE HEX INFO CODE TO MSG
  MVC ERR3A+26(19),ATXT2DSN MOVE DSNAME TO MSG
  ST R15,STR15 SAVE R15 VALUE
  UNPK UNPKFLD(3),STR15+3(2) UNPACK R15 VALUE + 1 BYTE
  TR UNPKFLD(2),TRTAB-24Ø TRANSLATE TO PRINTABLE HEX
  MVC ERR3A+58(2),UNPKFLD MOVE HEX R15 VALUE TO MSG
  WTO MF=(E,HEADER) DISPLAY HEADER
  WTO MF=(E,ERR3) DISPLAY ERROR MESSAGE
  WTO MF=(E,ERR3A) DISPLAY ERROR MESSAGE
  CLC ERR3+53(4)=C'17Ø8' SPECIFIC ERROR? (NOT FOUND)
  BE ITSA17Ø8
  CLC ERR3+53(4)=C'Ø21Ø' SPECIFIC ERROR? (ENQUEUED)
  BE ITSAØ21Ø
  B DISPREST

* FOR AN 'Ø21Ø' ERROR (RESOURCE ALREADY ALLOCATED), WE WILL COME HERE *
* AND WAIT *ONCE* FOR 5 MINUTES BEFORE RETRYING. WE COULD CHANGE THE *
* FLAGS TO WAIT UNTIL THE RESOURCE BECOMES AVAILABLE, BUT THAT COULD *
* MEAN WAITING FOR AN UNKNOWN PERIOD OF TIME (EG IN CASE OF HARDWARE *
* ERROR, ETC...).

ITSAØ21Ø DS ØH
  CLI WAITFLG,C'Y' ALREADY BEEN HERE ONCE?
  BE DISPREST YES..JUST ABEND
  MVI WAITFLG,C'Y' NO..MAKE SURE ITS SET
  WTO MF=(E,ERR7) DISPLAY ERROR MESSAGE
  STIMER WAIT,BINTVL=BINTIM WAIT 3ØØ SECONDS (5 MINS)
  B DISPOLD GO BACK AND RETRY 'DISP=OLD'

ITSA17Ø8 DS ØH
  WTO MF=(E,ERR317Ø8) DISPLAY ERROR MESSAGE

DISPREST DS ØH
  WTO MF=(E,ERR3ABND) DISPLAY ERROR MESSAGE
  WTO MF=(E,HEADER) DISPLAY HEADER

ABEND 3,DUMP

BADLOC DS ØH
  ST R15,STR15 SAVE R15 VALUE
  UNPK UNPKFLD(3),STR15+3(2) UNPACK R15 VALUE + 1 BYTE
  TR UNPKFLD(2),TRTAB-24Ø TRANSLATE TO PRINTABLE HEX
  MVC ERR4+48(2),UNPKFLD MOVE HEX R15 VALUE TO MSG
  WTO MF=(E,HEADER) DISPLAY HEADER
  WTO MF=(E,ERR4) DISPLAY ERROR MESSAGE
  WTO MF=(E,HEADER) DISPLAY HEADER

**MESSAGES...**

*HEADER* WTO '=======================================================X
                      ==',ROUTCDE=11,MF=L

ERR1   WTO 'CARTDYN - INVALID FUNCTION PASSED "...", ABENDING...', ROUTCDE=11,MF=L

ERR2   WTO 'CARTDYN - INVALID CARSTO CARTRIDGE NUMBER PASSED "...", ABENDING...', ROUTCDE=11,MF=L

ERR3   WTO 'CARTDYN - ERROR IN SVC 99 "..." REQUEST. ERROR=X"...", INFO=X"...", R15 WAS X"...", ROUTCDE=11,MF=L

ERR3A  WTO 'CARTDYN - DSNAME WAS ".....................". R15 WAS X"...", ROUTCDE=11,MF=L

ERR317Ø8 WTO 'CARTDYN - RETURN CODE X"17Ø8" IMPLIES THAT THE DATASET DOES NOT EXIST.', ROUTCDE=11,MF=L

ERR3ABND WTO 'CARTDYN - PROGRAM ABENDING...', ROUTCDE=11,MF=L

ERR4   WTO 'CARTDYN - ERROR IN "LOCATE" MACRO - R15 = X"....", ABENDING...', ROUTCDE=11,MF=L

ERR5   WTO 'CARTDYN - A REQUEST HAS BEEN MADE TO ALLOCATE DATASET "X.....................".', ROUTCDE=11,MF=L

ERR5A  WTO 'CARTDYN - BUT THIS ALREADY EXISTS - ABENDING...', ROUTCDE=11,MF=L

ERR6   WTO 'CARTDYN - A "MOD" REQUEST HAS BEEN MADE FOR DATASET "..X.....................".', ROUTCDE=11,MF=L

ERR6A  WTO 'CARTDYN - BUT IT DOES NOT EXIST - ABENDING...', ROUTCDE=11,MF=L

ERR7   WTO 'CARTDYN - CART ALREADY IN USE, WAITING 5 MINUTES BEFORE RETRYING ONCE', ROUTCDE=11,MF=L

* SVC 99 ALLOC PARMS... *

DC C'*SVC 99'*
DS Ø0
W99APARM DC X'80',AL3(W99ARB) ALLOC PARMS (BITØ ON)
W99ARB DC AL1(2Ø),AL1(1),AL2(Ø) ALLOC REQUEST BLOCK
W99AERR DC AL2(Ø) ERROR CODE
W99INFO DC AL2(Ø) INFO. CODE
DC A(W99ATXT) ADDRESS OF TEXT POINTERS
DC A(Ø) RESERVED
DC X'10',AL3(Ø) FLAGS2 WAIT FOR UNITS
* TEXT UNIT POINTERS:
W99ATXT DC X'00',AL3(WATXT1) ...DDNAME
DC X'00',AL3(WATXT2) ...DSNAME
DC X'00',AL3(WATXT4) ...DATASET STATUS
DC X'00',AL3(WATXT5) ...DATASET NORMAL DISP
DC X'00',AL3(WATXT6) ...DATASET CONDITIONAL DISP
DC X'00',AL3(WATXT15) ...UNIT DESCRIPTION
DC X'00',AL3(WATXT23) ...RETPD SPECIFICATION
WATXT1 DC XL2'0001' TEXT UNIT 0001 = DDNAME
DC XL2'0001'
ATXT1DDL DC AL2(8) DDNAME LENGTH
ATXT1DDN DC CL8' ' DDNAME

* WATXT2 DC XL2'0002' TEXT UNIT 0002 = DSNAME
DC XL2'0001'
ATXT2DSL DC AL2(44) DDNAME LENGTH
ATXT2DSN DC CL44'OFFL.CARSTO.CARTNNN' DSNAME

* WATXT4 DC XL2'0004' TEXT UNIT 0004 = DATASET
DC XL2'0001' STATUS
DC XL2'0001' LENGTH=1
WATXT4ST DC XL1'00' STATUS

* WATXT5 DC XL2'0005' TEXT UNIT 0005 = NORMAL
DC XL2'0001' DISPOSITION
DC XL2'0001' LENGTH=1
WATXT5DI DC XL1'00' DISP

* WATXT6 DC XL2'0006' TEXT UNIT 0006 = CONDITIONAL
DC XL2'0001' DISPOSITION
DC XL2'0001' LENGTH=1
WATXT6DI DC XL1'00' DISP

* WATXT15 DC XL2'0015' TEXT UNIT 0015 = UNIT
DC XL2'0001'
DC XL2'0004' LENGTH=4
DC C'CART' UNIT='CART'

* THE FOLLOWING TEXT UNIT SHOULD ONLY HAVE AN AFFECT FOR 'DISP=NEW'
*
WATXT23 DC XL2'0023' TEXT UNIT 0023 = RETPD
DC XL2'0001'
DC XL2'0002' LENGTH=2
DC XL2'0E4C' RETPD=3660 DAYS

* SVC 99 DE-ALLOC PARMS...  *
*                                *
DS  OF
W99DPARM DC X'80',AL3(W99DRB) SVC99 DEALLOC PARMS
*
W99DRB DC AL1(20),AL1(2),AL2(0) DEALLOC REQUEST BLOCK
W99DERR DC AL2(0) ERROR CODE
W99DINFO DC AL2(0) INFO. CODE
  DC A(W99DTXT) ADDRESS OF TEXT POINTERS
  DC A(0) RESERVED
  DC A(0) FLAGS2
*
W99DTXT DC X'80',AL3(WDTXT1) TEXT UNIT POINTERS
*
WDTXT1 DC XL2'0001' TEXT UNIT 0002 = DEALLOC
  DC XL2'0001' KEY 0001 = BY DDNAME
DTXT1DDL DC AL2(8) DDNAME LENGTH
DTXT1DDN DC CL8' ' DDNAME
*                                *
* WORKING STORAGE...              *
*                                *
DLC      DC CL3'DLC'
NEW      DC CL3'NEW'
MOD      DC CL3'MOD'
OLD      DC CL3'OLD'
WAITFLG DC C'N'
UNPKFLD DS XL5
TRTAB    DC CL16'0123456789ABCDEF'
BINTIM   DC F'30000' 300 SECONDS
STRI5    DS F
NAME     CAMLST NAME,DSN,,CAMAREA
DSN      DC CL44' '
CAMAREA  DS ØD
  DS 265C
*                                *
* EQUATES, ETC...                *
*                                *
YREGS
LTORG
PARMS  DSECT
DDNAME DS CL8
CART_NO DS CL3
FUNCTION DS CL3
END

Grant Carson
Systems Programmer (UK) © Xephon 2001
Query Common Storage Utilization

QCSA
QCSA is an ISPF dialog to display the current size and usage of the MVS common storage areas, CSA, SQA, extended CSA, and extended SQA, as well as the amount of any SQA overflow. It provides ‘real-time’ statistics. This information is very important because adequate E/CSA and E/SQA storage are essential for efficient operation and a shortage of CSA can lead to eventual system failure. There is currently no easy way to display usage on OS/390, hence this ISPF dialog. The dialog has been run successfully on MVS Version 5 and OS/390 Versions 1 and 2.

VIRTUAL STORAGE OVERVIEW
The MVS 2GB virtual storage region can be divided into private areas and a common area. The common area contains system control blocks and programs and is subdivided into the following areas:

- Prefixed Storage Area (PSA)
- Common Service Area (CSA)
- Pageable Link Pack Area (PLPA)
- Fixed Link Pack Area (FLPA)
- Modified Link Pack Area (MLPA)
- System Queue Area (SQA)
- Nucleus.

All these areas are located below the 16MB line and all, except the PSA, have a corresponding extended area located above the 16MB line. A diagram showing the placement of these areas within the 2GB region is shown in Figure 1. The common areas are shown shaded. For simplification, in the rest of this article when a particular virtual storage area, eg CSA, is referenced, this includes both CSA and the extended CSA.
The size of each of these areas is fixed at a cold start IPL, that is an IPL where CLPA is specified to clear and reload the link pack area. The PSA has a fixed size of 4KB. The size of the link pack area (PLPA, FLPA, and MLPA) is determined by the total size of the modules located within the LPA list concatenation. The Nucleus size is determined by the modules in SYS1.NUCLEUS. The important thing about these areas – PSA, all LPA and nucleus – is that the space used does not change during system activity. This is one area where CSA and SQA differ – usage fluctuates depending on current system requirements. Another difference between CSA and SQA, and the other common storage areas is that the sizes of the CSA and SQA are directly controlled by the installation, rather than determined by the system. The size of CSA and SQA and their extended counterparts are specified by the CSA=(a,b) and SQA=(a,b) parameters in the IEASYS.xx member of PARMLIB, or during NIP. The amount remaining after rounding and after all the common areas have been defined is assigned to the private area. This means that defining a larger CSA will result in a smaller private area, as shown below.

<table>
<thead>
<tr>
<th>Area</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extended Private</td>
<td>2GB</td>
</tr>
<tr>
<td>Extended CSA</td>
<td></td>
</tr>
<tr>
<td>- EMLPA</td>
<td></td>
</tr>
<tr>
<td>Extended LPA</td>
<td></td>
</tr>
<tr>
<td>- EFLPA</td>
<td></td>
</tr>
<tr>
<td>- EPLPA</td>
<td></td>
</tr>
<tr>
<td>Extended SQA</td>
<td></td>
</tr>
<tr>
<td>Extended Nucleus</td>
<td></td>
</tr>
<tr>
<td>Nucleus</td>
<td></td>
</tr>
<tr>
<td>SQA</td>
<td></td>
</tr>
<tr>
<td>- PLPA</td>
<td></td>
</tr>
<tr>
<td>LPA</td>
<td></td>
</tr>
<tr>
<td>- FLPA</td>
<td></td>
</tr>
<tr>
<td>- MLPA</td>
<td></td>
</tr>
<tr>
<td>CSA</td>
<td></td>
</tr>
<tr>
<td>Private</td>
<td></td>
</tr>
<tr>
<td>PSA</td>
<td></td>
</tr>
<tr>
<td></td>
<td>16MB</td>
</tr>
</tbody>
</table>

* Areas analysed and displayed by QCSA dialog.

Shading identifies common areas.

Figure 1: MVS virtual storage layout
THE PROBLEM

The problem particular to CSA and SQA is that although the size of each is fixed at IPL time, the usage varies depending on system activity. It is important to specify these sizes accurately and yet there are many factors to be taken into account: too small and system failure can occur: too large and you will see a reduction in the size of the private areas, although a large CSA size can be used deliberately to reserve space for future LPA growth. Such growth could be hampered if users were allowed to obtain very large private areas. A large CSA specification effectively limits the maximum private area that a user job can acquire.

If the CSA and SQA sizes are not defined accurately, system performance will be severely degraded. For example, if fewer than 8 frames of SQA remain for allocation purposes, the system stops creating new address spaces. In addition, software errors can result in areas of SQA and CSA being allocated but never freed, leading to eventual exhaustion. This is often referred to as a ‘creeping storage’ problem. A quick search of the MVS problems database with the keywords ‘CSA exhaustion’ will show how common this problem is.

If the size allocated for SQA is too small or is used up very quickly, the system can convert unused CSA to SQA, but this will of course impact on your CSA requirements because the converted storage is no longer available for CSA requests. The overflow route is that when extended SQA is full, the system attempts to use extended CSA. When both extended SQA and extended CSA are used up, the system allocates space from SQA and CSA below 16 megabytes. Below-the-line SQA will overflow into below-the-line CSA. The same is not true for CSA and there is no capability to use SQA in case of a CSA shortage. Worse still is that a shortage of CSA can lead to a system failure. This is why it is important to be able to monitor the use of CSA and SQA.

QCSA COMMAND

QCSA, the ISPF dialog described in this article, is written in REXX and uses a single ISPF panel named QCSAWIN. The ISPF dialog is executed by the command ‘QCSA’. In ISPF, the result is a display similar to the one shown in Figure 2.
The display is updated by pressing <ENTER>. To end the dialog press <PF3>. By default under ISPF, the panel will be displayed in a pop-up window, but, if you would prefer to have the panel displayed on the full screen the NOPOP parameter can be used (see next section).

<table>
<thead>
<tr>
<th>Area</th>
<th>size</th>
<th>used</th>
<th>pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>CSA</td>
<td>4,912 K</td>
<td>1,684 K</td>
<td>34.29%</td>
</tr>
<tr>
<td>ECSA</td>
<td>15,116 K</td>
<td>12,090 K</td>
<td>79.98%</td>
</tr>
<tr>
<td>SQA</td>
<td>1,056 K</td>
<td>861 K</td>
<td>81.51%</td>
</tr>
<tr>
<td>ESQA</td>
<td>10,216 K</td>
<td>5,100 K</td>
<td>49.93%</td>
</tr>
</tbody>
</table>

E/SQA Overflow
- CSA converted to SQA: 0 K
- ECSA converted to ESQA: 0 K

**Figure 2: ISPF panel display – QCSAWIN**

<table>
<thead>
<tr>
<th>Common Storage Utilization</th>
</tr>
</thead>
<tbody>
<tr>
<td>Area</td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td>CSA</td>
</tr>
<tr>
<td>ECSA</td>
</tr>
<tr>
<td>SQA</td>
</tr>
<tr>
<td>ESQA</td>
</tr>
</tbody>
</table>

E/SQA Overflow
- CSA converted to SQA: 0 K
- ECSA converted to ESQA: 0 K

**Figure 3: TSO line display**
The fields in the panel should be self-explanatory, but for clarification – in the E/SQA OVERFLOW section, a non-zero value for ‘CSA converted to SQA’ indicates that SQA has been exhausted and overflowed into CSA. The figure shows how much CSA has been assigned to SQA usage and is therefore no longer available for CSA allocations.

Although the QCSA command is designed to run under ISPF and display an ISPF panel, it has been written to allow it to run under native TSO and in batch as well. In these cases the virtual storage layout will be presented in a simple line display, as shown in Figure 3. This is automatic depending on the environment in which the QCSA REXX EXEC is executed, but the line display can be requested under ISPF by using an optional parameter (see below).

The line display is also used if the display of the panel should fail for any reason. The QCSA command does not use LIBDEF but assumes that the panel is available in the active ISPPLIB concatenation.

PARAMETERS

No parameters are required to run the QCSA command, but there are several optional parameters that are used to indicate the type of display required:

- (none) – display is determined by the active environment.
- NOPOP – display in full screen panel, but not pop-up window.
- TSO – to show TSO line display, even under ISPF.
- DEBUG – show intermediate values.

Each of the parameters can be abbreviated to any length, down to a single character, and only one parameter can be specified at a time. An invalid parameter will cause a message to be issued and the dialog will process as if no parameter had been entered.

The NOPOP parameter is used under ISPF to display the results in a full-screen panel rather than an ISPF pop-up window. This is useful if you wish to do a screen print of the display.
The TSO parameter is used under ISPF to display the results in a TSO line display format rather than an ISPF panel. This form of display, as shown in Figure 3, is also used when a problem has been encountered with the panel, for example it is missing.

The DEBUG parameter is designed for use during modifications and testing. It uses the REXX SAY command to write intermediate values and calculations to the screen.

ABOUT THE CODE
The QCSA REXX EXEC uses the REXX STORAGE function to retrieve information from the CVT (Communications Vector Table) and GDA (Global Data Area). The CVT provides the address of the GDA and the GDA provides the necessary information about size and usage of the common areas. The GDA control block is used by VSM (Virtual Storage Manager) to contain information about system-related virtual storage and to anchor SQA and CSA queues.

NUMERIC DIGITS is set to 15 to cope with the large values used in calculations. Parameters are processed in the sub-routine VERPRM. Here the ABBREV function is used to permit any abbreviation of each parameter down to a single character and flags are set to indicate which parameter has been specified. Byte values are converted to kilobytes, and megabytes if appropriate, to improve readability. When run under ISPF, the dialog expects to find the panel in the active ISPPLIB concatenation. LIBDEF is not used but the EXEC can easily be changed to make use of this feature. The REXX RIGHT function is used for the TSO line display to right-align values and present the results in a clear, easily readable format. In the ISPF panel this is achieved by defining a panel attribute character with the JUST(RIGHT) statement.

The specification of panel attribute characters is a common cause of translation problems when moving between the PC ASCII environment and the mainframe and EBCDIC, but any problems should be easy to recognize and remedy. The QCSAWIN panel uses three attribute characters which are defined in the panel and one default character. After downloading the dialog, be prepared to review the panel before running for the first time.
REFERENCE

A description of the common storage areas, their function and how they are defined and used can be found in the *OS/390 MVS Initialization and Tuning Guide* SC28-1751. The *OS/390 MVS Initialization and Tuning Reference* SC28-1752 describes the CSA and SQA parameters in IEASYSxx. Details of the structure of the CVT and GDA and descriptions and offsets of the fields used by QCSA can be found in *OS/390 MVS Data Areas, Volume 1* (ABEP - DALT) SY28-1164 and *Volume 2* (DCCB - ITTCTE) SY28-1165. Information about REXX functions can be found in the *OS/390 TSO/E REXX Reference manual* SC28-1975. For ISPF, the manual *OS/390 ISPF Services Guide* SC28-1272 describes the ISPF services used in the QCSA REXX while panel definition is documented in *OS/390 ISPF Dialog Developer’s Guide and Reference* SC28-1273.

QCSA

```plaintext
/* +— — — — — — — — — — — — — — REXX — — — — — — — — — — — — — — — + */
/* | DOC:  QCSA command to display Common Storage usage            | */
/* |       Optional parameters:                                    | */
/* |         NOPOP - display in panel, but not pop-up window        | */
/* |         TSO   - display in TSO format                         | */
/* |         DEBUG - show intermediate values                      | */
/* |       Uses ISPF panel QCSAWIN (as pop-up window)              | */
/* +— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — –+ */
parse upper arg parm
parm = strip(parm)
if parm ¬= '' then call VERPRM parm
numeric digits(15)                          /* digits to 15          */
gda= d2x(c2d(storage(10,4))+560)            /* address GDA           */
gda= d2x(c2d(storage(gda,4)))

  csasz= d2x(x2d(gda)+112)                    /* CSA size              */
  csasz= c2d(storage(csasz,4))
  csaKB = KB(csasz)
  csaMB = MB(csasz)
  if debug = 'Y' then
    say 'CSA size:' csasz csaKB csaMB

  ecsasz= d2x(x2d(gda)+128)                   /* ECSA size             */
  ecsasz= c2d(storage(ecsasz,4))
  ecsaKB = KB(ecsasz)
  ecsaMB = MB(ecsasz)
  if debug = 'Y' then
    say 'ECSA size:' ecsasz ecsaKB ecsaMB
```

csare= d2x(x2d(gda)+132)   /* unallocated common */
csare= c2d(storage(csare,4))
csareKB = KB(csare)
csareMB = MB(csare)
if debug = 'Y' then
  say 'Unallocated common area (SQA+CSA):' csare csareKB csareMB

csacv= d2x(x2d(gda)+140)   /* ECSA size */
csacv= c2d(storage(csacv,4))
csacvKB = KB(csacv)
csacvMB = MB(csacv)
if debug = 'Y' then
  say 'CSA converted to SQA:' csacv csacvKB csacvMB

sqasz= d2x(x2d(gda)+148)   /* SQA size */
sqasz= c2d(storage(sqasz,4))
sqaKB = KB(sqasz)
sqaMB = MB(sqasz)
if debug = 'Y' then
  say 'SQA size:' sqasz sqaKB sqaMB

esqasz= d2x(x2d(gda)+156)   /* ESQA size */
esqasz= c2d(storage(esqasz,4))
esqaKB = KB(esqasz)
esqaMB = MB(esqasz)
if debug = 'Y' then
  say 'ESQA size:' esqasz esqaKB esqaMB

pvtsz= d2x(x2d(gda)+164)   /* pvt size */
pvtsz= c2d(storage(pvtsz,4))
pvtKB = KB(pvtsz)
pvtMB = MB(pvtsz)
if debug = 'Y' then
  say 'Private area size:' pvtsz pvtKB pvtMB

epvtsz= d2x(x2d(gda)+172)   /* epvt size */
epvtsz= c2d(storage(epvtsz,4))
epvtKB = KB(epvtsz)
epvtMB = MB(epvtsz)
if debug = 'Y' then
  say 'EPVT size:' epvtsz epvtKB epvtMB

call FINDUSE

if sysvar(sysispf) = 'ACTIVE' & tso = 'Y' then
  call POPUP
else call DISPLAY

exit
/* +— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —+ */
/* |  Subroutines                                                  | */
/* +— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —+ */
/* |   VERPRM:  verify parameters passed                           | */
/* |   FINDUSE: find storage usage figures                         | */
/* |   POPUP:   ISPF commands to display pop-up window             | */
/* |   DISPLAY: report used in TSO line mode                       | */
/* |   MB:      function to convert bytes to MegaBytes             | */
/* |   KB:      function to convert bytes to KiloBytes             | */
/* |   THOU:    function to insert commas as thousands separator   | */
/* +— — — — — — — — — — — — — — — — — — — — — — — — — — — — — — — —+ */
/*— — — — — — — — — — — — — — — —*/
/* Verify parameters passed      */
/*— — — — — — — — — — — — — — — —*/
VERPRM:                                     /* verify parameters     */
parse arg parm
select
  when abbrev('NOPOPUP', parm, 1) then NOPOP = 'Y'
  when abbrev('TSO', parm, 1) then TSO = 'Y'
  when abbrev('DEBUG', parm, 1) then DEBUG = 'Y'
  otherwise say parm 'invalid option - specify NOPOP, TSO or DEBUG'
end
return
/*— — — — — — — — — — — — — — — —*/
/* FIND usage figures            */
/*— — — — — — — — — — — — — — — —*/
FINDUSE:                                    /* calculate utilization */
csaal = d2x(x2d(gda)+432)                    /* CSA allocated         */
csaal = c2d(storage(csaal,4))
csaalKB = KB(csaal)
csaalMB = MB(csaal)
if debug = 'Y' then
  say 'CSA allocated:' csaal csaalKB csaalMB
ecsaal = d2x(x2d(gda)+436)                    /* ECSA allocated        */
ecsaal = c2d(storage(ecsaal,4))
cecsaalKB = KB(ecsaal)
cecsaalMB = MB(ecsaal)
if debug = 'Y' then
  say 'ECSA allocated:' ecsaal ecsaalKB ecsaalMB
sqaal = d2x(x2d(gda)+440)                    /* SQA allocated         */
sqaal = c2d(storage(sqaal,4))
sqaalKB = KB(sqaal)
sqaalMB = MB(sqaal)
if debug = 'Y' then
  say 'SQA allocated:' sqaal sqaalKB sqaalMB
esqaal = d2x(x2d(gda)+444)                    /* ESQA allocated        */
esqaal = c2d(storage(esqaal,4))
esqaalKB = KB(esqaal)
esqaalMB = MB(esqaal)
if debug = 'Y' then
    say 'ESQA allocated:' esqaal esqaalKB esqaalMB

csacv= d2x(x2d(gda)+448) /* CSA converted to SQA */
csacv= c2d(storage(csacv,4))
csacvKB = KB(csacv)
csacvMB = MB(csacv)
if debug = 'Y' then
    say 'CSA converted to SQA:' csacv csacvKB csacvMB

cesacv= d2x(x2d(gda)+448) /* ECSA converted to ESQA*/
cesacv= c2d(storage(ecsacv,4))
cesacvKB = KB(ecsacv)
cesacvMB = MB(ecsacv)
if debug = 'Y' then
    say 'ECSA converted to ESQA:' ecsacv ecsacvKB ecsacvMB

/* perform calculations */
csaupct = format(csaal*100/csasz,,2)
if debug = 'Y' then
    say 'CSA used' csaupct '%'

ecsaupt = format(ecsaal*100/ecsasz,,2)
if debug = 'Y' then
    say 'ECSA used' ecsaupt '%'

sqaupct = format(sqaal*100/sqasz,,2)
if debug = 'Y' then
    say 'SQA used' sqaupct '%'

esqaupct = format(esqaal*100/esqasz,,2)
if debug = 'Y' then
    say 'ESQA used' esqaupct '%'
return

/*— — — — — — — — — — — — — — — —*/
/* ISPF pop-up window */
/*— — — — — — — — — — — — — — — —*/
POPUP:
winrc = Ø
"ISPEXEC CONTROL ERRORS RETURN"
do while winrc = Ø
    if nopop ¬= 'Y' & pop ¬= 1 then
        do
            "ISPEXEC ADDPOP ROW(4) COLUMN(4)"
pop = 1
        end
        "ISPEXEC DISPLAY PANEL(QCSAWIN)"
    winrc = rc
    if winrc > 8 then call DISPLAY
    if winrc = 8 then
do
  "ISPEXEC REMPOP"
  pop = Ø
end
if winrc = Ø then call FINDUSE
end
return
/*— — — — — — — — — — — — — — — — */
/* Generate TSO line display       */
/*— — — — — — — — — — — — — — — — */
DISPLAY:
say 'Common Storage Utilisation'
say ''
say 'Area   size     used    pct'
card = 'CSA'
card = insert(right(csaKB,9),card,7)
card = insert(right(csaalKB,9),card,18)
card = insert(right(csaupct'%',6),card,30)
say card
card = 'ECSA'
card = insert(right(ecsaKB,9),card,7)
card = insert(right(ecsaalKB,9),card,18)
card = insert(right(ecsaupct'%',6),card,30)
say card
card = 'SQA'
card = insert(right(sqaKB,9),card,7)
card = insert(right(sqaalKB,9),card,18)
card = insert(right(sqaupct'%',6),card,30)
say card
card = 'ESQA'
card = insert(right(esqaKB,9),card,7)
card = insert(right(esqaalKB,9),card,18)
card = insert(right(esqaupct'%',6),card,30)
say card
say ''
say 'E/SQA Overflow'
say 'CSA converted to SQA  ' csacvKB
say 'ECSA converted to ESQA ' ecsacvKB
return
/*— — — — — — — — — — — — — — — — — — — — — */
/* Internal function:            */
/* Convert byte value into Megabytes */
/*— — — — — — — — — — — — — — — — — — — — — */
MB: procedure
  arg bytes
  x = format(bytes/1024/1024,,2)
x = x 'M'
return x
/*— — — — — — — — — — — — — — — — — — — — — */
/* Internal function:            */
/* Convert byte value into Kilobytes */
/*— — — — — — — — — — — — — — — — — — — — — */
KB: procedure                        /* convert bytes to KB */
arg bytes
x = format(bytes/1024,,0)
if length(strip(x)) > 3 then x = thou(x)  /* ,000 separator needed */
x = x 'K'
return x

/*— — — — — — — — — — — — — — — — — — — ——*/
/* Internal function:                         */
/* Insert comma as thousands separator */
/*— — — — — — — — — — — — — — — — — — — ——*/
THOU: procedure                        /* insert commas in num */
arg bignum
bignum = strip(bignum)
cpos = 3                                /* position for comma */
do until length(bignum)-cpos <= 0
  if length(bignum)-cpos <= 0 then return bignum
  bignum = insert(',',bignum,length(bignum)-cpos)
cpos = cpos + 4                          /* position of next comma*/
end
return bignum

QCSAWIN ISPF PANEL
)
)ATTR default(@+_
@ TYPE(TEXT) COLOR(TURQ) CAPS(OFF)
| TYPE(OUTPUT) COLOR(WHITE) JUST(RIGHT)
$ TYPE(TEXT) COLOR(WHITE) CAPS(OFF)
)BODY WINDOW(50,11)
@ Area         size        used         pct
@
@ CSA     |csaKB    +  |csaalKB +  |csaupct $%
@ ECSA    |ecsaKB   +  |ecsaalKB+  |ecsaupct$%
@ SQA     |sqaKB    +  |sqaalKB +  |sqaupct $%
@ ESQA    |esqaKB   +  |esqaalKB+  |esqaupct$%
@
@ E/SQA Overflow
@  CSA converted to SQA |csacvKB +
@  ECSA converted to ESQA |ecsacvKB +
)INIT
&ZWINTTL = 'Common Storage Usage'
)PROC
)END
/* Panel QCSAWIN - used by dialog QCSA

Moira Hunter (UK)
OS/390 Technical Specialist © Xephon 2001

© 2001. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
On-line messages and codes revisited

INTRODUCTION

In *MVS Update* Issue 153 (June 1999), Chan Tin Pui presented an original idea in his contribution *On-line explanation of OS/390 messages*. The article explains how messages and codes can be extracted from a printout of a Bookmanager manual, stored as members in a PDS and displayed on-line by a small ISPF application.

However, bright as the concept was, we found it rather hard to implement. The messages were isolated with the use of edit macros in the foreground, which makes it a rather cumbersome job to create the complete PDS. Furthermore, the supplied coding was not adapted to all the different formats that IBM uses in its *Messages and Codes* bookshelf. Because the message number was used as the PDS membername, the eight character limit for directory entries applied. A lot of messages – and this is often the case for ‘newer’ publications like the ones about Unix System Services (USS) – exceed this limit. Therefore they could not be included using the described algorithm.

The new version described below solves some of these drawbacks, besides filling the PDS in batch. Once the message is extracted, it works as follows. Suppose you are browsing the system log in SDSF and you put the cursor somewhere on ‘IEA630I’ in the line:

```
TSUØ2ØØ8 ØØØØØ29Ø IEA63ØI OPERATOR JEDSPSY NOW ACTIVE, SYSTEM=JEDSPV24, LU=
```

After pushing a pre-defined function key (PFK) the following window (MCBR00) pops up:

```
—<JED:SP Message Tool   jan@jedsp.com>—
Command ——>                      Scroll ——>

IEA63ØI OPERATOR JEDPSY NOW ACTIVE, SYSTEM=JEDSPV24, LU=

****************************** Top of Data ******************************
IEA63ØI OPERATOR opername NOW ACTIVE, SYSTEM=sysname, LU=unitname

Explanation: Multiple console support (MCS) activated an operator on a Time Sharing Option Extensions (TS0/E) terminal.
```
In the message text:

opername  The userid or console name of the TSO/E operator.

sysname   The name of the system to which the operator logged on.  The system name is specified on the SYSNAME parameter in the IEASYSxx parmlib member.

unitname  The logical unit name. This name identifies the physical location of the terminal that the TSO/E operator used to logon to the system.

Source:  Communications task (COMMTASK)
System Action:  The system continues processing.

Just pushing the PFK with a blank underneath the cursor would make panel MCREAD00 pop up:

—<JED:SP Message Tool>—
——<jan@jedsp.com>——

Message            ===>
System ABEND code  ===>
Wait state code    ===>
CICS ABEND code    ===>
DB2 SQL code (+/-) ===>  

PF3 to exit

Note: I am not aware of any copyright infringements if you store a modified copy of a publication you are licensed to use on your system. It would clearly be a different matter if you have no licence to the IBM BookManager files, in which case I do not see the use of employing it.

THIS VERSION

After reading the article, I realized that the advantages of having the capability to look-up a message or code while browsing job output or the system log could have a large impact on our personal production level in a daily systems programmer work environment. I pursued the idea and started developing this version on a Linux system in my free time. By the way, REXX runs fine on a Linux machine. It was a small effort to port the coding to the mainframe afterwards. Basically this new version offers the following extras:
• All messages and codes manuals from the table below can be loaded in a PDS in one batch run.

• Long message IDs are indexed. The long message is replaced by a generated member name and an index is kept in member ’$#*$#$#$’.

• DB2 SQL warning and error codes are included.

• Wait state codes are inventoried and prefixed by ‘W’. This is probably not very usable on the machine that produces the wait state, but this way at least the wait state code can be easily examined and debugged from another LPAR or system image.

• Information of no immediate value like multiple blank lines or copyright messages are filtered out.

• When the application is launched from a real message line – and not from its primary panel – the remainder of the original message line is displayed together with the message contents. This is very handy when one wants to know the explanation of a certain message that also contains return and reason codes. By simply putting the cursor on the line in the syslog and pressing a PF key, one can not only browse the message information, but also with a bit of luck – the return and reason codes are visible on the top of the browse window.

INSTALLATION NOTES

The first versions of the REXX program below which transforms a BookManager file to PDS members, were developed on a PC under Linux. It is to be regretted that, before I started coding, I did not check for differences between the printout of a manual created by BookManager OS/390 system and the one produced by BookManager PC. There exist numerous differences in the layout between the two listings. To use the batch program (MCGET) that converts a sequential BookManager list to PDS members the following steps must be carried out:

• Open the book in BookManager on a PC.
• Select File, click Print. Select ‘Entire file’ and ‘Print to file’.

• Choose a file name and folder and click ‘OK’. I chose filenames from the form <FLQ>.<product>MSG like in JEDSP.RACFMSG.

• After BookManager has finished printing upload the file to your mainframe. This is most easily done with an FTP product. Unlike IND$FILE-based file transfer, most FTP clients allow you to select more than one file. I created a directory with all the BookManager print files and could upload them all with just one click. Just for the record I used WS/FTP. I pre-allocated the message files with the following DCB:

```
DSORG=PS,RECFM=VB,LRECL=255,BLksz=27998,SPACE=(CYL(10,1))
```

• Pre-allocate also the final message PDS with the DCB:

```
DSORG=PO,RECFM=VB,LRECL=255,BLksz=27998,SPACE=(CYL(200,10,4000))
```

It is possible to use a PDSE instead of a PDS, but due to the static nature of this dataset – write once, read many – there are no benefits and it would only decrease the performance. With all the message datasets processed, the message PDS occupied 181 cylinders and 3,125 directory blocks to store 66,201 members. This is of course without ISPF statistics, in which case a directory block can describe 22 members on a standard 3390-3.

• Run MCGET and optionally delete the sequential message datasets.

CUSTOMIZATION NOTES
The easiest way to use the application is by hiding the ‘MCDISP’ command underneath a PF key:

• From an ISPF command line type ‘ISRDDN’ and give the command: ‘MEMBER ISPCMDS’. This will point you to the dataset that has the active ISPCMDS member.

• Go to ISPF 3.1 and rename the ISPCMDS member to ISXCMDS.

• In Option 3.9 give ISX as a prefix. ISPF will respond with a list of your current ISPCMDS settings. Put the cursor somewhere in the list and issue the ‘I’ line command. Fill in ‘MCDISP’ and
return using PF3.

- **MCDISP** is now on the command list. With the ‘U’ line command you can enter the required action, which is in this case:

```
SELECT CMD(%MCDISP) PARM(&ZPARM)
```

Put your cursor on ‘update’ and press enter.

- Rename the member back to ISPCMDS using ISPF 3.1 again. You have to restart ISPF before the new command table becomes active.

- Set a PF key to ‘MCDISP’. I use PF17 because this is the key above ‘FIND’.

Note that everything that could be specific to our site, for instance the job account field, is printed in italics.

**THE MESSAGE COLLECT PROGRAM MCGET**

IBM is not always consistent in the way it writes message books. The REXX program MCGET reflects this attitude. For a large number of books, exception rules are written. Apart from that, the program is quite simple. It swallows the manual as a whole into a stem variable and then starts to examine the lines one by one trying to locate the beginning and end of a particular message. The program could be easily adapted for other manuals. The JCL used to run the program is a regular IKJEFT01 to run an edit macro in batch. The editor interface is used to sort the index file so that the display program (MCDISP) can use a binary search mechanism to look-up a message with a name longer than 8 characters. Apart from the number of messages, the output of the program consists of the start and end time, the last message ID and the ID of the longest message. This is often the last one since the program cannot determine when the last message explanation is finished. By editing the last message in the PDS, obsolete information after the end of the message can be discarded.
JCL MCGET

The //SYSIN DD statement contains instream the names of the unloaded BookManager files. Remember that all formats are based on manuals that are printed using BookManager PC.

//JEDSPSYZ JOB (JAN), 'JAN DE DECKER',CLASS=A, MSGCLASS=X, NOTIFY=&SYSUID,
// REGION=OM, COND=(Ø, NE)
// *
// ** RUN A REXX IN AN ISPF ENVIRONMENT UNDER TSO/E
// *
// ** PART1 EXEC MCGET
// SYSIN DD *
JEDSP.ASMMSG
JEDSP.BDTMSG
JEDSP.CICSMMSG
JEDSP.DB2MSG
JEDSP.DCDSMSG
JEDSP.DFSMSG
JEDSP.DUMPMSG
JEDSP.GDDMSMSG
JEDSP.HCDMSG
JEDSP.ICFMSG
/*
//JEDSPSYZ JOB (JAN), 'JAN DE DECKER',CLASS=A, MSGCLASS=X, NOTIFY=&SYSUID,
// REGION=OM, COND=(Ø, NE)
// ** PART2 EXEC MCGET
// SYSIN DD *
JEDSP.IMSMSG
JEDSP.ISPFMSG
JEDSP.JESMSG
JEDSP.LEMSG
JEDSP.MVSCODES
JEDSP.MVSMSG1
JEDSP.MVSMSG2
JEDSP.MVSMSG3
JEDSP.MVSMSG4
JEDSP.MVSMSG5
/*
//JEDSPSYZ JOB (JAN), 'JAN DE DECKER',CLASS=A, MSGCLASS=X, NOTIFY=&SYSUID,
// REGION=OM, COND=(Ø, NE)
// ** PART3 EXEC MCGET
// SYSIN DD *
JEDSP.MQMSG
JEDSP.NCPMSG
JEDSP.NETVMSG
JEDSP.NPMMMSG
JEDSP.OEMSG
JEDSP.OPCAMSMSG
JEDSP.PSFMSG
JEDSP.RACFMSG
The JCL is split into four jobs because I encountered some real storage shortage problems while trying to do everything in one job. This is caused by the large number of dynamic allocations but could be no problem in another hardware environment. The procedure that is called by the JCL:

```
//MCGET    PROC
//SØ       EXEC PGM=IKJEFTØ1,PARM='ISPSTART CMD(%MCGET JEDSP.MSG)'
//SYSEXEC  DD DISP=SHR,DSN=JEDSP.L.REXX
//SYSTSIN  DD DUMMY
//SYSTSPRT DD SYSOUT=* 
//ISPPPLIB DD DSN=ISP.SISPPENU,DISP=SHR
//ISPMLIB  DD DSN=ISP.SISPMENU,DISP=SHR
//ISPSLIB  DD DSN=ISP.SISPSENU,DISP=SHR
//ISPTLIB  DD DSN=ISP.SISPTENU,DISP=SHR
//ISPFLIST DD SYSOUT=* 
//ISPFLOG  DD SYSOUT=* 
//ISPPROF DD DSN=&&PROF,
//    DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//    UNIT=SYSDA,
//    SPACE=(TRK,(5,1,10))
//ISPCTL1  DD UNIT=SYSDA,
//    DCB=(RECFM=FB,LRECL=80,BLKSIZE=0),
//    SPACE=(TRK,(5,1))

REXX MCGET

/* Rexx to produce a PDS with messages from Bookmanager message publications. The format is dependent on the CD-ROM printed version and not on an OS/390 listing. The message datasets to process are read from SYSIN. The output is written to the file specified in the parameter field. Long messages are written as $#$xxxxx. The relationship between $#$xxxxx and the actual message is kept in the member $#$#$#$#.
MCGET creates a file that is later used by MCDISP (ISPF message browse utility). To sort the $#$#$#$# member, a macro (MCSORT) is
used.
IBM is not very consistent in its layout of message manuals, hence the large number of exceptions in the coding.
Possible enhancements:
* DB2 - add codes: ØØC1......
* IMS - add abend codes
* NCP - add sense codes

*/
arg outputfile
address "TSO"
call get_long_msg
call get_input_files
/*
Following characters are not allowed in member names. Control is in stow_msg.
*/
invalid = '/)(,.;+-|@#%=?¶*|1'
total_message_count = Ø
prefixing = 'OFF'
do i_x = 1 to files.Ø
   longest_msg_l = Ø
   file_count = Ø
   say 'Treating file:' files.i_x left('-', 2Ø)
say '   —— Processing started: ' date() 'at' time()
   if read_bkmngr(files.i_x) <> Ø then iterate
call getmsgid
   last_msg = ''
do i = 1 to msgid.Ø
      first = i
      last = i + 1
      if icf = 'ON' then do
         first = msgline.first
         last = msgline.last - 7
      end
      else do
         first = msgline.first + 1
         last = msgline.last - 1
      end
call get_msg
call stow_msg
say '   —— Last message:       ' last_msg
say '   —— Number of messages: ' file_count
say '   —— Longest message:    ' longest_msg_l '('longest_msg_n')'
say '   —— Processing ended:   ' date() 'at' time()
end
call Check_and_store_prefix
say ' ' say '
say ' ' say '
say ' ' say '
say ' ' say ' ————————————'
say 'Total number of messages:' total_message_count
say ' ————————————'

exit

*——————————————————————————————————-
Read_bkmngr: Read the text file created by bookmanager in the stem variable inputrec.
*——————————————————————————————————*/

Read_bkmngr:

arg inputfile
"ALLOC DD(#TEMP#) DSN('"inputfile"') SHR REUSE"
if rc <> Ø then do
  say 'Allocation of' inputfile 'failed.'
  say 'Processing continues with next file.'
  return(8)
end
"EXECIO * DISKR #TEMP# (STEM inputrec. FINIS)"
if rc <> Ø then do
  say 'Read of' inputfile 'failed.'
  say 'Processing continues with next file.'
  return(8)
end
"FREE DD(#TEMP#)"
if rc <> Ø then do
  say 'Free of' inputfile 'failed.'
  say 'Processing continues with next file.'
  return(8)
end
return(Ø)

*————————————————————————————————————————————————————————————————————
Get_msgid: Scan the input file and extract all messagenumbers. A messagenumber is recognized as a string of 8 or 9 characters coming immediately underneath the title of the book. The messagenumbers are stored in the stem variable msgid. The corresponding line numbers of the inputfile in msgline. Counters are kept in msid.Ø and msgline.Ø.
*————————————————————————————————————————————————————————————————————*/

Get_msgid:
msgid. = ''
msgline. = ''
j = Ø
call set_special_off
do i = 1 to inputrec.Ø
  call check_special(inputrec.i)
  inputrec.i = translate(inputrec.i, ' ', 'FA BA BB OC 37')
  if ims = 'ON' & pos('o', inputrec.i) = 2 then do
    inputrec.i = overlay(' ', inputrec.i, 2, 1)
  end
  select
    when bdt = 'ON' & pos('BDT', inputrec.i) = 4 then do
      if word(strip(inputrec.i), 1) = msgid.j then iterate
      j = j + 1
msgid.j = word(strip(inputrec.i), 1)
msgline.j = i - 1
end
when dfs = 'ON' then do
dfs_prefix = left(strip(strip(inputrec.i, 'B', '.')), 3)
if dfs_prefix = 'IOE' then do
  p = i - 1
  if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
    j = j + 1
    msgid.j = word(strip(strip(inputrec.i, 'B', '.')), 1)
    msgline.j = i - 1
  end
end
when icf = 'ON' then do
  if pos('=-=-=-=-=-=-=-', strip(inputrec.i)) <> Ø then do
    p = i - 1
    icf_msg1 = word(strip(inputrec.p), 2)
    if length(icf_msg1) < 6 then iterate
    q = i + 5
    icf_msg2 = word(strip(inputrec.q), 1)
    if icf_msg1 = icf_msg2 then do
      j = j + 1
      msgid.j = icf_msg1
      msgline.j = q
      iterate
    end
  end
end
when ispf = 'ON' then do
  inputrec.i = translate(inputrec.i, ' ', '6A'x)
  inputrec.i = translate(inputrec.i, ' ', '6D'x)
ispf_prefix = left(strip(inputrec.i), 3)
if ispf_prefix = 'ISP' | ,
ispf_prefix = 'ISR' | ,
ispf_prefix = 'FLM' then do
  p = i - 2
  if pos(left('+-', 50, '-'), strip(inputrec.p)) <> Ø then do
    j = j + 1
    msgid.j = word(strip(inputrec.i), 1)
    msgline.j = i - 1
  end
end
when ncp = 'ON' then do
  ncp_prefix = left(strip(strip(inputrec.i, 'B', '.')), 3)
if ncp_prefix = 'CWA' | ,
ncp_prefix = 'DSJ' | ,
ncp_prefix = 'ICN' | ,
ncp_prefix = 'IFL' | ,
ncp_prefix = 'IFU' | ,
ncp_prefix = 'IFV' | ,
ncp_prefix = 'IFW' |
ncp_prefix = 'IFZ' |
ncp_prefix = 'IHR' then do
  p = i - 1
  if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
    j = j + 1
    msgid.j = word(strip(strip(inputrec.i, 'B', '.'), 1))
    msgline.j = i - 1
  end
end

when npm = 'ON' then do
  npm_prefix = left(strip(strip(inputrec.i, 'B', '.'), 3))
  if npm_prefix = 'FNW' then do
    p = i - 1
    if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
      j = j + 1
      msgid.j = word(strip(strip(inputrec.i, 'B', '.'), 1))
      msgline.j = i - 1
    end
  end
end

when psf = 'ON' then do
  psf_prefix = left(strip(strip(inputrec.i, 'B', '.'), 3))
  if psf_prefix = 'APS' then do
    p = i - 1
    if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
      j = j + 1
      msgid.j = word(strip(strip(inputrec.i, 'B', '.'), 1))
      msgline.j = i - 1
    end
  end
end

when rmf = 'ON' then do
  rmf_prefix = left(strip(strip(inputrec.i, 'B', '.'), 3))
  if rmf_prefix = 'ERB' | rmf_prefix = 'GPM' then do
    p = i - 1
    if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
      j = j + 1
      msgid.j = word(strip(strip(inputrec.i, 'B', '.'), 1))
      msgline.j = i - 1
    end
  end
end

when tsoe = 'ON' then do
  tsoe_prefix = left(strip(strip(inputrec.i, 'B', '.'), 3))
  if tsoe_prefix = 'ADF' |
    tsoe_prefix = 'CHS' |
    tsoe_prefix = 'COF' |
    tsoe_prefix = 'ICQ' |
    tsoe_prefix = 'IDY' |
tsoe_prefix = 'IKJ' | ,
tsoe_prefix = 'INM' | ,
tsoe_prefix = 'IRX' then do
  p = i - 1
  if pos(left('-', 50, '-'), inputrec.p) <> Ø then do
    j = j + 1
   msgid.j = word(strip(strip(inputrec.i, 'B', '.'), 1))
    msgline.j = i - 1
  end
end
end
when header(inputrec.i) then check_next = 'ON'
when check_next = 'ON' then do
  number_prefix = 'ON' then do
    leave
  end
  if abbrev(inputrec.i, left('', 30)) & ,
     length(strip(inputrec.i)) < 11 then do
    /* strip of blanks */
    /* strip of '.' as in some ADR messages */
    /* strip of '|' as in some CICS messages */
    /* strip of '+' as in some CICS messages */
    /* strip of '#' as in some CICS messages */
    inputrec.i = strip(inputrec.i)
    inputrec.i = strip(inputrec.i, 'B', '.')
    inputrec.i = strip(inputrec.i, 'B', '|')
    inputrec.i = strip(inputrec.i, 'B', '#')
    if db2 = 'OFF' then inputrec.i = strip(inputrec.i, 'B', '+')
    select
    /* check multipage message */
    when strip(word(inputrec.i, 1)) = msgid.j then do
      check_next = 'OFF'
      iterate
    end
    /* check Short titles that are no messages */
    /* If only a-z or A-Z characters return if not CICS */
  when datatype(inputrec.i, M) then do
    if cics = 'OFF' & number_prefix = 'OFF' then do
      check_next = 'OFF'
      iterate
    end
  when gddm = 'ON' then do
    /* see gddm action and severity codes */
    gddm_code = right(inputrec.i, 2)
    if gddm_code = 'I' | ,
      gddm_code = 'W' | ,
      gddm_code = 'A' | ,
      gddm_code = 'E' | ,
      gddm_code = 'S' | ,
gddm_code = ' U' then do
    inputrec.i = word(inputrec.i, 1)
end
when cics = 'ON' then do
    /* see cics action and severity codes */
    cics_code = right(inputrec.i, 3)
    if cics_code = ' A' | ,
        cics_code = ' D' | ,
        cics_code = ' E' | ,
        cics_code = ' I' | ,
        cics_code = ' W' | ,
        cics_code = ' S' then do
        inputrec.i = word(inputrec.i, 1)
    end
    otherwise nop
end
j = j + 1
msgid.j = inputrec.i
msgline.j = i
end
check_next = 'OFF'
end
otherwise nop
end
end
msgid.Ø = j
j = j + 1
msgline.j = i
return
/*————————————————————————————————————————————————————————————————————
Get_msg: I indexes the msgid stem variable that contains the message identification. First is the first line in the input file that has to be dealt with, last points to the next message description. In between there can be other chapters. This procedure skips till it finds the message id at the front of an input line followed by something else (the message itself). It copies the input lines to msgcontent. while skipping headers and footers. Empty lines are discarded if there are multiple ones after another. The last line is deleted if it is a blank line. msgcontents.Ø contains the number of relevant lines in msgcontents.
————————————————————————————————————————————————————————————————————*/
Get_msg:
msgcontent. = ''
msglines = Ø
if bdt = 'ON' then skip = 'OFF'
else skip = 'ON'
blankline = Ø
/* Convert long messages to short version */

do j = first to last
   inputline = strip(inputrec.j)
   inputline = strip(inputline, 'B', '_')
   inputline = strip(inputline, 'B', '|')
   inputline = strip(inputline, 'B', ':')
   if db2 = 'OFF' then inputline = strip(inputline, 'B', '+')
   inputline = strip(inputline)
   if pos(left('-', 4Ø, '-'), inputline) <> Ø then iterate
   if pos(left('+-', 4Ø, '-'), inputline) <> Ø then iterate
   if abbrev(inputline, msgid.i) then skip = 'OFF'
   if skip = 'ON' then iterate
   if header(inputrec.j) then do
      msglines = msglines - 2
      next_line = j + 1
      if inputrec.next_line = strip(msgid.i) then do
         j = j + 2
         iterate
      end
   else leave
   end
   if bdt = 'ON' | ispf = 'ON' | ncp = 'ON' | npm = 'ON' | rmf = 'ON' then do
      if abbrev(inputline, 'Copyright') then do
         msglines = msglines - 1
         j = j + 4
         iterate
      end
   end
   if blankline = Ø & length(inputline) = Ø then blankline = j
   else if blankline <> Ø & length(inputline) = Ø then iterate
   else blankline = Ø
   msglines = msglines + 1
   msgcontent.msglines = strip(inputrec.j, 'T', ' ')
end
   if length(strip(msgcontent.msglines)) = Ø then do
      msgcontent.Ø = msglines - 1
   end
   else msgcontent.Ø = msglines
return
/*————————————————————————————————————————————————————————————————————
Stow_msg: Stows the message in a PDS.
————————————————————————————————————————————————————————————————————*/

stow_msg:
select
/*
JES2 messages start with '$'
*/
when abbrev(msgid.i, '$') then nop
when abbrev(msgid.i, '+') | abbrev(msgid.i, '-') then do
/*
DB2 codes: +xxx and -xxx changed to $#Pxxx or $#Mxxx

msgid.i = '$#' || translate(msgid.i, 'PM', '+-')
end

If an invalid character is found, return.

When verify(msgid.i, invalid, M) <> Ø then return
when msgcontent.Ø <= Ø then do
    /*
    Ignore empty messages but log them.
    */
    say 'Empty message:' msgid.i
    return
end

When datatype(msgid.i, M) then do
    /*
    If only a-z or A-Z characters return if not cics or wait states.
    */
    if cics = 'OFF' & number_prefix = 'OFF' then return
    if length(msgid.i) > 4 then return
end

When datatype(left(msgid.i, 1), M) = Ø & ,
    number_prefix = 'OFF' then do
    /*
    If not +, -, $ first character must be a-z or A-Z.
    */
    return
end
otherwise nop
end

If length(msgid.i) > 8 then do
    prefixing = 'ON'
    prefix_c = prefix_c + 1
    prefix.prefix_c = '$$'$right(prefix_c, 5, 'Ø') msgid.i
    msgid.i = '$$'$right(prefix_c, 5, 'Ø')
end

total_message_count = total_message_count + 1
file_count = file_count + 1

If number_prefix = 'ON' then do
    if wait_state_codes = 'OFF' then msgid.i = 'S' || msgid.i
    else msgid.i = 'W' || msgid.i
    if msgid.i = 'SFnn' then wait_state_codes = 'ON'
end

msg_length = Ø
do j_x = 1 to msgcontent.Ø
    msg_length = msg_length + length(msgcontent.j_x)
end
If msg_length > longest_msg_l then do
    longest_msg_l = msg_length
    longest_msg_n = msgid.i
end
"ALLOC DD(#TEMP#) DSN("outputfile"("msgid.i")') SHR"
if rc <> Ø then say 'Error allocating:' msgid.i 'in' outputfile msgid.i
"EXECIO" msgcontent.Ø "DISKW #TEMP# (STEM msgcontent. FINIS)"
if rc <> Ø then do
  say 'Error in EXECIO:' msgid.i 'in' outputfile
  exit(8)
end
else last_msg = msgid.i
"FREE DD(#TEMP#)"
return

/*————————————————————————————————————————————————————————————————————
Check_and_store_prefix: Write all the changed prefixes to a file.  
Sort the prefix member on message name.  
————————————————————————————————————————————————————————————————————*/
Check_and_store_prefix:
if prefixing = 'OFF' then return
prefix.Ø = prefix_c
"ALLOC DD(#TEMP#) DSN("outputfile"($$$#$#)) SHR"
if rc <> Ø then do
  say 'Error allocating prefix member $$$#$# in' outputfile
  exit(8)
end
"EXECIO" prefix.Ø "DISKW #TEMP# (STEM prefix. FINIS)"
if rc <> Ø then do
  say 'Error writing prefix member $$$#$# in' outputfile
  exit(8)
end
"FREE DD(#TEMP#)"
address "ISPEXEC"
"EDIT DATASET('"outputfile"($$$#$#)) MACRO(MCSORT)"
if rc <> Ø then do
  say 'Error sorting prefix member $$$#$# in' outputfile
  exit(8)
end
return

/*————————————————————————————————————————————————————————————————————
Header: Check a passed string to see if it is a header. Return one if this is the case, zero otherwise
————————————————————————————————————————————————————————————————————*/
header:
parse arg h_line
ispf_header = left("+-", 5Ø, "-")
if pos("BDT Messages and Codes", h_line) = 33
  (pos(’CICS’, h_line) = 22 & pos(’Messages and Codes’, h_line) = 36)
  ,
  pos("DB2", h_line) = 2Ø & pos("Messages", h_line) = 38 
  ,
  pos("DFS Messages and Codes", h_line) = 33 
  ,
  (pos("GDDM", h_line) = 28 & pos("Messages", h_line) = 38) 
  ,
  pos("HCD Messages", h_line) = 39 
  ,
Get_long_msg: Try to read the member $#$#$#$# in the output message PDS. Set the number of the next long message accordingly.

Get_long_msg:
index_exists = SYSDSN(""outputfile"($#$#$#$")"
if index_exists = 'OK' then do
   "ALLOC DD(#TEMP#) DSN('"outputfile"($#$#$#$")') SHR"
   "EXECIO * DISKR #TEMP# (STEM prefix. FINIS)"
   prefix_c = prefix.Ø
   "FREE DD(#TEMP#)"
end
else do
   prefix_c = Ø
   prefix. = ''
   prefix.Ø = Ø
end
return
/*————————————————————————————————————————————————————————————————————*/
Get_input_files: Read the SYSIN DD. This contains all the filenames that will be treated.

Get_input_files:

files. = ''
"EXECIO * DISKR SYSIN (STEM files. FINIS)"
if rc <> Ø then do
    say 'Read of the names of the Bookmanager inputfiles failed.'
    say 'Check SYSIN. Processing terminates.'
    exit(8)
end
do i = 1 to files.Ø
    files.i = strip(files.i)
end
return

Set_special_off: Sets all the variables that indicate a manual with special treatment to 'OFF'.

Set_special_off:
parse value 'OFF' with book_check 1 bdt 1 cics 1 db2 1 dfs 1 gddm ,
    1 icf 1 ims 1 ispf 1 ncp 1 number_prefix ,
    1 rmf 1 tsoe 1 npm 1 psf 1 wait_state_codes
return

Check_special: Check for a manual that has a special layout and set the variables dependently.

Check_special:
parse arg s_line
select
    when book_check = 'ON' then return
    when pos('BDT Messages and Codes', s_line) = 33 then do
        bdt = 'ON'
        book_check = 'ON'
    end
    when (pos('CICS', s_line) = 22 & ,
        pos('Messages and Codes', s_line) = 36) then do
        cics = 'ON'
        book_check = 'ON'
    end
    when pos("DB2", s_line) = 2Ø & ,
        pos("Messages", s_line) = 38 then do
        db2 = 'ON'
        book_check = 'ON'
    end
    when pos("DFS Messages and Codes", s_line) = 33 then do
        dfs = 'ON'
book_check = 'ON'
end
when pos('GDDM', s_line) = 28 & ,
   pos('Messages', s_line) = 38 then do
gddm = 'ON'
book_check = 'ON'
end
when pos("IMS/ESA", s_line) = 23 & ,
   pos("Messages and Codes", s_line) = 34 then do
ims = 'ON'
book_check = 'ON'
end
when pos('Integrated Cryptographic', s_line) = 9 then do
icf = 'ON'
book_check = 'ON'
end
when pos('ISPF Messages and Codes', s_line) = 33 then do
ispf = 'ON'
book_check = 'ON'
end
when (pos('NCP', s_line) = 15 & ,
       pos('Messages and Codes', s_line) = 42) then do
ncp = 'ON'
book_check = 'ON'
end
when pos('NPM Messages and Codes', s_line) = 27 then do
npm = 'ON'
book_check = 'ON'
end
when pos('Print Services Facility/MVS', s_line) = 2 then do
psf = 'ON'
book_check = 'ON'
end
when pos('RMF Messages and Codes', s_line) = 33 then do
rmf = 'ON'
book_check = 'ON'
end
when pos('MVS System Codes', s_line) = 37 then do
   if bdt = 'OFF' & psf = 'OFF' then number_prefix = 'ON'
   book_check = 'ON'
end
when pos('TSO/E Messages', s_line) = 38 then do
tsoe = 'ON'
book_check = 'ON'
end
otherwise nop
end
return
MCSORT EDIT MACRO

/* REXX */
address "ISREDIT"
"MACRO ()"
"RESET"
"STATS OFF"
"LOCATE .ZFIRST"
"SORT 10 A"
"SAVE"
"END"
exit

THE ISPF/PDF APPLICATION

As usual the ISPF application consists of a set of REXX-driven panels and messages.

MCDISP

The MCDISP program drives the application. One paragraph I am rather proud of is the Log_base_2 subroutine. This little morsel of coding returns the logarithm base 2 of a passed number calculated in a rather - IMHO - unorthodox way. The name of the message PDS is hardcoded near the beginning. This is something you probably want to change.

/* REXX to display a message from the file msgdsn.
   With thanks to Chan Tin Pui - MVS Update Issue 153 (June 1999) */

msgdsn = 'JEDSP.MSG'
br_panel = 'MCBRØØ' /* Default browse panel */
call get_cursor_content
call show_msg
exit

/*————————————————————————————————————————————————————————————————————
Show_msg: If a word is found underneath the cursor, the message is
looked-up. If found the contents are browsed, if not the
panel to pick-up the message is displayed.
————————————————————————————————————————————————————————————————————*/

Show_msg:

checked = 'N'
do forever
   if checked = 'N' & messcode <> '' then do
if check_format(messcode 'DUMMY') <> Ø then call get_msg_panel
end

select
when messcode = '' then do
  /*
  There is nothing underneath the cursor. Display the panel
to pick up a message or code.
  */
call get_msg_panel
end

when length(messcode) <= 8 then do
  /*
  The message or code has a length smaller or equal to 8.
  Check the existence as a member name.
  */
  member_exists = SYSDSN('msgdsn("messcode")')
  address "ISPEXEC"
  if member_exists = 'OK' then do
    /*
    The message or code is found as a member.
    Browse it and exit
    */
    "BROWSE DATASET("msgdsn("messcode")") PANEL("br_panel")"
    exit
  end
  else do
    /*
    The message or code is not found as a member.
    Display a message and pick up the value.
    */
    p1 = messcode
    p2 = msgdsn
    "SETMSG MSG(MCDSPØØØ)"
    call get_msg_panel
  end
end

otherwise do
  /*
  There is a word underneath the cursor that is longer than 8
  characters. Check the existence of the index member.
  */
  index_exists = SYSDSN('msgdsn($########$)')
  if index_exists = 'OK' then do
    /*
    The index exists. Allocate it, read it into prefix, and
    free it.
    */
    address "TSO"
    "ALLOC DD(#TEMP#) DSN('msgdsn($########$)') SHR"
    "EXECIO * DISKR #TEMP# (STEM prefix. FINIS)"
    "FREE DD(#TEMP#)"
  end
end
address "ISPEXEC"
/*
Do a binary search to locate the string in the prefix.
stem variable.
*/
if bin_search_index(messcode) = Ø then do
    /*
The message or code is found. Browse it and exit.
*/
    "BROWSE DATASET('"msgdsn"("pointer")') PANEL("br_panel")"
    exit
end
else do
    /*
The message or code is not found. Set a message and
The message or code is not found. Set a message and
*/
p1 = messcode
p2 = msgdsn
"SETMSG MSG(MCDS000)"
call get_msg_panel
end
else do
    /*
There is no index file. Display a message and try to pick
up another message or code.
*/
    address "ISPEXEC"
p1 = msgdsn
"SETMSG MSG(MCDS001)"
call get_msg_panel
end
end

return

Get_msg_panel: Routine to display a panel to pick up a message or code.
If PF3 is pushed the application ends. If the message
or code is filled in here another browse panel is used.
The line where the cursor is, is not copied to the
browse panel.

Get_msg_panel:

address "ISPEXEC"
br_panel = 'MCBR01' /* Browse panel without line */
sac = ''


WSC = ''
DAC = ''
MSG = ''

"ADDPOP"
do forever
   CHECKED = 'N'
   "DISPLAY PANEL(MCREADØØ)"
   IF CAB <> '' & MSG = '' THEN MSG = CAB
   SELECT
      WHEN STRIP(PF3) = 'END' | .
         (SAC = '' & DAC = '' & MSG = '' & WSC = '').
         THEN EXIT
      WHEN MSG <> '' THEN DO
         IF CHECK_FORMAT(MSG 'MSG') = Ø THEN LEAVE
         END
      WHEN SAC <> '' THEN DO
         IF CHECK_FORMAT(SAC 'SAC') = Ø THEN LEAVE
         END
      WHEN WSC <> '' THEN DO
         IF CHECK_FORMAT(WSC 'WSC') = Ø THEN LEAVE
         END
      WHEN DAC <> '' THEN DO
         IF CHECK_FORMAT(DAC 'DAC') = Ø THEN LEAVE
         END
      OTHERWISE NOP
   END
"REMPOP"

RETURN

/*————————————————————————————————————————————————————————————————————
Get_cursor_content: Follow the in-storage pointers to the screen buffer.
Set the variable messcode to the word underneath the
cursor. A word containing only characters in
the 'constant' valid. The screen contents starting
from the word underneath the cursor till EOL are
copied into msgline. This is displayed on top of the
message with the standard browse panel.
————————————————————————————————————————————————————————————————————*/

Get_cursor_content:

Tcb    = PTR(54Ø)
Tcb    = PTR(Tcb + 132)
Fsa    = PTR(Tcb + 112)
R1     = PTR(Fsa + 24)
Tld    = PTR(R1)
Tls    = PTR(Tld + Ø96)
Csr    = PTR(Tld + 164)
ScrW   = PTR(Tld + 192)
Offset = ScrW * TRUNC(Csr/ScrW)
csrp = csr - offl + 1
linead = d2x(tls + offl)
line = storage(linead, scrw)

messcode = ''
valid = '+-$ABCDEFGHIJKLMNOPQRSTUVWXYZØ123456789'

upper line

msgline = line
p = verify(line, valid, csrp)
if p > Ø then line = left(line, p - 1)
right_border = p
p = verify(reverse(line), valid)
if p > Ø then line = right(line, p - 1)
left_border = right_border - p
msgline = right(msgline, scrw - left_border)

messcode = line

return

/* ——————————————————————————————————————————————————————————————————————
Check_format: Check the format of the input fields from the message input panel for all given types (SAC, WSC, DAC and MSG).
If an anomaly is found (eg the character 'Z' in a wait state code a message is set and control is returned to the caller with rc=8. If the format is correct rc=Ø.
If called from show_msg, the type is set explicitly and a recursive call is done.
—————————————————————————————————————————————————————————————————————*/

Check_format:

arg param type

if checked = 'Y' then return(Ø)
checked = 'Y'

param = strip(param)

select
  when type = 'SAC' then do
    select
      when length(param) <> 3 then "SETMSG MSG(MCDSPØØ3)"
      when datatype(param, X) & pos(param, ' ') = Ø then do
        messcode = 'S'param
        /*
          Some abend codes have wildcards.
        */
        if abbrev(messcode, 'SØC') then messcode = 'SØCX'
        else if abbrev(messcode, 'SF') then messcode = 'SFnn'
        return(Ø)
      return
  return
when type = 'WSC' then do
  select
  when length(param) <> 3 then "SETMSG MSG(MCDSPØØ5)"
  when datatype(param, X) & pos(param, ' ') = Ø then do
    messcode = 'W'param
    if abbrev(messcode, 'WFF') then messcode = 'WFFØ'
    return(Ø)
  end
  otherwise "SETMSG MSG(MCDSPØØ6)"
end

when type = 'DAC' then do
  select
  when length(param) > 5 then "SETMSG MSG(MCDSPØØ7)"
  when abbrev(param, '+') | abbrev(param, '-') then do
    hex_part = right(param, 3)
    if datatype(hex_part, X) & pos(hex_part, ' ') = Ø then do
      messcode = '$#' || translate(param, 'PM', '+-')
      return(Ø)
    end
    /* Check message number: right part DAC is hex */
    else "SETMSG MSG(MCDSPØØ8)"
  end
  otherwise "SETMSG MSG(MCDSPØØ9)"
end

when type = 'MSG' then do
  if datatype(left(param, 1), M) | abbrev(param, '$') then do
    messcode = param
    return(Ø)
  end
  /* Message should start with A-Z or $ */
  else "SETMSG MSG(MCDSPØØ2)"
end

otherwise do
  /*
  The type is not specified. This is a call from show_msg. Set
  the type and do a recursive call.
  */
  checked = 'N'
  select
    when datatype(left(param, 1), M) | abbrev(param, '$') then do
      return(check_format(param 'MSG'))
    end
    when abbrev(param, '+') | abbrev(param, '-') then do
      return(check_format(param 'DAC'))
    end
end
otherwise return(check_format(param 'SAC'))
end
end
end

return(8)

/*————————————————————————————————————————————————————————————————————
Ptr: Return a pointer (length 4 if expressed in hexadecimal) from
the passed decimal address.
————————————————————————————————————————————————————————————————————*/

Ptr:

arg value

return x2d(c2x(storage(d2x(value), 4)))

/*————————————————————————————————————————————————————————————————————
Bin_search_index: A binary search routine that goes through the sorted
stem variable prefix. and checks the content of the
passed argument with that of position 10 of the
stem. It relies on the fact that prefix. is sorted
on position 10. The first 8 positions of the stem is
the generated member name. If the name is found it
is set in the variable member and the return code is
zero. Otherwise the return code is 8.
————————————————————————————————————————————————————————————————————*/

Bin_search_index:

arg str

pointer = ''

depth = log_base_2(prefix.Ø)
delta = 2 ** depth / 2
p_rec = delta
do depth
delta = delta / 2
record = strip(word(prefix.p_rec, 2))
select
when record > str then do
  p_rec = p_rec - delta
end
when record < str then do
  p_rec = p_rec + delta
  if p_rec > prefix.Ø then p_rec = prefix.Ø
end
otherwise do
  pointer = word(prefix.p_rec, 1)
  return(Ø)
end
end
return(8)
/*
Log_base_2: Returns the logarithm base 2 of a passed number. The
calculation is done in a - IMHO - quite unorthodox way.
*/
Log_base_2:

arg no

return(length(strip(x2b(d2x(no)), 'L', Ø)))

PANELS MCREAD00, MCBR00, AND MCBR01

The Common User Access (CUA) interface was used to define all the panels. The CUA guidelines define the default colours and emphasise techniques for individual panel elements. See *Interactive System Productivity Facility (ISPF): Dialog Developer’s Guide and Reference OS/390 Version 2 Release 5.0* [SC28-1273] for more details.

The MCREAD00 panel is displayed when the MCDISP REXX EXECs are executed and the word underneath the cursor cannot be found in the message PDS:

```m Pros
)ATTR DEFAULT(](#
£ TYPE(RP)
] TYPE(NT)
( TYPE(NEF) PADC(USER)
) TYPE(PIN)
)BODY EXPAND($$) WINDOW(42,12)
£-$-$-
£-$-$-<JED:SP Message Tool>-$$-
£-$-$-<jan@jedsp.com>-$$-
£
] Message    ===> [MSG ]
] System ABEND code ===> [SAC]
] Wait state code    ===> [WSC]
] CICS ABEND code    ===> [CAB ]
] DB2 SQL code (+/-) ===> [DAC ]
]
)PF3 to exit
]
)INIT
.CURSOR = MSG
)PROC
&PF3 = .RESP
)END
```
In the case that the MCDISP program was started with a valid message number underneath the cursor, the MCBR00 panel displays the member. The line that was under the cursor ID is repeated on the top of the browse panel. This way eventual return and/or reason codes can be easily looked-up.

```plaintext
)ATTR DEFAULT(|#_)
?   TYPE(RP)
+   TYPE(EE)
£   TYPE(PT)
]   TYPE(NT)
{   TYPE(NEF) PADC(USER)
$   AREA(DYNAMIC) EXTEND(ON) SCROLL(ON)
)BODY  EXPAND($$) WIDTH(&ZWIDTH) CMD(ZCMD)
?-$$<JED:SP Message Tool jan@jedsp.com>-$-$- [Command ===>({Z$ $ Scroll
===>({Z £ +MSGLINE
#
$$DATA$ $
$ $
$$ $ $
$ $
)INIT
.ZVARS = '(ZCMD ZSCBR)'
.HELP = MHELP00
&ZCMD = ' '
VGET (ZSCBR) PROFILE
IF (&ZSCBR = ' ') &ZSCBR = 'CSR'
)REINIT
REFRESH(ZCMD,ZSCBR,ZDATA)
)PROC
&ZCURSOR = .CURSOR
&ZCSROFF = .CSRPOS
VPUT (ZSCBR) PROFILE
&ZLVLINE = LVLINE(ZDATA)
)END

The MCBR01 panel displays a member of the message PDS when there was no additional input underneath the cursor. This is, for instance, the case when a message is requested by the MCREAD00 panel.

)ATTR DEFAULT(|#_)
?   TYPE(RP)
+   TYPE(EE)
£   TYPE(PT)
```
CONCLUSION

The application described can replace the need for a commercial product that displays message contents in an interactive session. It saves a lot of time in our shop and I hope it will do in yours too. We are all so used to it now that it has become a natural habit to press PF17 while looking at another weird message. All comments and enhancements are welcomed by e-mail (jan@jedsp.com).

Jan de Decker  
Systems Engineer  
JED:SP NV (Belgium)
Documenting dataset usage in jobs

THE PROBLEM
We have thousands of production jobs at our installation. Some of these are very old and not properly documented. To help alleviate this problem we planned to eliminate tape datasets from production jobs, so that all data will reside on DASD and will be backed up using SnapShot and DFHSM. At the beginning of the operation we needed precise information about the datasets on the tapes.

A SOLUTION
We wrote the following REXX procedure (JCLDOC) to generate dataset documentation from job streams. JCLDOC reads all the members from PDS or PDSE datasets containing jobs to be analysed. The procedure scans JCL statements, extracts key information from them, and puts them in table format. We have found this procedure helpful in the following situations:

• When we have installed a new application we require brief information about the datasets used in batch processing.

• When migrating to SMS or to VTS, it provides precise information about all datasets in our production environment.

The procedure generates two types of report as shown in Figure 1.

SOURCE

/**************************** REXX *****************************/
/* Procedure forms dataset documentation from JCL. */
/* Output is a list of datasets with information about the jobs and */
/* programs that process them. */
/* Input:  library with JCL */
/* Output: reports of different types, depending of parameter value */
/* %JCLDOC Parm */
/* Parm can be one of the following: */
/* - Dsname Volume  - name of library with JCL to process, and */
/*   optionally Volume on which dataset resides, if it is not in */
/*   catalog. */
/* - REPORT1 - generate report grouped by jobs. */
/* - REPORT2 - generate report grouped by datasets. */
/*****************************/
Report 1 – information about datasets is grouped by jobs. We can see all the datasets that are used in each analysed job. From the following part of REPORT1 we can see how it looks:

<table>
<thead>
<tr>
<th>JOBNAME</th>
<th>STEPNAME</th>
<th>PGMNAME</th>
<th>DDNAME</th>
<th>DISP</th>
<th>DNAME / SYSOUT CLASS</th>
<th>UNIT</th>
<th>LABEL</th>
<th>JCL DSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>SYSTSRT</td>
<td>SYOUT=*</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>BASE</td>
<td>SHR</td>
<td>appl1d.BASE.PERM</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>EXTRACT</td>
<td>(NEW,CATLG)</td>
<td>appl1d.EXTRACT.TEMP</td>
<td>TAPE</td>
<td>1,SL</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>SYSPRINT</td>
<td>SYOUT=*</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>REPORT</td>
<td>SYOUT=*</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>SYSDUMP</td>
<td>SYOUT=*</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>app1DN01</td>
<td>STEP0003</td>
<td>APPLB265</td>
<td>SYSTSIN</td>
<td>*</td>
<td>POSTP.APPL.CNTL(APPLDN01)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Report 2 – information is grouped by dataset name. We can see, according to disposition, which program creates a dataset and which one uses it. In the last column of the report above we can see members in which we have to make synchronous change (in our case we change the unit to point to DASD instead of a tape).
/* Trace ?R */
ARG Arg1 Arg2
userid=SYSVAR(SYSUID)
prefix=SYSVAR(SYSPREF)
"PROFILE NOPREFIX"

rrc=Ø
Select
When Arg1 = 'REPORT1'
Then call Report 126 44
When Arg1 = 'REPORT2'
Then call Report 59 44
Otherwise
   rrc = Maka_Jcl_Doc( Arg1, Arg2 )
End /* select */
If prefix <> ''
Then "PROFILE PREFIX("prefix")"
Return rrc

/*-----------------------------------------------------------------*/
/* Get dataset */
 /*-----------------------------------------------------------------*/
Maka_Jcl_Doc: Procedure
Arg DsName, Volume
rrc=Ø
If SYSDSN(DsName) <> 'OK'
Then Do
   Say 'Missing dataset name'
   rrc=12
End
Else Do
   findm=Ø
   Records.Ø=Ø
   t=OUTTRAP('dsnc.',,NOCONCAT)
   "LISTDS "DsName
   t=OUTTRAP('OFF')
   PARSE UPPER VAR dsnc.3 recfm lrecl blksizl dsorg
   If dsorg = 'PS' OR (dsorg = 'PO' AND Index(DsName,',')) > Ø)
   Then Do
      rcu=get_dataset(DsName, Volume)
      rrc=MAX(rrc,rcu)
   End
Else
   If dsorg = 'PO'
   Then Do:
      t=OUTTRAP('dsnc.',,NOCONCAT)
      "LISTDS "DsName" members "
      t=OUTTRAP('OFF')
      Do i=1 To dsnc.Ø
         If INDEX(dsnc.i,'MEMBERS') > Ø
            Then Leave
      End
   "EXECIO Ø DISKW jcldoc (OPEN)"
   Do i=i+1 To dsnc.Ø
      Parse Var Dsnc.i Member Rest
Ds_Name=DsName||(Member')
rcu=get_dataset(Ds_Name, Volume)
rrc=MAX(rrc,rcu)
End
"EXECIO Ø DISKW jcdloc (FINIS)"
End
Else Do
Say 'This Dsorg dsorg ' is not supported |||
rrc=16
End
End
Return rrc
/*-----------------------------------------------------------------*/
/* Get dataset                                                     */
/*-----------------------------------------------------------------*/
get_dataset:Procedure Expose Records.
Arg Ds_Name, Volume
call alloc_Ds 'INOUT' Ds_Name Volume
rrc=Ø
"EXECIO Ø DISKR inout (OPEN)"
If RC <> Ø
Then Do
Say '>>> Dataset' DS_name ' CANNOT BE OPENED |||'
EXIT 4
End
"EXECIO * DISKR inout (STEM Records.)"
"EXECIO Ø DISKR inout (FINIS)"
indjcl=Ø
call Analyze_records
If indjcl=Ø
Then Say '***' Left(DS_name,44) 'No Jcl'
Else Say '***' Left(DS_name,44) 'Jcl'
Return rrc
 /*-----------------------------------------------------------------*/
/* Analyse Record                                                  */
 /*-----------------------------------------------------------------*/
Analyze_records:Procedure Expose Records. jobname stepname pgmname,
 ddname dsname disp unit label ds_name indjcl
 jobname  = '
 stepname = '
 pgmname  = '
 old_ddname = '
 ddname   = '
 disp     = '
 dsname   = '
 unit     = '
 label    = '
 jclstmt  = '
 indprt   = '
Do i=1 to Records.Ø
   If substr(records.i,1,2) = '//' & substr(records.i,3,1) <> '*'
     Then Do
       PARSE VAR records.i jcl1 jcl2 jcl3 comment
If index(jcl2,'=') = Ø
Then jclstmt = jcl2
Select
When jclstmt = 'JOB'
Then Do
jclstmt = ''
jobname = substr(jcl1,3)
End
When jclstmt = 'EXEC'
Then Do
jclstmt = ''
stepname = substr(jcl1,3)
jclparm=jcl2
PARSE VAR jclparm parml '=' parmrest
Select
When parml = 'PGM'
Then pgmname = get_subparm(parmrest)
When parml = 'PROC'
Then pgmname = get_subparm(parmrest)
Otherwise
    pgmname = get_subparm(parml)
End
End
When jclstmt = 'DD'
Then Do
ddname = substr(jcl1,3)
If old_ddname ¬= ddname & ddname ¬= ''
Then old_ddname = ddname
If index(jcl2,'=') = Ø
Then jclparm = jcl2
Else jclparm = jcl2
If right(jclparm,1) = ','
Then indprt = ''
Else indprt = 'Y'
Do While(jclparm ¬= '')
    PARSE VAR jclparm parml '=' parmrest
    Select
    When parml = 'DSN'
    Then dsname = get_subparm(parmrest)
    When parml = 'SYSOUT'
    Then dsname = 'SYSOUT='||get_subparm(parmrest)
    When parml = '*'
    Then dsname = '*'||get_subparm(parmrest)
    When parml = 'UNIT'
    Then unit = get_subparm(parmrest)
    When parml = 'LABEL'
    Then label = get_subparm(parmrest)
    Otherwise
        rest = get_subparm(parmrest)
    End
jclparm=parmrest
End
If indprt = 'Y'
Then call print_doc
End
Otherwise
  If jcl2 = ''
    Then ddname=''
  Else If index(jcl2,'=') >Ø
    Then jclparm=jcl2
    Else jclparm=jcl3
End
End
Return
/*-----------------------------------------------------------------*/
/* Get subparameter                                              */
/*-----------------------------------------------------------------*/
get_subparm: Procedure Expose parmrest
Arg parm
If substr(parm,1,1) = '('
Then Do
  NoB=Ø
  LenParm = length(parm)
  Do j=1 to LenParm Until(NoB=Ø)
    If substr(parm,j,1) = '('
      Then NoB=NoB+1
    Else
      If substr(parm,j,1) = ')' 
        Then NoB=NoB-1
    End
    subparm = substr(parm,1,j)
    j=j+1
    If substr(parm,j,1) = ','
      Then j=j+1
    parmrest = substr(parm,j)
  End
Else PARSE VAR parm subparm ',' parmrest
return subparm
/*-----------------------------------------------------------------*/
/* print Doc                                                       */
/*-----------------------------------------------------------------*/
Print_Doc: Procedure Expose jobname stepname pgmname, 
          old_ddname disp dsname unit label ds_name indjcl
row.1 = Left(jobname,8) Left(stepname,8),
       Left(pgmname,8) Left(old_ddname,8),
       Left(disp,21) Left(dsname,44),
       Left(unit,21) Left(label,21) ds_name 
"EXECIO 1 DISKW jcldoc (STEM row.)"
disp     = ''
dsname   = ''
unit     = ''
label = ''
If indjcl = Ø
  Then indjcl=1
return
/*-----------------------------------------------------------------*/
/* Alloc Dataset */
/*-----------------------------------------------------------------*/
Alloc_DS: Procedure  
Arg DD_Name Ds_Name Volume 
msgstat=MSG("OFF")    /* Inhibit the display of TSO/E informational */
"FREE F("DD_Name")"
t=MSG(msgstat)        /* Returns the previos status of message */.
If Volume = ''
Then "ALLOC F("DD_Name") DA("*****Ds_Name*****") SHR "
Else "ALLOC F("DD_Name") DA("*****Ds_Name*****") SHR 
   " VOLUME("Volume") UNIT(SYSDA)"
Return
/*-----------------------------------------------------------------*/
/* print header */
/*-----------------------------------------------------------------*/
Print_Header:Procedure  
row.1 = copies('-',174)
row.2 = CENTER('JOBNAME',8)||'|'||,
        CENTER('STEPNAME',8)||'|'||,
        CENTER('PGMNAME',8)||'|'||,
        CENTER('DDNAME',8)||'|'||,
        CENTER('DISP',21)||'|'||,
        CENTER('DSNAME / SYSOUT CLASS',44)||'|'||,
        CENTER('UNIT',21)||'|'||,
        CENTER('LABEL',21)||'|'||,
        CENTER('JCL DSN',21)
"EXECIO 2 DISKW REPORT (STEM row.)"
Return
/*-----------------------------------------------------------------*/
/* report 1 */
/*-----------------------------------------------------------------*/
Report: Procedure  
Arg Pos Len 
"EXECIO Ø DISKR jcldoc (OPEN)"
"EXECIO Ø DISKW REPORT (OPEN)"
call Print_header 
"EXECIO 1 DISKR jcldoc (STEM Record.)"
old_name='-----'
Do While(RC < 2)
  name=substr(record.1,Pos,Len)
  If old_name ¬= name
  Then Do
    row.1 = copies('-',174)
    "EXECIO 1 DISKW REPORT (STEM row.)"
    old_name=name
    End
  "EXECIO 1 DISKW REPORT (STEM Record.)"
JOB FOR SUBMITTING JCLDOC

//useridF JOB CLASS=A,MSGCLASS=X,MSGLEVEL=(Ø,Ø),NOTIFY=&SYSUID
//IDCAMS EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=X
//SYSIN DD *
DELETE userid.#JCLDOC.LIST
DELETE userid.#JCLDOC2.LIST
SET MAXCC=Ø
/*
//JCLDOC EXEC PGM=IKJEFTØ1,DYNAMNBR=5Ø,REGION=4M
//SYSPROC DD DSN=userid.USER.CLIST,DISP=SHR
//SYSTSPRT DD SYSOUT=*/
//REPORT DD SYSOUT=*/
//JCLDOC DD DSN=userid.#JCLDOC.LIST,DISP=(NEW,CATLG,DELETE),
//UNIT=SYSDA,DCB=(RECFM=FB,LRECL=175,BLKSIZE=Ø),
//SPACE=(TRK,(10,5),RLSE)
//SYSTSIN DD *
%JCLDOC userid.RADNA.CNTL
%JCLDOC REPORT1
/

Emina Spasic and Dragan Nicholic
Systems Programmers (Yugoslavia) © Xephon 2001

LegacyJ has begun shipping version 2.6 of its PERCobol Enterprise Cobol compiler, with a range of new capabilities covering data access, enterprise COBOL, and application tools.

Data access and data types are covered via C-ISAM and D-ISAM file access to X/Open ISAM compliant shared libraries. There are also AcuCOBOL native file sources including sequential, relative and indexed, AcuCOBOL full data type support, and complete elementary Micro Focus data-type support.

Enterprise COBOL features include object-oriented COBOL support to create Java classes and methods in COBOL, plus direct support of EJB using COBOL classes, and sessions capability for independent program copies in a single process. Also new is a LegacyJ deployment tool for building distribution packages with application, runtime and resource files needed to deploy an application to local or remote servers.

The product contains COBOL compile and execution functionality, accessing traditional COBOL file types and databases across a variety of operating platforms. It generates platform-independent graphical user interfaces and allows these GUIs to morph consistently with other interfaces present on the host platform.

For further information contact:
LegacyJ, 5035 Almaden Expressway
San Jose, California 95119, USA.
Tel: 408 979 8090
Fax: 408 979 8099
http://www.legacyj.com

IBM has announced Version 2.2 of its Tivoli Manager for Domino for OS/390, supporting clustering, partitioned servers, full monitoring functions, an ability to scan text-based logs, and management of the mailbox database.

Version 2.2 provides OS/390 Domino administrators with improved monitoring and administration capabilities. Installation and implementation is via CDs and does not impact security of OS/390 systems, it is claimed.

It supports Domino Release 5.03 and higher, including transaction logging, partitioned servers, and clustered servers. Using the Version 2.2 tools, administrators gain visibility to Access Control List changes, and the ability to perform a quick scan of Domino system logs.

The new release also supports protocols including NNTP, POP3, and the SMTP MTA. Besides monitoring these protocols, it can monitor the integrity of Domino Enterprise Connections Services if these are in place. Reporting mechanisms provide in-depth reports and trend analysis capabilities.

For further information contact:
Tivoli Systems, 9442 Capital of Texas Highway, North Austin, TX 78759, USA.
Tel: 512 436 8000
Fax: 512 794 0623

Tivoli Systems, Sefton Park, Bells Hills, Buckinghamshire, SL2 4HD, UK.
Tel: 01753 896 896
Fax: 01753 896 899
http://www.tivoli.com

* * *