May 2003

In this issue

3  Structured design and program messages
7  Keeping track of deleted members
10 Calculating the index CI size
16 Parsing strings in Assembler programs – part 2
39 Individual descriptions for PDS members on a member selection panel
50 Descending key support in IMS made easy
66 Finding CSECTs within LPA load modules in virtual storage
74 MVS news

© Xphon plc 2003
**MVS Update**

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

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

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

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

**Editor**
Trevor Eddolls

**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.

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

© Xephon plc 2003. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher. Copying permits are available from Xephon in the form of pressure-sensitive labels, for application to individual copies. A pack of 240 labels costs $36 (£24), giving a cost per copy of 15 cents (10 pence). To order, contact Xephon at any of the addresses above.

*Printed in England.*
Structured design and program messages

In a previous article (Structured design approach to program messages, MVS Update issue 199, April 2003) we discussed the use of a structured methodology for the development and maintenance of program messages. Now that we have a defined structure in place to house the messages, we turn our attention to how we might access the messages programmatically. As with any programming exercise, there are multiple ways to satisfy our requirements. For us, the primary requirements continue to be the use of a standard structure for the messages, such as the ME$$AGE$ CSECT structure we previously described, and a simple and standardized way to reference the messages.

We will now turn our attention to how we can provide a simple and standard access to the message structure. To accomplish this task we developed the $EDTML macro for message look-up. Using the macro is simple and straightforward. To invoke the macro, we need three parameters. The first parameter is the message number, which corresponds to the message number or ID in the ME$$AGE$ table structure. It will allow us to perform a simple look-up in the table. The second parameter that is needed is a location to place the address of the returned message structure. This can be either a register or a fullword storage location. The last parameter that is needed is the address of the message structure CSECT. In our previous article, we had decided on the naming convention of ME$$AGE$ for the messages CSECT. Obviously you can name it whatever you like. Our suggestion would be to pick a name and then use that name consistently. Let’s turn our attention to a very simple example of a message CSECT and then see how we use $EDTML to perform the look-up.

MESSAGES CSECT CSECT NAME
MESSAGES AMODE 31 SPECIFY AN ADDRESSING MODE
MESSAGES RMODE ANY SPECIFY THE RESIDENCY
SPACE 1
DC AL4(A_NEXT-A_FIRST) SIZE OF AN ENTRY
If you create the message source member as above and assemble it, you will see that a simple table look-up structure is created. The fullword at location X'00' in the CSECT contains the size of each table entry, while the fullword at location X'04' contains the number of message entries that are in the table itself. The contents of these two fullword locations provide us with the necessary information to traverse the following 5-byte entries. Each of the 5-byte entries contains the message number in byte X'00' and the displacement to the message at byte X'01'.

Now let's turn our attention to the $EDTML macro. We have included the macro source below for your examination. There are a variety of methods that could be used to traverse the table to locate the message. Since we have chosen to limit the table size to 255 entries, we have opted to use a sequential scan of the table. There are other algorithms that are more efficient, but based on the table size constraint we did not feel it necessary to employ any of them. If you should choose to modify the table structure to handle more than 255 messages, our
recommendation would be to consider one of the alternative
look-up techniques. As designed, the $EDTML macro makes
use of registers 14, 15, 0, and 1, so we save and restore these
registers as part of the code. Also note that, if we are not able to
locate the requested message in the table, then high values are
returned.

MACRO

**********************************************************************
*       THIS MACRO IS DESIGNED TO BE USED WITH A STANDARD MESSAGES   *
*       CSECT. YOU PROVIDE THE MESSAGE NUMBER THAT YOU WANT TO LO- *
*       CATE, AND THE MACRO WILL RETURN THE ADDRESS OF THE MESSAGE   *
*       IF IT IS IN THE TABLE.                                       *
*                                                                    *
*       INPUT: CONSISTS OF THREE PARAMETERS, THE MESSAGE NUMBER, THE *
*       REGISTER OR FULLWORD TO PLACE THE MESSAGE ADDRESS            *
*       INTO, AND THE ADDRESS OF THE MESSAGE TABLE CSECT. IF         *
*       THE MESSAGE IS NOT FOUND IN THE TABLE, HIGH VALUES           *
*       ARE RETURNED.                                                *
*                                                                    *
*       EXAMPLE: $EDTML   MSG#, (REGISTER), MSG. TABLE CSECT ADDR     *
*       EXAMPLE: $EDTML   MSG#, FIELD, MSG. TABLE CSECT ADDR           *
**********************************************************************

$EDTML

LCLC  &LBL1
LCLC  &LBL2
LCLC  &LBL3
LCLC  &MSGNO
LCLC  &RVAL
LCLC  &MSGTBL

**********************************************************************
* SEE HOW MANY PARAMETERS WERE PROVIDED. WE MUST HAVE THREE PARMS  *
* TO PERFORM THE MESSAGE LOOKUP                                      *
**********************************************************************

AIF   (N'&SYSLIST EQ 3).MT4
MNOTE 12,'$EDTML ERROR - YOU MUST PROVIDE THREE PARAMETERS'
AGO   .MEND

.MT4     ANOP

**********************************************************************
* GO AHEAD AND CREATE THE LABELS WE WILL NEED.                      *
**********************************************************************

&LBL1   SETC  'LB1'. '&SYSNDX'
&LBL2   SETC  'LB2'. '&SYSNDX'
&LBL3   SETC  'LB3'. '&SYSNDX'

**********************************************************************
* PICK UP THE PARMS AND ASSIGN THEM TO LOCAL VARIABLES.             *
**********************************************************************

&MSGNO   SETC  '&SYSLIST(1)'
One concern that you may have is that, as we have set up the structure, it will accommodate only 255 messages. Although this may seem very restrictive, we believe that it helps facilitate a more modular program design. You could consider segregating the messages by type – a table for informational, a table for warnings, and a table for errors. Or you may opt to differentiate them by groups of sequence numbers. The key is not how you break them down, but the fact that each table will look the same from a programmatic perspective.

So where does this discussion about messages, message tables, and table look-up techniques lead us? Our intent with this short discussion was to provide a very simple example of how we
might begin the process of utilizing standard repeatable techniques for program development. Our goal with this standardization process is to decrease the programming effort, decrease the maintenance effort, and increase our ability to more rapidly develop solution-oriented tools and programs. Our hope is that you can use these techniques to your benefit as well.

Keeping track of deleted members

PROBLEM ADDRESSED
Every now and then it happens that someone from our site’s large user base asks the perennial question:

“Where is my member? It used to be there in that library, but it is not there any more. Who deleted it?”

During the course of a recent migration to a new release of the operating system, this question occurred more frequently than usual, and prompted me to search for a quick, simple, and easy-to-use solution that would supply a straightforward answer.

SOLUTION PROPOSED
In a search for a solution I asked myself whether there is any SMF record that will allow me to determine who deleted a member in a PDS/E library. It came as a surprise to find that actually such an SMF record was introduced with z/OS V1R3.

When enabled by SMFPRMxx TYPE parameter, SMF creates a type 42, subtype 21 record, which is written each time a member is deleted from a PDS or a PDSE to indicate who or what (job, started task, or TSO user) deleted the member. It contains the name of the dataset and the volume serial of the volume on which
it resided, as well as all the aliases of the member that will fit in the SMF record.

A detailed description of the layout of an SMF type 42 record and its subtypes can be obtained from the *MVS System Management Facilities (SMF)*, SA22-7630-03 manual. You can also find the subtype descriptions in macro IGWSMF in SYS1.MACLIB.

Based on the record description obtained from this manual, a simple report writer was written.

Before proceeding any further it might be helpful to see what types of SMF records are contained in a dataset one is about to process and what system or time they represent. When one tries to browse such a dataset, ISPF rejects this request and complains about the record format. The excellent utility ERBSCAN (from IBM) was used to browse a sequential SMF dataset, since it returns a list of records showing the type and subtype and information about when the record has been written and on what system.

This report writer uses ICETOOL because of its ability to access and process the wealth of information written by SMF. To process the SMF records with ICETOOL poses a few potential problems, among which is that some processing might be disrupted. Sorting SMF data may issue an error message (ICE204A 5), set a return code of 16, and terminate if it detects an incomplete spanned record. In order to overcome this potential obstacle, DFSORT’s SPANINC=RC4 option was used to remove the incomplete spanned records.

It should be noted that SPANINC=RC0 tells DFSORT (Release 14) to issue a warning message, set a return code of 0, and eliminate all incomplete spanned records it detects. Valid records (that is, complete spanned records) are recovered and written to the output dataset, while SPANINC=RC4 does the same thing as SPANINC=RC0, but with a return code of 4 instead of 0. The shipped default is SPANINC=RC16.
CODE

The code is a two part stream. In the first part (COPYSMF) selected SMF records (selection being defined by INCLUDEs) are copied from the SMF dataset to a file, which can be used as a base of archived records.

In the second part (RPT42), the captured records are formatted and a report produced. The report shows the job and user performing the deletion, along with the member deleted, its dataset name, and volume serial number of the dataset, as well as the date and time deletion took place.

```
// JOBCARD  JOB ...  
// COPYSMF  EXEC PGM=ICETOOL  
// TOOLMSG  DD SYSOUT=*  
// DFSMSG   DD SYSOUT=*  
// RAWSMF   DD DSN=your.smf.dataset,DISP=SHR  
// SMF42    DD DSN=userid.T4221.TEST,  
// SPACE=(CYL,(xx,yy)),UNIT=SYSDA,  
// DISP=(NEW,CATLG,DELETE),  
// DCC=(RECFM=VB,LRECL=32756,BLKSIZE=32760)  
// TOOLIN  DD *  
COPY FROM(RAWSMF) TO(SMF42) USING(SMFI)  
// SMFI CNTL DD *  
OPTI ON SPANINC=RC4,VLSHRT  
INCLUDE COND=(6,1,BI,EQ,42,AND,23,2,BI,EQ,21)  
/*

// RPT42  EXEC PGM=ICETOOL  
// TOOLMSG  DD SYSOUT=*  
// DFSMSG   DD SYSOUT=*  
// SMF42    DD DSN=userid.T4221.TEST,DISP=SHR  
// SMFREP  DD SYSOUT=*  
// TOOLIN  DD *  
DISPLAY FROM(SMF42) LIST(SMFREP)  
TITLE('Deleted PDS / PDSE members') DATE(4MD/) TIME -  
HEADER('Sys') ON(15,4,CH) -  
HEADER('Date') ON(11,4,DT1,E'9999/99/99') -  
HEADER('Time') ON(7,4,TM1,E'99:99:99') -  
HEADER('Job Name') ON(89,12,CH) -  
HEADER('Step Name') ON(101,8,CH) -  
HEADER('Dataset') ON(117,35,CH) -  
HEADER('Member') ON(169,12,CH) -  
HEADER('Vol Ser') ON(157,11,CH) BLANK  
/*
```

Mile D Pekic
Systems Programmer (Yugoslavia)  

© Xephon 2003
Calculating the index CI size

At our site we have lots of VSAM datasets. Our department is responsible for, amongst other things, how they behave.

A VSAM dataset’s Control Interval size depends on the keylength and the CA size of the dataset. In CICS, we have buffers for the data and the indexes. These buffers match the CI sizes of the VSAM datasets. For instance, we have 1KB and 2KB buffers, and a CI size of 720 bytes is rounded up to 1KB and a CI size of 1342 bytes is expanded to 2KB. We have found that the throughput is maximal if we use these values.

However, if 2KB is the maximum buffer in CICS, an advised CI size of 3,000 will become 2KB; otherwise CICS wouldn’t be able to process the dataset. In this case, we don’t achieve maximum throughput, but we must make a compromise – it’s either reduced throughput or no throughput at all.

The program presented here allows you to obtain a list of the advised CI sizes.

THE PL/I PROGRAM – IXCISIZE

/****** VSAM INFO TO CALCULATE INDEX CI SIZE =IXCISIZE=******/
/* PROGRAM : IXCISIZE */
/* PURPOSE : GET A LIST OF ADVISED CI SIZES */
/* INPUT : IDCAMS : LIST OUTPUT OF IDCAMS */
/* OUTPUT : PRINT01 : LIST YOU REQUESTED */
/* PARAMETER : F = ONLY DIFFERENCES OR */
/* SOMETHING ELSE = COMPLETE LIST */
/* FOLLOWED BY */
/* NUMBER = LIST CI SIZE GREATER THAN */
/****** VSAM INFO TO CALCULATE INDEX CI SIZE =IXCISIZE=******/
IXCISIZE: PROC (PARM) OPTIONS(MAIN) REORDER;
DCL PARM CHAR(009) VAR;
DCL 1 P BASED(ADDR(PARM)),
  2 PARM_LL CHAR(02),
  2 PARM_F CHAR(01),
  2 PARM_CI_SIZE PIC'9999';
DCL IDCAMS FILE RECORD SEQL INPUT ;
DCL PRINT01 FILE RECORD SEQL OUTPUT;
ON ENDFILE (IDCAMS)  EOF  = '1' B;
/* RECORD GIVEN BY IDCAMS */
DCL REC       CHAR(131) INIT (' ');
/* GET DATASETNAME */
DCL 1 LCØ     BASED(ADDR(REC)),
   2 NVT1     CHAR(01),
   2 FIND_CLUSTER CHAR(07),
   2 NVT2     CHAR(09),
   2 NAME     CHAR(44);
/* FIND INDEX TYPE */
DCL 1 LC1     BASED(ADDR(REC)),
   2 NVT1     CHAR(08),
   2 FIND_INDEX CHAR(05);
/* FIND KEYLENGTH AND CI / CA SIZES */
DCL 1 LC2     BASED(ADDR(REC)),
   2 NVT1     CHAR(08),
   2 FIND_KEYLENGTH CHAR(06),
   2 NVT2     CHAR(15),
   2 KEYLENGTH CHAR(03),
   2 NVT3     CHAR(63),
   2 FIND_CI_CA CHAR(05),
   2 NVT4     CHAR(15),
   2 CI_CA    CHAR(04);
/* GET CISIZE FROM IDCAMS */
DCL 1 LC3     BASED(ADDR(REC)),
   2 NVT1     CHAR(95),
   2 FIND_CISIZE CHAR(06),
   2 NVT4     CHAR(14),
   2 CISIZE   CHAR(04);
/* OUTPUT HEADER LINES */
DCL 1 K1,
   2 ASA      CHAR(01) INIT('1'),
   2 TEXT1    CHAR(10) INIT('* * * * '),
   2 TEXT2    CHAR(27) INIT(' VSAM ADVICE CISIZE OF THE '),
   2 TEXT3    CHAR(26) INIT(' INDEX COMPONENT * * * '),
   2 TEXT4    CHAR(40) INIT(' '),
   2 TEXT5    CHAR(07) INIT(' DATE: '),
   2 DATE1    CHAR(08) INIT(' '),
   2 TEXT6    CHAR(04) INIT(' '),
   2 TEXT7    CHAR(05) INIT(' PAGE '),
   2 PAGE     PIC'ZZ9' INIT(0);
DCL 1 K2,
   2 ASA      CHAR(01) INIT(' '),
   2 TEXT1    CHAR(44) INIT(' DATASETNAME '),
   2 NVT1     CHAR(01) INIT(' '),
   2 TEXT2    CHAR(09) INIT(' KEYLENGTH '),
   2 NVT2     CHAR(03) INIT(' '),
   2 TEXT3    CHAR(05) INIT(' CI / CA '),
   2 NVT3     CHAR(03) INIT(' '),
   2 TEXT4    CHAR(10) INIT(' CISIZE NOW '),
/* DETAIL LINES */
DCL 1 D1,
2 ASA CHAR(Ø1) INIT(' '),
2 NAME CHAR(44) INIT(' '),
2 NVT1 CHAR(Ø3) INIT(' '),
2 KEYLL CHAR(Ø3),
2 NVT2 CHAR(Ø7) INIT(' '),
2 CI_CA CHAR(Ø4),
2 NVT3 CHAR(Ø7) INIT(' '),
2 IND_CI CHAR(Ø4),
2 NVT4 CHAR(11) INIT(' '),
2 ADV_CI PIC'ZZZZ99',
2 NVT5 CHAR(16) INIT(' '),
2 AFG_CI PIC'ZZ99';
DCL D1_AFG_CI CHAR(Ø4) BASED(ADDR(D1.AFG_CI));
DCL CATNAME CHAR(Ø8) BASED(ADDR(D1.NAME));
/* BUILTINS */
DCL (ADDR,DATE,SUBSTR) BUILTIN;
/* HELP FIELDS */
DCL EOF BIT (Ø1) INIT ('Ø'B);
DCL (I,J) FIXED BIN (15);
DCL RECORD_COUNT FIXED BIN (15) INIT(99);
DCL HELP_KEYLL_C CHAR(Ø3),
HELP_KEYLL_P PIC'999' BASED(ADDR(HELP_KEYLL_C));
DCL HELP_CI_CA_C CHAR(Ø4),
HELP_CI_CA_P PIC'9999' BASED(ADDR(HELP_CI_CA_C));
/* CONTROL INTERVAL SIZES TABLE */
DCL CI_TABEL(Ø:5) FIXED BIN (15)
INIT(4Ø96, 4Ø96, 2Ø48, 1Ø24, 512, Ø);
/* BEGIN MAIN PROGRAM */
K1.DATE1 = DATE;
K1.DATE1 = SUBSTR(K1.DATE1,5,2) || '-' ||
SUBSTR(K1.DATE1,3,2) || '-' ||
SUBSTR(K1.DATE1,1,2);
OPEN FILE (IDCAMS),
FILE (PRINTØ1);
READ FILE (IDCAMS) INTO (REC);
DO WHILE (¬EOF);
    DO WHILE (LC1.FIND_INDEX ¬= 'INDEX' & ¬EOF);
        IF LC0.FIND_CLUSTER = 'CLUSTER' THEN D1.NAME = LC0.NAME;
        REC = ' '; READ FILE (IDCAMS) INTO (REC); END;
    IF ¬EOF
THEN DO;
  IF CATNAME = 'CATALOG.'
  THEN RECORD_COUNT = 99;
  DO WHILE (FIND_KEYLENGTH ≠ 'KEYLEN' &
       ¬EOF);
    REC = ' ';
    READ FILE (IDCAMS) INTO (REC);
  END;
  IF ¬EOF
  THEN DO;
    /* GET THE DETAIL OUTPUT FROM IDCAMS
       AND FILL THE DETAIL LINES                         */
    DO I = 1 TO 3 WHILE (SUBSTR(LC2.KEYLENGTH,I,1) = '·');
       SUBSTR(LC2.KEYLENGTH,I,1) = ' ';
    END;
    D1.KEYLL = LC2.KEYLENGTH;
    DO WHILE (FIND_CICA ≠ 'CI/CA' &
       ¬EOF);
      REC = ' ';
      READ FILE (IDCAMS) INTO (REC);
    END;
    DO I = 1 TO 4 WHILE (SUBSTR(LC2.CICA,I,1) = '·');
       SUBSTR(LC2.CICA,I,1) = ' ';
    END;
    D1.CI_CA = LC2.CICA;
    DO WHILE (FIND_CISIZE ≠ 'CISIZE' &
       ¬EOF);
      REC = ' ';
      READ FILE (IDCAMS) INTO (REC);
    END;
    DO I = 1 TO 4 WHILE (SUBSTR(LC3.CISIZE,I,1) = '·');
       SUBSTR(LC3.CISIZE,I,1) = ' ';
    END;
    D1.IND_CI = LC3.CISIZE;
    HELP_KEYLL_C = D1.KEYLL;
    DO I = 1 TO 3 WHILE (SUBSTR(HELP_KEYLL_C,I,1) = '');
       SUBSTR(HELP_KEYLL_C,I,1) = 'Ø';
    END;
    HELP_CICA_C = D1.CI_CA;
    DO I = 1 TO 4 WHILE (SUBSTR(HELP_CICA_C,I,1) = '');
       SUBSTR(HELP_CICA_C,I,1) = 'Ø';
    END;
    I = HELP_KEYLL_P / 2;
    I = I * HELP_CICA_P;
    D1.ADV_CI = I;
    J = 1;
    DO WHILE (CI_TABEL(J) >= I);
       J = J + 1;
    END;
    D1.AFG_CI = CI_TABEL(J - 1);
IF D1.AFG_CI > PARM_CI_SIZE
THEN DO;
    /* PRINT ROUTINE                           */
    IF RECORD_COUNT > 60
THEN DO;
        RECORD_COUNT = 3;
        K1.PAGE = K1.PAGE + 1;
        WRITE FILE(PRINT01) FROM(K1);
        WRITE FILE(PRINT01) FROM(K2);
        D1.ASA = '0';
        END;
    IF PARM_F != 'F'
THEN DO;
        WRITE FILE (PRINT01) FROM (D1);
        RECORD_COUNT = RECORD_COUNT + 1;
        END;
    IF D1.IND_CI != D1_AFG_CI
THEN DO;
        IF PARM_F != 'F'
        THEN D1.ASA = '+';
        ELSE RECORD_COUNT = RECORD_COUNT + 1;
        WRITE FILE (PRINT01) FROM (D1);
        END;
    D1.ASA = ' ';
    END;
    END;
END; 
CLOSE FILE (IDCAM5),
    FILE (PRINT01);
END IXCISIZE;

RUNNING THE PROGRAM

To run the compiled program, you need the following JCL:

// STREAM1   JOB CENTRUM1,'VSAM INFO',
//             CLASS=A,MSGCLASS=A
// * PROJECT    : DISK UTILITIES
// * JOB        : LIST DATA TO KNOW CONTROL INTERVAL SIZE OF WHICH
// *              VSAM DATASET HAS TO BE CHANGED
// * STEPS      : NR | PROGRAM | FUNCTION
// *              --|--------|-------
// *              10 | IDCAMS | LIST VSAM DATASETS
// *              20 | IXCISIZE | PRINT DIFFERENCES OF CISIZE NOW AND
// * SYSOUTSRT. : 1 : ON PAPER
// * LIST ALL CHARACTERISTICS OF THE VSAM DATASETS INCLUDING HISTORY
OUTPUT SAMPLES

Various output samples are shown below.

Sample 1 with PARM=’/F1024’ (as above):

```
1 * * * *   VSAM ADVISE CISIZE OF THE INDEX COMPONENT * * * *
DATE: 07-07-02  PAGE 1
   DATASETNAME    KEYLENGTH  CI/CA  CISIZE NOW  ADVISED  ROUNDED CISIZE
 VSAM.ALG.DATASET1  15    180      1536       1260       2048
 VSAM.ALG.DATASET2  18    168      1536       1512       2048
 VSAM.IPCS.DATASET1 128    45       512        2880       4096
```

Sample 2 with PARM=’/F’:

```
1 * * * *   VSAM ADVISE CISIZE OF THE INDEX COMPONENT * * * *
DATE: 07-07-02  PAGE 1
   DATASETNAME    KEYLENGTH  CI/CA  CISIZE NOW  ADVISED  ROUNDED CISIZE
 VSAM.ALG.DATASET5  8    180      1536        720       1024
 VSAM.ALG.DATASET9  8    180      1536        720       1024
 VSAM.ALG.DATASET1 128    12       1024         12       512
```

Sample 3 WITHOUT PARM:

```
1 * * * *   VSAM ADVISE CISIZE OF THE INDEX COMPONENT * * * *
DATE: 07-07-02  PAGE 1
   DATASETNAME    KEYLENGTH  CI/CA  CISIZE NOW  ADVISED  ROUNDED CISIZE
 0VSAM.AALG.DSET1  54    180      4096       4860       4096
 VSAM.AALG.DSET2  8    180      1536        720       1024
+VSAM.AALG.DSET2  8    180      1536        720       1024
 VSAM.AALG.DSET3  8    180      1536        720       1024
+VSAM.AALG.DSET3  8    180      1536        720       1024
 VSAM.AALG.DSET4  54    180      4096       4860       4096
```
Parsing strings in Assembler programs – part 2

This month we conclude the code to re-create some of the more common REXX string handling functions for use in Assembler programs.

SOURCE CODE FOR THE RDSPARID PROGRAM

```
RDSPARID TITLE 'ASSEMBLER ROUTINE TO EMULATE REXX INDEX'
*------------------------------------------------------------------*
* Name            : RDSPARID
* Function        : Called from RDSPARSE to perform PARSE function
*                   to emulate REXX 'INDEX' function.
*                   The source data is examined for the first
*                   occurrence of the substring. The offset within
*                   the source string is returned in the result
*                   field. Note that the offset is returned in REXX
*                   format rather than Assembler (ie offset of 1 for
*                   the first byte rather than 0). This enables the
*                   caller to determine that the substring is not
*                   found in the source data by getting a result of
*                   Ø returned.
* Attributes      : Amode(31)
*                   Rmode(Any)
*                   RENT
* Register Usage  :
* R1 - Parameters Passed : +Ø Address of Source Data :
*     +--------+
*     |LL|Source Data |
*     +--------+
```

© Xephon 2003
Address of Template List:

<table>
<thead>
<tr>
<th>Ptr</th>
<th>-&gt;</th>
<th>LL</th>
<th>Substring Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>----------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Ptr</th>
<th>-&gt;</th>
<th>LL</th>
<th>Result Data</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>---------------</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ØØØØ</th>
<th>-&gt;</th>
<th>Last Entry (Null)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>-------------------</td>
</tr>
</tbody>
</table>

R2 - pointer to source parm
R3 - pointer to template list
R4 - current template
R5 - length of source line
R6 - pointer to end of source line
R7 - pointer to start of source data
R8 -
R9 - number of words/start of correct word
R10 - branch and link
R11 -
R12 - base
R13 - Savearea

------------------------------------------------------------------------
RDSPARID CSECT
RDSPARID AMODE 31
RDSPARID RMODE ANY
BAKR R14,RØ        linkage stack
LR   R12,R15        copy entry address to base
USING RDSPARID,R12  address it
MODID
LR   R2,R1          protect parms
STORAGE OBTAIN,      grab some storage
LENGTH=WORKL,        this much
ADDR=(R13),          address in r13
MVC 4(4,R13),=C'F1SA' set acronym in save area
LR   R1,R2          restore parms
GETPARMS EQU *
LM   R2,R3,Ø(R1)    copy parms passed
XR   R5,R5          clear r5
ICM  R5,B'ØØØ1',Ø(R2) store length of source data
LA   R2,1(R2)       bump to start of source data
LR   R6,R2          copy start address
LR   R7,R2          copy start address
AR   R6,R5          point to end of source
BCTR R6,RØ          minus 1 - last char
XR   R9,R9          zero r9 - offset
XR   R11,R11         zero r11 - length of substring
XR   R8,R8          zero r8 - address of substring

WORDNUM EQU *

------------------------------------------------------------------------
* get the address of the template = substring *
------------------------------------------------------------------------

ICM  R4,B'1111',Ø(R3)  get first template addr
BZ  EXIT           if zero - quit
USING TEMPLATE, R4
LA  R8, ARG_SRC    get address of the substring
ICM R11, B'0001', ARG_LEN  get the length of the substring
BCTR R11, R0     minus one for compare
* now we get the address of the template = result field *
*-------------------------------------------------------------------*
ICM  R4, B'1111', 4(R3)  get 2nd template addr
INDXLOOP EQU *
*-------------------------------------------------------------------*
* loop thru the input text hunting for the substring *
*-------------------------------------------------------------------*
EX  R11, INDEXCLC     do the compare
BE  GOTINDEX         equal - tell user
LA  R2,1(R2)        get next byte from source
CR  R2, R6          compare against end
BH  RETURNØØ        yes - exit not found
B   INDXLOOP        loop for all input source
GOTINDEX EQU *
LR  R9, R2          copy end address
SR  R9, R7          get offset within data
LA  R9, 1(R9)       add one (otherwise asm offset)
RETURNØØ EQU *
MVI  ARG_LEN,X'04'  set length of answer to 4
STCM  R9, B'1111', ARG_SRC  store result
EXIT EQU *
STORAGE RELEASE,  free some storage X
LENGTH=WORKL,  this much X
ADDR=(R13)     address in R13
XR  R15, R15       set rc to zero
PR          return
*-------------------------------------------------------------------*
* Constants Variables and DSECTs *
*-------------------------------------------------------------------*
INDEXCLC CLC Ø(Ø, R2), Ø(R8)  compare substring
*  
TEMPLATE DSECT
ARG_LEN DS X  length of arg
ARG_SRC DS C  arg data
* 
WORKAREA DSECT
SAVEAREA DS 18D
WORKL EQU "-WORKAREA
* 
YREGS
END
SOURCE CODE FOR THE RDSPARPT PROGRAM

RDSPARPT TITLE 'PROGRAM TO PERFORM PATTERN MATCHING'
*-------------------------------------------------------------------*
* Name            : RDSPARPT
* Function        : Called from RDSPARSE to perform ISPF-like
*                   pattern matching.
*                   Wildcard character    : '*'
*                   Placeholder character : '%'
* Attributes      : Amode(31)
*                   Rmode(Any)
*                   RENT
* Register Usage  :
* R1  - Parameters Passed : +0  Address of Source Data :
*                    +-----+------------------+
*                     |LL|Source Data       |
*                    +-----+------------------+
*                    +4  Address of Template List :
*                    +-----+------------------+
*                     |Ptr | -> |LL|Pattern data    |
*                    +-----+------------------+
*                     |Ptr | -> |LL|Result data     |
*                    +-----+------------------+
*                     |0000| -> |Last Entry (Null)|
*                    +-----+------------------+
* R2  - raw data string
* R3  - list of templates
* R4  - template
* R5  - n/a
* R6  - end of source data
* R7  - end of pattern
* R8  - address of pattern data
* R9  -
* R10 - branch and link
* R11 - result (0=nomatch,1=match)
* R12 - base
* R13 - SaveArea
*-------------------------------------------------------------------*

RDSPARPT CSECT
RDSPARPT AMODE 31
RDSPARPT RMODE ANY
BAKR R14,R0
LR R12,R15
USING RDSPARPT,R12
MODID
LR R2,R1
STORAGE OBTAIN,
LENGTH=WORKL,          this much                        X
ADDR=(R13)             address in r13
USING WORKAREA, R13
MVC 4(,R13),=C'F1SA'
LR R1,R2                  restore parms
GETPARMS EQU *
LM R2,R3,0(R1)            load up parms
XR R5,R5                  clear r5
ICM R5,B'0001',0(R2)      store length of source data
ST R5,STRLEN             remember length
LA R2,1(R2)               bump to start of source data
XR R7,R7                  zero r9
XR R11,R11                zero r11

*-------------------------------------------------------------------*
* parms passed :                                                    *
* R2 ---> address of raw data                                       *
* R3 ---> address of template list                                  *
* R4 ---> current template                                          *
*-------------------------------------------------------------------*
ICM R4,B'1111',0(R3)      get template address
USING TEMPLATE, R4        address it
ICM R7,B'0001',ARG_LEN    get length of pattern
ST R7,PATLEN              Store pattern length
LA R8,ARG_SRC             get result template
ICM R4,B'1111',4(R3)      get result template
ASAXWC PATTERNSTR=(R8),    x
  PATTERNSTRLEN= PATLEN,   x
  STRING=(R2),           x
  STRINGLEN= STRLEN,     x
  ZERO OR MORE=ASTERIX,  x
  ONE CHAR= PERCENT,     x
  MF=(E,PATTERN1)
LTR R15,R15
BZ MATCH

NOMATCH EQU *
XR R11,R11                set result
B RETURN0000              return

MATCH EQU *
LA R11,1                   set result
RETURN0000 EQU *
MVI ARG_LEN,X'04'          set length
STCM R11,B'1111',ARG_SRC   store result
STORAGE RELEASE,           free some storage
  LENGTH=WORKL,          this much                        X
  ADDR=(R13)             address in r13
XR R15,R15                set rc
PR                           return

*-------------------------------------------------------------------*
* Constants Variables and DSECTs                                      *

REXX
SOURCE CODE FOR THE RDSPARST PROGRAM

RDSPARST TITLE 'ASSEMBLER ROUTINE TO EMULATE REXX STRIP FUNCTION'

* NAME            : RDSPARST
* Function        : Called from RDSPARSE to perform PARSE function
to emulate REXX 'STRIP' function.
* Attributes      : Amode(31)
                   : Rmode(Any)
                   : RENT
* Register Usage  :

* R1  - Parameters Passed : +Ø Address of Source Data :
  ++++++-------------------+
  |LL|Source Data         |
  ++++++-------------------+
  +4 Address of Template List :
  ++++++-------------------+
  |Ptr| -> |LL|Strip Character    |
  ++++++-------------------+
  |Ptr| -> |LL|Strip Option      |
  ++++++-------------------+
  |Ptr| -> |LL|Result Data       |
  ++++++-------------------+
  |ØØØØ| -> |Last Entry (Null)  |

* R2  - pointer to source parm
* R3  - pointer to template list
* R4  - current template
RDSPARST CSECT
RDSPARST AMODE 31
RDSPARST RMODE ANY

BAKR R14,R0
LR R12,R15
USING RDSPARST,R12
MODID
LR R2,R1
STORAGE OBTAIN, LENGTH=WORKL, ADDR=(R13)
USING WORKAREA,R13
MVC 4(4,R13),=C'F1SA'
LR R1,R2
GETPARMS EQU *
LM R2,R3,Ø(R1)
XR R5,R5
ICM R5,B'0001',Ø(R2)
LA R2,1(R2)
BCTR R5,RØ
EX R5,PARMCOPY
LA R2,COPYTEXT
LR R6,R2
AR R6,R5
XR R8,R8
LA R5,1(R5)

STRIPOPT EQU *
ICM R4,B'1111',Ø(R3)
BZ EXIT
USING TEMPLATE,R4
LA R10,ARG_SRC

STRIPCHR EQU *
ICM R4,B'1111',Ø(R3)
BZ EXIT
USING TEMPLATE,R4
LA R10,ARG_SRC

* R5 - length of source line
* R6 - pointer to end of source line
* R7 - pointer to length of result
* R8 - pointer to result
* R9 - pointer to strip option
* R10 - pointer to strip char
* R11 - n/a
* R12 - base
* R13 - SaveArea
USING TEMPLATE, R4
LA R9, ARG_SRC
RESULT EQU *

address it
point to the strip option

* get the address of the template = result *

ICM R4, B'1111', 8(R3)  get 3rd template addr
BZ EXIT  if zero - quit
USING TEMPLATE, R4  address it
LA R7, ARG_LEN  point to the result field length
LA R8, ARG_SRC  point to the result field
STCM R5, B'0001', 0(R7)  copy length field

FRONT EQU *
TM 0(R9), LEADING  strip leading chars ?
BNO BACK  no - strip trailing

STRIP1ST EQU *
CLC 0(1, R2), 0(R10)  is it the strip char ?
BNE BACK  no - strip finished
LA R2, 1(R2)  get next source char
XR R15, R15  zero reg 15
ICM R15, B'0001', 0(R7)  load up length
BZ STRIPEND  if zero - we have finished
BCTR R15, R0  subtract 1
STCM R15, B'0001', 0(R7)  store it back
CR R2, R6  end of source ?
BLN STRIPEND  yes - return
B STRIP1ST  and try again

BACK EQU *
TM 0(R9), TRAILING  strip trailing chars ?
BNO STRIPEND  no - return

STRIP2ND EQU *
CLC 0(1, R6), 0(R10)  is it the strip char ?
BNE STRIPEND  no - trailing strip finished
BCTR R6, R0  get next source char
XR R15, R15  zero reg 15
ICM R15, B'0001', 0(R7)  load up length
BZ STRIPEND  if zero - we have finished
BCTR R15, R0  subtract 1
STCM R15, B'0001', 0(R7)  store it back
CR R6, R2  end of source ?
BNH STRIPEND  yes - return
B STRIP2ND  and try again

STRIPEND EQU *
ICM R15, B'0001', 0(R7)  load up length of result
BZ EXIT  if zero - no data
BCTR R15, R0  subtract 1
EX R15, MOVERES  move in result field

EXIT EQU *
STORAGE RELEASE,  free some storage
LENGTH=WORKL,  this much

ADDR=(R13)             address in r13
XR R15,R15             set rc to zero
PR                   return
*-------------------------------------------------------------------*
* Constants Variables and DSECTs                                 *
*-------------------------------------------------------------------*
PARMCOPY MVC COPYTEXT(Ø),Ø(R2)      executed move
MOVERES MVC Ø(R,8),Ø(R2)          executed move
LEADING EQU X'FØ'                  mask for leading chars
TRAILING EQU X'ØF'                  mask for trailing chars
* TEMPLATED SECT
ARG_LEN DS X                      length of arg
ARG_SRC DS C                      arg data
* WORKAREA DSECT
SAVEAREA DS 18D
COPYTEXT DS CL256
WORKL EQU *-WORKAREA
*
YREGS
END

SOURCE CODE FOR THE RDSPARVR PROGRAM

RDSPARVR TITLE 'ASSEMBLER ROUTINE TO EMULATE REXX PARSE VAR'
*------------------------------------------------------------------*
* Name            : RDSPARVR                                      *
* Function        : Called from RDSPARSE to perform PARSE function  *
*                   to emulate REXX 'PARSE VAR'.                   *
*                   The program is passed a list of templates that *
*                   can be either SEPARATORS or RESULT fields.      *
*                   In the template list, if the LL field is X'00' *
*                   then it is a RESULT field, otherwise it is a    *
*                   SEPARATOR.                                      *
*                   If two RESULT fields occur in the template list *
*                   without a SEPARATOR field in-between, then a    *
*                   default separator of a spaces is assumed and the *
*                   second result field will start at the next non- *
*                   blank byte.                                     *
*                   A special result field of '.' can be used to    *
*                   indicate that the result data can be thrown      *
*                   away.                                          *
* Attributes      : Amode(31)                                    *
Register Usage:

- **R1** - Parameters Passed: +Ø Address of Source Data:
  - +---+------------------+
  - |LL|Source Data      |
  - +---+------------------+
  - +4 Address of Template List:
    - +---+------------------+
    - |Ptr| -> |LL|Result/Separator |
    - +---+------------------+
    - |...| -> |LL|Result/Separator |
    - +---+------------------+
    - |ØØØØ| -> |Last Entry (Null) |
    - +---+------------------+

- **r2** - pointer to source parm
- **r3** - pointer to template list
- **r4** - current template
- **r5** - length of source line
- **r6** - pointer to end of source line
- **r7** - pointer to result length
- **r8** - pointer to result
- **r9** - length of result
- **r10** - branch and link
- **r11** -
- **r12** - base
- **r13** - workarea

RDSPARVR CSECT
RDSPARVR AMODE 31
RDSPARVR RMODE ANY

BAKR R14,RØ Linkage stack
LR R12,R15 copy entry address
USING RDSPARVR,R12 address it

GETPARMS EQU *
LM R2,R3,Ø(R1) copy parms passed

GETSTOR EQU *
STORAGE OBTAIN, get the workarea storage X
LENGTH=WORKLEN, this much X
ADDR=(R11), put address in r11 X
SP=Ø,KEY=8, subpool Ø storage key 8 X
LOC=BELOW, below the line X
COND=NO unconditional
USING WORKAREA,R11 address workarea
LA R13,SAVEAREA  point to savearea
MVC 4(4,R13),="C'F1SA'"  set label in savearea
XR R5,R5  clear r5
ICM R5,B'0001',Ø(R2)  store length of source data
LA R2,1(R2)  bump to start of source data
LR R6,R2  copy start address
AR R6,R5  point to end of source
BCTR R6,R0  minus 1 = last char
XR R8,R8  zero r8 - address of result
XR R9,R9  zero r9 - length of result
MVI SPECIAL,X'00'  reset flag

PARSE EQU *

* -------------------------------------------------------------------*
* Now we get the address of the template and examine it to see if *
* we have reached the last template or not.                         *
* -------------------------------------------------------------------*
ICM R4,B'1111',Ø(R3)  get first template addr
USING TEMPLATE,R4  address the dsect
BNZ ISRESULT  if non-zero - have result area

* -------------------------------------------------------------------*
* if we get here we have processed all the templates passed to *
* the program. if there is a current result area, and we have some *
* source data left - we copy the remaining data into the result *
* area.                                                             *
* this is like placing data into the result3 field in the rexx *
* statement:                                                        *
* parse var source result1 (sep1) result2 (sep2) result3 *
* -------------------------------------------------------------------*
LTR R8,R8  is there a current result address
BZ RETURN00  no - no need to copy remainder
TM SPECIAL,DOT  was it a dot ?
BO RETURN00  yes - no need to copy rest
LR R15,R6  get address of end of source data
SR R15,R2  subtract where we are = length
LA R15,1(R15)  add one for length
STCM R15,B'0001',Ø(R7)  store the length in the result
BCTR R15,R0  minus 1 for executed move
EX R15,MOVEREST  move in the rest of the source
B RETURN00  and exit

ISRESULT EQU *

* -------------------------------------------------------------------*
* check to see what kind of template it is                          *
* -------------------------------------------------------------------*
CLI ARG_LEN,RESULT  is it a result area ?
BNE ISSEP  no - must be separator

* -------------------------------------------------------------------*
* It is a result area, so we point to the result address and its *
* length field to make them current.                               *
* Special circumstances                                           *
* (1): If the first byte of the result area is a *
*       dot ('.') we indicate that we do not want *
* any data (just like in rexx). *
* (2): Two consecutive result area templates in a *
* row force a default separator of a space *
* to be used (like rexx). *

```
LTR  R8,R8          have we already got a result ?
BZ   NORM_RES       no - normal
MVI  SEP_LEN,X'Ø1'  yes - set length of one
MVI  SEP_DATA,C' '  and default sep to space
OI   SPECIAL,SPACES and set flag
S    R3,=F'4'       and point back an arg
B    COMPARE        force a sep of space
NORM_RES EQU *
LA   R7,ARG_LEN     get address of result length
LA   R8,ARG_SRC     get address of result
CLI  Ø(R8),C'.'     is it a dot (as in rexx)
BNE  NEXT_ARG       no - get next arg
OI   SPECIAL,DOT    indicate dot
B    NEXT_ARG       get next arg
ISSEP EQU *

* It is a separator, so we copy the source data into the current *
* result area (if there is one) byte by byte until we find the *
* separator data in the source. *

```

```
ICM  R15,B'ØØØ1',ARG_LEN    get length for execute
STCM R15,B'ØØØ1',SEP_LEN    store the length
BCTR R15,R0                  minus one for the move
EX   R15,MOVESEP             copy the data to the workarea
COMPARE EQU *
ICM  R15,B'ØØØ1',SEP_LEN    get length for execute
BCTR R15,R0                  minus one for clc
EX   R15,PARSECLC            do the compare
BNE  COPYBYTE                not equal - copy into result

* We have found the separator - reset the result address and length *
* fields and go and get the next template. *
```

```
XR   R8,R8             equal - reset result address
XR   R9,R9             and result length
XR   R15,R15           zero r15
ICM  R15,B'ØØØ1',ARG_LEN get length for separator
AR   R2,R15            - point past separator
TM   SPECIAL,SPACES    was it special spaces ?
BNE  NOSPACES          no - carry on as normal
BAL  R10,FINDWORD      yes - find next word
NOSPACES EQU *
MVI  SPECIAL,X'Ø0'     reset flag
B    NEXT_ARG          - get next arg
COPYBYTE EQU *
```

We have not found the separator yet - so copy the current byte into the result field and update the length field.

NB - The byte-by-byte copy is bypassed if there is no active result field. This is like the rexx statement:
\[\text{parse var source (sep1) result1}\]

```assembly
LA R9,1(R9)     \text{add one to length of result}
LTR R8,R8       \text{is there a result address}
BZ NO_MVC       \text{no - don't copy byte}
TM SPECIAL,DOT  \text{was it a dot?}
BO NO_MVC       \text{yes - don't copy byte}
MVC $0(1,R8),0(R2) \text{copy the source to result}
LA R8,1(R8)     \text{shuffle thru the result}
STCM R9,B'0001',0(R7) \text{store the result length}

NO_MVC EQU *
LA R2,1(R2)     \text{shuffle thru source}
CR R2,R6       \text{is it end of source?}
BH RETURNØØ    \text{yes - get out}
B COMPARE       \text{look for separator text again}

NEXT_ARG EQU *
LA R3,4(R3)     \text{get next template address}
B PARSE         \text{and parse again}

RETURNØØ EQU *
BAL R10,FREESTOR \text{free the workarea}
XR R15,R15      \text{set rc to zero}
PR              \text{return}

FREESTOR EQU *

FINDWORD EQU *

STORAGE RELEASE, \text{release workarea storage}
LENGTH=WORKLEN, \text{this much}
ADDR=(R11), \text{address in r11}
SP=$0,KEY=8, \text{subpool 0 storage key 8}
COND=NO \text{unconditional}
BR R10 \text{return}

FINDWORD EQU *

CLI $0(R2),C' ' \text{is it a space}
BNER R10 \text{no - must be start next word}
LA R2,1(R2) \text{get next byte}
CR R2,R6 \text{is it end of source?}
BH RETURNØØ \text{yes - get out}
B FINDWORD \text{no - keep looking}
```

* Constants Variables and DSECTs

© 2003, Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
SOURCE CODE FOR THE RDSPARWD PROGRAM

**RDSPARWD** TITLE 'ASSEMBLER ROUTINE TO EMULATE REXX WORD FUNCTION'  
*-------------------------------------------------------------------*

* Name            : RDSPARWD  
* Function        : Called from RDSPARSE to perform PARSE function  
*                   to emulate REXX 'WORD'.  
* Attributes      : Amode(31)  
*                   Rmode(Any)  
*                   RENT  
* Register Usage  :  
* R1  - Parameters Passed : +Ø Address of Source Data :  
*                    +---+------------------+
*                    |LL|Source Data       |
*                    +---+------------------+
* +4 Address of Template List :  
*                    +---+------------------+
*                    |Ptr|Word Number        |
*                    +---+------------------+
*                    +---+------------------+
*                    |Ptr|Result Data        |
*                    +---+------------------+
*                    +---+------------------+
*                    |ØØØØ|Last Entry (Null)   |
*                    +---+------------------+
*
* r2 - pointer to source parm
* r3 - pointer to template list
* r4 - current template
* r5 - length of source line
* r6 - pointer to end of source line
* r7 - pointer to length of result
* r8 - pointer to result
* r9 - number of words/start of correct word
* r10 - branch and link
* r11 - word number
* r12 - base
* r13 - SaveArea

*------------------------------------------------------------------*
RDSPARWD CSECT
RDSPARWD AMODE 31
RDSPARWD RMODE ANY

BAKR R14, R0
LR R12, R15
USING RDSPARWD, R12
MODID
LR R2, R1
STORAGE OBTAIN, LENGTH=WORKL, ADDR=(R13)
MVC 4(4, R13), =C'F1SA'
LR R1, R2
GETPARMS EQU *

LM R2, R3, 0(R1)
XR R5, R5
ICM R5, B'0001', 0(R2)
LA R2, 1(R2)
LR R6, R2
AR R6, R5
BCTR R6, R0
XR R8, R8
XR R9, R9

WORDNUM EQU *

* get the address of the template = word number

*------------------------------------------------------------------*
ICM R4, B'1111', 0(R3)
BZ EXIT
USING TEMPLATE, R4
ICM R11, B'1111', ARG_SRC

* now we get the address of the template = result field

*------------------------------------------------------------------*
ICM R4, B'1111', 4(R3)
LA R8, ARG_SRC
LA R7, ARG_LEN
WORDLOOP EQU *
* loop thru the input text hunting for the correct word number *
*-------------------------------------------------------------------*
BAL R1Ø,FINDWORD           get a word
LA R9,1(R9)                add to word count
CR R9,R11                  is it correct word number ?
BE GOTWORD                 yes - get the word
BAL R10,FINDSPCE           get next space
B WORDLOOP                 loop for all input source

GOTWORD EQU *
LR R9,R2                   use r9 for word start
XR R11,R11                 indicate word found

FINDEND EQU *
BAL R10,FINDSPCE           find a space

RETURNØØ EQU *
LTR R11,R11                was word found ?
BNZ EXIT                   no - bypass result
LR R15,R2                  store address of end of word
SR R15,R9                  get length of word
STCM R15,'0001',Ø(R7)      store the word length
BCTR R15,MOVEWORD          subtract one for move
EX R15,MOVEWORD           move in the result

EXIT EQU *
STORAGE RELEASE,          free some storage
LENGTH=WORKL,              this much
ADDR=(R13)                 address in r13
XR R15,R15                 set rc to zero
PR                         return

FINDWORD EQU *
* routine to hunt for the beginning of a word (non-space) *
*-------------------------------------------------------------------*
CLI Ø(R2),C ' '           is it a space ?
BNER R10                  no - must be start next word
LA R2,1(R2)               get next byte
CR R2,R6                  is it end of source ?
BH RETURNØØ               yes - get out
B FINDWORD                no - keep looking

FINDSPCE EQU *
* routine to hunt for the end of a word (space) *
*-------------------------------------------------------------------*
CLI Ø(R2),C ' '           is it a space ?
BER R10                   yes - must be end of word
LA R2,1(R2)               get next byte
CR R2,R6                  is it end of source ?
BH RETURNØØ               yes - get out
B      FINDSPCE               no - keep looking
*-------------------------------------------------------------------*
* Constants variables and DSECTs                                    *
*-------------------------------------------------------------------*
MOVEWORD MVC Ø(Ø,R8),Ø(R9)             executed move of result
*
TEMPLATE DSECT
ARG_LEN  DS    X                      length of arg
ARG_SRC  DS    C                      arg data
*
WORKAREA DSECT
SAVEAREA DS   18D
WORKL    EQU   *- WORKAREA
*
YREGS
END

SOURCE CODE FOR THE RDSPARWI PROGRAM

RDSPARWI TITLE 'ASSEMBLER ROUTINE TO EMULATE REXX WORDINDEX'
*------------------------------------------------------------------*
* Name            : RDSPARWI                                       *
* Function        : Called from PARSE to perform PARSE function    *
*                   to emulate REXX 'WORDINDEX'                     *
* Attributes      : Amode(31)                                      *
*                   Rmode(Any)                                     *
*                   RENT                                          *
* Register Usage  :                                               *
* R1  - Parameters Passed : +Ø  Address of Source Data :          *
*                  +----------------------------------------------+
*                  | LL|Source Data          |                                *
*                  +----------------------------------------------+
*                  |ØØØØ| -> |Last Entry (Null)    |                                *
*                  +----------------------------------------------+
* r2  - pointer to source parm                                    *
* r3  - pointer to template list                                  *
* r4  - current template                                          *
* r5  - length of source line                                    *
* r6  - pointer to end of source line                            *
* r7 - pointer to start of source data
* r8 -
* r9 - number of words/start of correct word
* r10 - branch and link
* r11 - word number
* r12 - base
* r13 - Save Area

---------------------------------------------
RDSPARWI CSECT
RDSPARWI AMODE 31
RDSPARWI RMODE ANY
  BAKR R14,RØ           linkage stack
  LR R12,R15            copy entry address to base
  USING RDSPARWI,R12    address it
MODID
  LR R2,R1              protect parms
  STORAGE OBTAIN,       grab some storage    X
    LENGTH=WORKL,       this much            X
    ADDR=(R13)          address in r13
  MVC 4(4,R13),=C'F1SA' set acronym in save area
  LR R1,R2              restore parms

GETPARMS EQU *
  LM R2,R3,Ø(R1)        copy parms passed
  XR R5,R5              clear r5
  ICM R5,B'0001',Ø(R2)  store length of source data
  LA R2,1(R2)           bump to start of source data
  LR R6,R2              copy start address
  LR R7,R2              copy start address
  AR R6,R5              point to end of source
  BCTR R6,RØ            minus 1 = last char
  XR R9,R9              zero r9 - number of words
  XR R15,R15            zero r15 - offset

WORDNUM EQU *
  *-------------------------------------------------------------------*
  * get the address of the template = word number                    *
  *-------------------------------------------------------------------*
  ICM R4,B'1111',Ø(R3)  get first template addr
  BZ EXIT                if zero - quit
  USING TEMPLATE,R4
  ICM R11,B'1111',ARG_SRC get word number

  *-------------------------------------------------------------------*
  * now we get the address of the template = result field            *
  *-------------------------------------------------------------------*
  ICM R4,B'1111',4(R3)  get first template addr

WORDLOOP EQU *
  *-------------------------------------------------------------------*
  * loop thru the input text hunting for the correct word number     *
  *-------------------------------------------------------------------*
  BAL R10,FINDWORD       get a word
  LA R9,1(R9)            add to word count
CR R9, R11 is it correct word number ?
BE GOTWORD yes - get the word
BAL R10, FINDSPCE get next space
B WORDLOOP loop for all input source

GOTWORD EQU *
LR R15, R2 copy end address
SR R15, R7 get off set within word
LA R15, 1(R15) add one (otherwise as m offset)

RETURNØØ EQU *
MVI ARG_LEN, X’04’ set length of answer to 4
STCM R15, B’1111’, ARG_SRC store result

EXIT EQU *
STORAGE RELEASE, free some storage
LENGTH=WORKL,
ADDR=(R13) address in r13
XR R15, R15 set rc to zero
PR return

FINDWORD EQU *
* routine to hunt for the beginning of a word (non-space) *
* ******************************************************* *
CLI Ø(R2), C’ ‘ is it a space ?
BNER R1Ø no - must be start next word
LA R2, 1(R2) get next byte
CR R2, R6 is it end of source ?
BH RETURNØØ yes - get out
B FINDWORD no - keep looking

FINDSPCE EQU *
* routine to hunt for the end of a word (space) *
* *********************************************** *
CLI Ø(R2), C’ ‘ is it a space ?
BER R1Ø yes - must be end of word
LA R2, 1(R2) get next byte
CR R2, R6 is it end of source ?
BH RETURNØØ yes - get out
B FINDSPCE no - keep looking

* Constants variables and DSECTs *
* *********************************** *

TEMPLATE DSECT
ARG_LEN DS X length of arg
ARG_SRC DS C arg data

WORKAREA DSECT
SAVEAREA DS 18D
WORKL EQU *- WORKAREA
SOURCE CODE FOR THE RDSPARWS PROGRAM

**RDSPARWS**

**TITLE** 'ASSEMBLER ROUTINE TO EMULATE REXX WORDS FUNCTION'

*------------------------------------------------------------------*

* Name            : RDSPARWS
*
* Function        : Called from RDSPARSE to perform PARSE function 
                   to emulate REXX 'WORDS'
*
* Attributes      : Amode(31)
                   Rmode(Any)
                   RENT
*
* Register Usage  :
*
* R1  - Parameters Passed : +Ø  Address of Source Data :
  +---+------------------+
  |LL|Source Data      |
  +---+------------------+
  +4  Address of Template List :
  +---+------------------+
  |Ptr|Result Data      |
  +---+------------------+
  |ØØØØ|Last Entry (Null) |
  +---+------------------+

* r2  - pointer to source parm
* r3  - pointer to template list
* r4  - current template
* r5  - length of source line
* r6  - pointer to end of source line
* r7  -
* r8  - pointer to result
* r9  - number of words
* r10 - branch and link
* r11 -
* r12 - base
* r13 - workarea
*------------------------------------------------------------------*

**RDSPARWS CSECT**

**RDSPARWS AMODE 31**

**RDSPARWS RMODE ANY**

BAKR R14,RØ linkage stack
LR    R12,R15 copy entry address to base
USING RDSPARWS,R12 address it
MODID
LR    R2,R1 protect parms
STORAGE OBTAIN, grab some storage X
LENGTH=WORKL,           this much
ADDR=(R13)              address in r13
MVC 4(4,R13),=C'F1SA'   set acronym in save area
LR R1,R2                restore parms

GETPARMS EQU *
LM R2,R3,Ø(R1)          copy parms passed
XR R5,R5                clear r5
ICM R5,B'0001',Ø(R2)    store length of source data
LA R2,1(R2)             bump to start of source data
LR R6,R2                copy start address
AR R6,R5                point to end of source
BCTR R6,RØ              minus 1 = last char
XR R8,R8                zero r8 - address of result
XR R9,R9                zero r9 - number of words

WORDS EQU *

*-------------------------------------------------------------------*
* now we get the address of the template = result field             *
*-------------------------------------------------------------------*
ICM R4,B'1111',Ø(R3)   get first template addr
BZ EXIT                 if zero - quit
USING TEMPLATE,R4
LA R8,ARG_SRC           get address of result

WORDLOOP EQU *
*-------------------------------------------------------------------*
* loop thru the input text hunting for each word                    *
*-------------------------------------------------------------------*
BAL R1Ø,FINDWORD        get a word
LA R9,1(R9)             add to word count
BAL R1Ø,FINDSPACE       get next space
B WORDLOOP               loop for all input source

RETURNØØ EQU *
MVI ARG_LEN,X'Ø4'       store length of result
STCM R9,B'1111',ARG_SRC store number of words in result

EXIT EQU *
STORAGE RELEASE,        free some storage
LENGTH=WORKL,           this much
ADDR=(R13)              address in r13
XR R15,R15              set rc to zero
PR                        return

FINDWORD EQU *
*-------------------------------------------------------------------*
* routine to hunt for the beginning of a word (non-space)           *
*-------------------------------------------------------------------*
CLI Ø(R2),C' '           is it a space ?
BNER R1Ø                 no - must be start next word
LA R2,1(R2)              get next byte
CR R2,R6                 is it end of source ?
BH RETURNØØ              yes - get out
B FINDWORD               no - keep looking
* FINDSPCE EQU *
* --------------- *
* routine to hunt for the end of a word (space) *
* --------------- *

CLI Ø(R2), C' ' is it a space?
BER R10 yes - must be end of word
LA R2, 1(R2) get next byte
CR R2, R6 is it end of source?
BH RETURNØØ yes - get out
B FINDSPCE no - keep looking

* Constants variables and DSECTs *
* ------------------------------- *

TEMPLATE DSECT
ARG_LEN  DS X       length of arg
ARG_SRC  DS C       arg data

WORKAREA DSECT
SAVEAREA DS 18D
WORKL   EQU * WORKAREA

YREGS
END

INSTALLING THE PARSE PROGRAM AND MACRO

Use the following JCL as a skeleton to assemble and link the PARSE routines into the main PARSE program. Once complete, the PARSE program will need to be made available to users via a STEPLIB or placed in the system LINKLIST or LPALIST.

//jobname JOB ..
//*
//* //ASMPARSE PROC MEMBER=
//ASM EXEC PGM=IEV9Ø, REGION=6Ø0K, PARM='RENT'
//SYSPRINT DD SYSOUT=* //SYSIN DD DSN=your.own.asm(&MEMBER), DISP=SHR
//SYSLIB DD DSN=SYS1.MACLIB, DISP=SHR
//* DD DSN=SYS1.MODGEN, DISP=SHR
//SYSUT1 DD UNIT=SYSDA, SPACE=(CYL,(2,1))
//* //SYSLIN DD DSN=your.own.obj(&MEMBER), DISP=SHR
//* //PEND
//* //VAR EXEC ASMPARSE, MEMBER=RDSPARVR
//* //WORD EXEC ASMPARSE, MEMBER=RDSPARWD
//* //WORDS EXEC ASMPARSE, MEMBER=RDSPARWS
The PARSE macro must be copied to an installation MACLIB and included in the SYSLIB concatenation of any assembly JCL that uses it.

Rob Scott
MVS Consultant (USA) © Rob Scott 2003

MVS Update on the Web

Code from individual articles of MVS Update, and complete issues in Acrobat PDF format, can be accessed on our Web site, at:

http://www.xephon.com/mvs

You will be asked to enter a word from the printed issue.
I haven’t seen an example of using a panel exit in a long time and think we have a particularly useful one. I took the existing ISPF Edit and Browse member selection panels, and added a call to a PANEL EXIT, which we use to replace the member statistics in the selection list with a single line description for each member. The single line description is taken directly from within the members that are listed on the selection panel.

Over the years, I have seen many ways that people have tried of keeping track of the contents of their PDS members. The limitation of eight characters is just too small to allow even the most structured of Assembler programmers to know what they have and where, after just a few years of accumulating members in their personal JCL or MACRO PDSs. I have seen people successfully use an index member, usually prefixed with a special character so that it floats to the top of a member list such as $$index. This arrangement is not too hard to use, particularly if you write a quick set of edit macros to put you into edit mode on the $$index member while you are editing existing members. There are drawbacks, however, to bouncing back and forth between a member selection list and an index member. The $$index member doesn’t automatically reflect the fact that members may have been renamed, deleted, or moved to another PDS. It can be frustrating to search a manually maintained $$index member to find what you want, only to discover that the member no longer exists.

I thought a better solution would be to keep the documentation for each member, within the member itself, and use the ISPF PANEXIT facility to replace the member statistics with a single line description for each member right on the member selection list panel. With the PANEXIT, we can simply scroll through the normal selection lists to find what we need. Since we do our formatting at member list display time, we read and format
enough information for one panel at a time, and processing a 2000-member dataset presents no more overhead than a dataset with only 20 or 30 members. Moreover, this documentation method provides data encapsulation, since the descriptive information travels with the member. Copying, renaming, or otherwise moving a member does not affect the documentation – it travels with the member. Likewise, deleting a member automatically does away with the associated documentation.

The processing logic is fairly straightforward. We leave an ISPF variable named SHOWDAT permanently in the user’s ISPF profile, and whenever a member selection panel is displayed we query its value, either SHOWME or NOSHOW. The selection list portion of the panel is contained in a single ISPF variable that we can pass to the panel exit for modification. During )INIT and )REINIT panel processing, if the SHOWDAT variable has been set to SHOWME, we save the existing display variable, and pass a copy of it to the panel exit. The panel exit then scans through the display variable, clearing the member statistics and replacing them with the member description information for each of the members in the current selection list – those that are in the display variable. To accomplish this, we first dynamically allocate the dataset and open it for read processing as a partitioned dataset. We save the existing statistics information, and then clear it from the variable. We scan the variable looking for the member names, and then issue a BLDL for each member in turn, followed by a POINT to direct us to the data for each member. After we are taken to the member data, we scan for a limited number of lines looking for our description data and, if found, we copy it into the variable where the member statistics originally were.

A couple of final considerations – we actually look for several different formats of documentation record so that they can exist as comments in the members: one format for JCL comments, another for Assembler source statements, and yet another for CLIST comments. We also arbitrarily limit how far we read into a member to look for documentation lines; that way if we have members with several thousand lines, and no documentation

© 2003. Xaphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
records, we don’t waste too much time reading all the way through them. Finally, when we are done with all of the members on the panel selection list, if we haven’t found any members with documentation lines, we restore the original statistics rather than display a panel selection list with just member names. We also choose to track and set the SHOW/NOSHOW variable right in the panel selection list that we modify with the PANEXIT statement, although it may have been more appropriate to do it with a simple ISPF command table entry.

Here is a sample of our panel changes:

• In the )INIT section:

  VGET (SHOWDAT) PROFILE
  &ZHOLD = &ZDATA /* hold a copy of the screen data - just in case*/
  PANEXIT((SHOWDAT,ZDSNT,ZDATA),LOAD,'ITPDSDAT') /*reformat if needed*/

• In the )REINIT section:

  VGET (SHOWDAT) PROFILE
  &ZHOLD = &ZDATA /* SAVE NEW COPY OF STATS */
  PANEXIT((SHOWDAT,ZDSNT,ZDATA),LOAD,'ITPDSDAT') /*reformat if needed*/

• In the )PROC section:

  VGET (SHOWDAT) PROFILE
  IF (&ZCMD = 'SHOWME','NOSHOW ')
    &ZDATA = &ZHOLD /* restore &zdata from &zhold */
    &SHOWDAT = &ZCMD /* save command literal in &showdat */
    &ZCMD = '' /* clear the command line*/
  VPUT (SHOWDAT) PROFILE

The format we look for in the data to use as a description can be any of the following formats:

"* *%PDSDOC ØØ any text you want to display for this member goes here"
"/* %PDSDOC ØØ any text you want to display for this member goes here"
"// *%PDSDOC ØØ any text you want to display for this member goes here"

I had intended to extend the idea and create documentation lines 01, 02, 03, etc for additional details that could be pulled out by a batch job to create extended documentation (a sort of master documentation list) at some later time, but have not yet done it.

To put a finishing touch to the change, I created a couple of simple edit macros to insert a model line, so that I don’t have to
continually look up the format of the documentation lines. Here is a sample of my DOCA macro that I use to insert the Assembler source. I have similar edit macros that create the JCL comment style card, and one for CLISTs as well, the only difference being the actual format of the card that gets inserted into my data.

```/* %PDSDOC ØØ EDIT MACRO TO ADD SAMPLE DOCUMENTATION LINES -  
ASM SRC */ 
ISREDIT MACRO () NOPROCESS 
/*************************************************************************/ 
/* INSERT ASSEMBLER FORMAT COMMENT LINE IN SOURCE FOR DOCUMENTATION */ 
/*************************************************************************/ 
ISREDIT (MEM) = MEMBER 
ISREDIT (DSN) = DATASET 
IF &MEM = THEN SET &DSNX = &STR(&DSN) 
IF &MEM NE THEN SET &DSNX = &STR(&DSN(&MEM)) 
ISREDIT PROCESS DEST 
IF &LASTCC = Ø THEN DO 
   ISREDIT LOCATE .ZDEST 
   IF &LASTCC = 8 THEN GOTO EMPTY 
   ISREDIT MASKLINE = "* *%PDSDOC ØØ DESCRIPTION GOES IN HERE" 
   ISREDIT LINE_AFTER .ZDEST = MASKLINE 
   ISREDIT FIND "DESCRIPTION GOES IN HERE" 
   GOTO ERROROUT 
END 
ELSE DO 
   EMPTY: + 
   ISPEXEC VGET ZLLGJOB1 
   ISREDIT MASKLINE = "* *%PDSDOC ØØ DESCRIPTION GOES IN HERE" 
   ISREDIT LINE_AFTER Ø = MASKLINE 
   ISREDIT FIND "DESCRIPTION GOES IN HERE" 
   ERROROUT:+ 
   ISREDIT MASKLINE = ' ' 
   SET RC = Ø 
END 
EXIT CODE(&RC)

And finally the panel exit code itself:

```* * INVOKED BY THE FOLLOWING IN THE PANEL DEFINITIONS * *
* * PANEXIT((SHOWDAT,ZDSNT,ZDATA),LOAD,'ITPDSDAT') * *
* *%PDSDOC ØØ PANEL EXIT TO SUPPORT THE SHOWME AND NOSHOW COMMANDS
  PUNCH ' SETOPT PARM(REUS=REFR,AMODE=31,RMODE=24,AC=Ø) '
  PUNCH ' ENTRY ITPDSDAT ' 
* *.............................................................*.*
* * PROGRAM NAME - ITPDSDAT * *```
* * FUNCTION - PROVIDE DATA FOR MEMBER LIST DISPLAYS *
* *.
* *
* * REG. USAGE.
* *
* * R0 - LINKAGE R4 - IN BUF  R8 - BAL (LVL1)  R12 - BASE
* *
* * R1 - LINKAGE R5 - WORK   R9 - BAL (LVL2)  R13 - TEMP STOR
* *
* * R2 - WORK   R6 - WORK   R10 - OPEN   R14 - LINKAGE
* *
* * R3 - WORK  R7 - WORK   R11 - OPEN   R15 - LINKAGE
* *
* *
* * INPUT PARMS - VDATA, COMMAND, AND DSN
* *
* * OUTPUT PARMS - VDATA AND COMMAND MAY BE UPDATED
* *
* * * *-----------------------------------------------------------*-*-*

ITPDSDAT CSECT
ITPDSDAT AMODE 31
ITPDSDAT RMODE ANY
COPY REGEQU
USING ITPDSDAT,15 TEMP ADDRESSABILITY
B     @PROLOG AROUND EYECATCHER
DC    C'ITPDSDAT - DATA FOR SEL. LISTS &SYSDATE'
@PROLOG BAKR R14,R0 SAVE REGISTER/PSW STATUS
LR    R8,R1 SAVE PARM POINTER
LR    12,15
DROP  15
USING ITPDSDAT,12 R12 IS NOW BASE
L     R3,DYNSIZE * LENGTH TO GET
STORAGE OBTAIN,ADDR=(1),SP=Ø,LENGTH=(3)
LR    R13,1 GET ADDRESS OF AREA
USING DYNAREA,13 USING FOR THE DYNAMIC AREA
SR    R5,R5
ICM   R5,B'1000',=X'40' MAKE THE MASK, A BLANK
LR    R2,R1 MOVE ADDRESS TO R2
MVCL  R2,R4 CLEAR GETMAIN' D TO BLANKS
* * * *-----------------------------------------------------------*-*-*
* * A LITTLE UP FRONT HOUSEKEEPING - XFER LITERALS ETC.
* *
* * * *-----------------------------------------------------------*-*-*
* * * *-----------------------------------------------------------*-*-*
* * * *-----------------------------------------------------------*-*-*
* * NOW FILL IN THE REQUESTED INFORMATION

* * * *-----------------------------------------------------------*-*-*
* * * *-----------------------------------------------------------*-*-*
* * * *-----------------------------------------------------------*-*-*
* * REG 1 --> ADDR 1 --> EXIT DATA
* *
* * +4 ADDR 2 --> PANEL NAME
* *
* * +8 ADDR 3 --> PANEL SECTION
* *
* * +12 ADDR 4 --> MESSAGE ID
* *
* * +16 ADDR 5 --> NUMBER OF VARIABLES
* *
* * +20 ADDR 6 --> ARRAY OF VARIABLE NAMES
* *
* *
* *
* *
+24 | ADDR 7 | -> ARRAY OF VARIABLE LENGTHS *
+28 | ADDR 8 | -> STRING OF VARIABLE VALUES *

LM R4, R7, 16(R8)          R4 = -> F(NUM VARIABLES)
                   R5 = -> C'VARNM1', C'VARNM2' ETC
                   R6 = -> A(LEN1), A(LEN2) ETC.
                   R7 = -> C'VAR1', C'VARIABLE' ETC

CLC Ø(6, R7), =C'SHOWME' MAKE SURE THEY WANT REFORMATTING
BNE RETURNØØ
L  R3, =F'32767'           * LEN TO GET FOR AN INPUT BUFFER
STORAGE OBTAIN, ADDR=(1), SP=Ø, LENGTH=(3)
ST  R1, BUFF1
MVC Ø(6, R7), =C' '      'RESET THE COMMAND THAT THEY GAVE US
L  R3, Ø(R6)           GET LENGTH OF FIRST VARIABLE
AR  R7, R3              POINT TO SECOND VARIABLE (DSN)
L  R3, 4(R6)           GET LEN OF DSN NEXT
BCTR R3, Ø               REDUCE LENGTH FOR EX TO COME
#BLANK DSNMAKE SURE IT IS BLANK TO START WITH
EX  R3, MOVEDSN              WE CAN USE IT LATER
LA  R9, DSNNAME
LA  R3, 1(R3)           BUMP IT BACK TO ORIGINAL LENGTH
AR  R9, R3              R9 = END OF DSN
AR  R7, R3              ADD TO START OF VARIABLE AREA
L  R6, 8(R6)           GET LENGTH OF VDATA
* NOW R6 = LENGTH OF VDATA, AND R7 POINTS TO VDATA
ST  R6, VDLEN                SAVE LENGTH OF DATA
ST  R7, VDADDR               SAVE ADDRESS OF VARIABLE DATA
LA  R2, SAVEALL              GET ADDRESS TO SAVE VARIABLE IN
LR  R3, R6                   GET LENGTH TO MOVE TO
LR  R5, R3                   MAKE FROM AND TO LENGTHS EQUAL
LR  R4, R7                   GET ADDRESS TO MOVE FROM
MVCL R2, R4
ZAP KONTFND, =PL1'Ø'         SET NO ENTRIES FOUND YET

DYNAL    DS    ØH
LA  R10, SVC99PRM         PTR TO SVC 99 RB PTR
LA  R11, SVC99PRM+4       PTR TO SVC 99 RB
USING  S99RBP, R10         RB PTR
USING  S99RBP, R11         RB
ST  R11, S99RBPTR        SET RB PTR TO RB
OI  S99RBPTR, S99RBPND     PTR HIGH ORDER BIT ON

* BUILD THE TEXT UNITS TU1 - TU4
* TU1 DSN=A.B.C
  MVC TU1(2),=AL2(DALDSNAM) KEY = DSN
  MVC TU1+2(2),=X'0001' NUMBER OF FIELDS
  MVC TU1+4(2),=X'002C' LENGTH OF FIELD
  MVI TU1+6(44),DSNAME DATASET NAME WAS ALREADY MOVED

* TU2 RETURN THE DDNAME
  MVC TU2(2),=AL2(DALRTDDN) KEY = RETURN DDNAME
  MVC TU2+2(2),=X'0001' NUMBER OF FIELDS
  MVC TU2+4(2),=X'0008' LENGTH OF FIELD
  MVC TU2+6(8),=CL8' ' PRE-CLEAR THE DDNAME

* INITIALIZE THE TEXT UNIT POINTER LIST

  LA R1,TU1 get addr of text unit 1
  ST R1,TU1P and save in list
  LA R1,TU2 get addr of text unit 2
  ST R1,TU2P and save in list
  LA R1,TU3 get addr of text unit 3 (static)
  ST R1,TU3P and save last in list
  LA R1,TU4 get addr of text unit 4 (static)
  ST R1,TU4P and save last in list

* INITIALIZE THE SVC 99 REQUEST BLOCK

  XC S99RB(RBLLENGTH),S99RB ZERO THE RB
  MVI S99RBLN,RBLLENGTH RB LENGTH
  MVI S99VERB,S99VRBAL RB VERB CODE=ALLOC
  LA R1,S99TPP ADR SVC 99 TEXT PTRS
  ST R1,S99TXTPP STORED IN RB
  OI TU4P,S99TUPLN HIGH ORDER BIT ON
  LR R1,R1Ø ADR OF RB POINTER

* allocate our pds now

  DYNALLOC INVOKES SVC99
  LTR R15,R15
  BZ ALLOCOK
  WTO 'DYNAMIC ALLOCATION ERROR - SHOWME ABORTING'

ALLO CK EQU *
  MVC DATAIN+4O(8),DDNAME
  OPEN (DATAIN,INPUT) FINALLY OPEN THE DATASET
  LA R2,DATAIN
  USING IHADCB,R2
  ICM R1,B'0001',DCBRECFM GET THE RECORD FORMAT
  STC R1,RECFM AND SAVE IT AWAY FOR LATER

LOOPTOP EQU *

#BLANK MEMNAM BLANK THE MEMBER NAME
#ZERO MEMTTRZL ZERO THE TTR LOW FIELD (TOP OF MEM)
#ZERO MEMTTRZ RH
ZERO THE TTR HIGH FLD ( END OF MEM)

MVC MEMNUM, =X'0001'  NUMBER OF MEMBERS TO GET

MVC MEMLEN, =X'0010'  UP TO THE FIRST 16 BYTES

MVC MEMNAM, 3(R7)  MOVE THE MEMBER NAME

ZAP KONT, =PL1'0'

BLDL DATAIN, MEMTEST  DO THE BLDL

LTR 15,15

BNZ ENDMEM  IF NO GOOD BLDL - DON'T DO IT

POINT DATAIN, MEMTTRZL  REPOSITION THE PDS TO THE MEMBER IN?

CLRLINE MVC 24(R7), C''

MVC 25(54,R7), 24(R7)  PRE BLANK THE LINE

READMO L R4, BUFF1

READ DECB, SF, DATAIN, (R4), 'S'

CHECK DECB  WAIT FOR EVENT COMPLETION

LA R2, DATAIN

USING IHADCB, R2

TM RECFM, DCBRECV  RECFM=VB

BNO NOTVB1

LH R1, Ø(R4)  PICK UP THE BDW

S R1, =F'4'  REDUCE SIZE REMAINING BY RDW LENGTH

STH R1, BLKSIZE  AND SAVE AS ACTUAL SIZE

LA R4, 4(R4)  BUMP PAST BDW

LH R1, Ø(R4)  PICK UP THE RDW

S R1, =F'4'  REDUCE SIZE REMAINING BY BDW LENGTH

STH R1, LRECL

LA R4, 4(R4)  BUMP PAST THE RDW

B VB1

NOTVB1 LH R1, DCBBLKS I

STH R1, BLKSIZE  GET BLOCK SIZE

LH R1, DCBLRECL

STH R1, LRECL  SAVE LRECL

DROP R2

L R2, DECB+16  GET STATUS AREA ADDRESS

LH R2, 14(R2)  GET RESIDUAL COUNT

LH R1, BLKSIZE  GET REQUESTED BLOCK SIZE

SR R1, R2  R1 = ACTUAL BYTE COUNT

STH R1, BLKSIZE  SAVE ACTUAL BYTE COUNT

* * * * *-----------------------------------------------------------*-*-*

* *- NOW LOOK FOR THE LITERAL SO WE CAN UPDATE THE DYNAMIC AREA -*

* * * * *-----------------------------------------------------------*-*-*

VB1 EQU *

NEXTREC AP KONT, =PL1'1'

COUNT THE RECORDS CHECKED

CP KONT, =PL2'15'

LOOK UP TO 15 RECORDS DEEP

BH ENDMEM

THEN STOP CHECKING THIS MEMBER

LR R5, R4

GET POINTER TO RECORD

* HERE IS THE ACTUAL CHECK *

CK2 CLI 3(R5), C' %'

CHECK FOR THE LITERAL

BE FBRECS

CK3 CLI 11(R5), C' %'

CHECK FOR LITERAL IN STD NUM D/S
BNE NEXTCRD
LA R5, 8(R5)  ADJUST POINTER FOR STD NUMS IN CLIST

FBRECS CLCØ(14,R5), =C'\* %PDSDOC ØØ ' CHECK FOR ASM TYPE CARDS
BE GOTIT1
CLCØ(14,R5), =C'/ %PDSDOC ØØ ' CHECK FOR JCL TYPE CARDS
BE GOTIT1
CLCØ(14,R5), =C'/ %PDSDOC ØØ ' CHECK FOR CLIST TYPE CARDS
BNE NEXTCRD

GOTIT1 EQU *
MVC24(55, R7), 14(R5) MOVE THE DESCRIPTION IN
AP KONT, =PL2 '90'
AP KONTFND, =PL1 '1' TRACK NUMBER OF ENTRIES FOUND
B ENDMEM

NEXTCRD LH R2, BLKSIZE
LA R3, LRECL
SR R2, R3 REDUCE BY RECORD JUST PROCESSED
BZ READMO
BNE NEXTREC

BP READMO IF DONE GET ANOTHER
S R3, =F'4' REDUCE LENGTH BY RDW LENGTH
STH R3, LRECL
LA R4, 4(R4) AND BUMP PAST THE RDW
B NEXTREC AND THEN JUST GO DO IT.

IOERR1 EQU * IF ERROR READING - GET OUT NOW...
ENDMEM EQU * END OF THE MEMBER OR DESC. IS DONE
S R6, =F'80'
BZ READDONE IF ZERO THEN DONE
BNE REALDONE IF LESS THEN REALLY DONE
LA R7, 80(R7) ELSE BUMP TO NEXT VALID LINE
B LOOPTOP

REALDONE EQU *
CP KONTFND, =PL1 'Ø'
B REALDONE

* IF WE DIDN'T FIND ANY DESCRIPTIONS ON THIS SCREEN - RESET IT
LM R2, R3, VDADDR GET TO ADDRESS AND LENGTH
LA R4, SAVEALL GET FROM ADDRESS
LR R5, R3 MAKE FROM AND TO LENS MATCH
MVCL R2, R4 RESTORE VARIABLE DATA

* now free our dataset input buffer and close and free the dataset
NORESET L R1, BUFF1

L R2,=F'32767'

STORAGE RELEASE, ADDR=(1), LENGTH=(2)
CLOSE DATAIN

RETURNØØ EQU *
L R2, DYN SIZE               * LENGTH TO GET
LR R8, R13
STORAGE RELEASE, ADDR=(8), LENGTH=(2)
XR R15, R15
PR

MOVEDSN MVC DSNAME(Ø), Ø(R7)
* *-*-*-----------------------------------------------------------*-*-*
* *- STATIC STORAGE AREA HERE - LTORG - MODELS ETC.                  -*
* *-*-*-----------------------------------------------------------*-*-*
DYNSIZE DC AL4(@DYNSIZE)           DYNAM AREA SIZE
DATAIN DCB MACRF=R, DDNAME=SYSUT1, DSORG=PO, EODAD=ENDMEM, X
SYNAD=IOERR1
TU3 DC AL2(DALCLOSE)           FREE=CLOSE '  
DC XL2'0000'               ZERO FIELDS PASSED/RETURNED
DC XL2'0000'               ZERO LENGTH
TU4 DC AL2(DALSTATS)           DISP=SHR
DC XL2'0001'               ONE FIELD PASSED / RETURNED
DC XL2'0001'               LENGTH OF FIELD
DC XL1'08'                 SPECIFICATION IS SHR (8)
LTORG
*

DYNAREA DSECT
SAVEAREA DS 18F             SAVEAREA FOR CALLED ROUTINES
BUFF1 DS F                 ADDRESS OF 32K INPUT BUFFER AREA
DBLWORK DS D               *
* next two lines must stay together *
VDADDR DS F                 SAVE AREA FOR VARIABLE
VDLEN DS F                 SAVE AREA FOR LENGTH OF VARIABLE
MEMBER DS CL8              SAVE ROOM FOR A MEMBER NAME
* BELOW IS AN AREA FOR A GOOD BLDL **
MEMTEST EQU *
MEMNUM DS XL2               NUMBER OF ENTRIES TO TEST
MEMLEN DS XL2               LENGTH OF FIELD TO FILL
MEMNAM DS CL8              NAME OF MEMBER TO BLDL FOR
MEMTTRZL DS XL4            TTR FOR THE MEMBER ( LOW TTR )
MEMTTRZH DS XL4            TTR FOR THE MEMBER ( HIGH TTR )
BLKSIZE DS H               *
LRECL DS H
KONT DS PL2
KONTFND DS PL2
RECFM DS XL1               THE RECORD FORMAT FROM THE DCB
DS OF
SVC99PRM DS CL(RBLENGTH+4)   RB PTR & RB STORAGE
It would be a simple matter to extend the processing to handle member selection lists from 3.4 member display lists that are formatted just a bit differently. The decision regarding which format to use could be based on the panel name that is passed in the standard parameter list, which is passed to the panel exit.

I found that this was both interesting to write and improves my productivity on a daily basis. I hope you can take advantage of the code in your shop as well.

Stephen G McColley
Senior Systems Programmer
SunTrust Bank (USA)  © Xephon 2003
Descising key support in IMS made easy

INTRODUCTION

IMS full-function databases provide a broad array of functions that are used to support some of the most demanding business applications in the world. One of the fundamental characteristics of these hierarchical databases is their ability to logically store and retrieve keyed segments in an ascending collating sequence. Unfortunately, these databases do not readily offer the reciprocal capability of processing keyed segments in a logically descending collating sequence. This document describes an application-transparent method of providing logically descending segment keys.

THE HARD WAY

Application designers have employed a variety of techniques to simulate a logically descending segment key function for IMS databases. Some applications have resorted to sorting a memory array of segments into a descending sequence. Other applications have incurred the expense of secondary indexing as a means of processing segments in an alternate order. Still other applications have undertaken the burden of translating the value of the sequence field in order to achieve a logically descending effect. The common problem with all of these approaches is their dependence upon the application programs to interpret and maintain an artificial segment sequence field.

THE EASY WAY

With the introduction of the Data Conversion User Exit Routine (DFSDBUX1), IMS now provides a General-Use Programming Interface from which to implement logically descending key functionality. The exit routine is invoked at the beginning and at the end of the DL/I Call Analysis routine (DFSDLA00). These are...
the ideal times for manipulating the segment sequence values in order to create the logically descending key effect. The Segment Search Argument (SSA) and Key Feedback (KFB) areas are also available to the exit routine for similar processing. Therefore, the exit routine can interpret and consistently maintain the segment sequence field while insulating the application programs from the underlying details.

EASY AS 1-2-3

The logically descending key function can be implemented with the sample exit routine by completing the following simple steps:

1. Within the sample exit routine, specify all the database (physical, logical, and relevant secondary index) segments that need the descending key function. The exit routine works as an extension to the application’s DL/I call; consequently, it operates according to the associated PCB’s segment image and sensitivity definitions. Assemble and link the new exit routine into an appropriate load library.

2. Add the DATXEXIT keyword to the specified (physical) Database Descriptions (DBD) and generate the new database control blocks.

3. Reorganize the source databases and convert the segment keys using the new exit routine. These tasks require the unloading of the source database using the old DBD (expanding affected segments that have compressed keys). The source database is then reloaded with the new DBD (compressing affected segments that have compressed keys) while invoking the new exit routine. This process is followed by the resolution of any logically-related databases and the rebuilding of secondary indices. Finally, the new databases can be brought on-line when the new exit routine and control blocks are cycled with the IMS subsystem.

CONCLUSION

Logically descending segment key functionality can be easily
supported through the Data Conversion User Exit Routine. This centralized approach is a more reliable and cost-effective means of maintaining a logically descending key sequence. It frees application programmers from having to devise and maintain different programmatic solutions. Logically descending key support enhances the functionality of IMS and thereby increases its value as a useful platform for deploying business applications.

**DFSDBUX1**

**MACRO**

```
&LABEL $DFSDBUX &FUNC, &NAME=, &BYTES=, &START=, &EXIT=
**********************************************************************
* FUNCTION:                                                         *
* THE $DFSDBUX MACRO PROVIDES A MEANS FOR CONSTRUCTING THE          *
* IMS SEGMENT REGISTRATION TABLE. THIS TABLE IS DEFINED IN MODULE    *
* $DFSDBUX, WHICH CONSISTS OF A SINGLE CSECT CONTAINING THE          *
* SEGMENT REGISTRATION TABLE AT OFFSET 0.                           *
* $DFSDBUX BEGIN AND $DFSDBUX END DELIMIT THE TABLE DEFINITION.    *
* BETWEEN THESE DELIMITERS ANY NUMBER OF $DFSDBUX DBD AND SEGM      *
* DECLARATIONS ARE USED TO DEFINE SPECIAL SEGMENT DATA CONVERSION    *
* TREATMENT. ONLY A SINGLE SEGMENT REGISTRATION TABLE (IE          *
* $DFSDBUX BEGIN/END PAIR) MAY BE DEFINED; ATTEMPTING TO DEFINE      *
* A SECOND TABLE WILL CAUSE AN ASSEMBLY ERROR.                     *
* THE MACRO HAS THE FOLLOWING PARAMETER SUPPORT:                   *
* BEGIN                                                          *
* THE BEGIN POSITIONAL PARAMETER SPECIFIES THE                    *
* BEGINNING OF THE SEGMENT REGISTRATION TABLE, AND IS             *
* REQUIRED PRIOR TO ANY OTHER $DFSDBUX INVOCATIONS.               *
* WHEN THIS PARAMETER IS SPECIFIED ANY OTHER                       *
* PARAMETERS ARE IGNORED.                                         *
* END                                                            *
* THE END POSITIONAL PARAMETER SPECIFIES THE                      *
* TERMINATION OF THE SEGMENT REGISTRATION TABLE, AND              *
* MUST BE THE LAST $DFSDBUX INVOCATION.                           *
* WHEN THIS PARAMETER IS SPECIFIED ANY OTHER                       *
* PARAMETERS ARE IGNORED.                                         *
* DBD                                                            *
* THE DBD POSITIONAL PARAMETER IDENTIFIES THE                     *
* DATABASE GROUP FOR SUBSEQUENT SEGMENT DECLARATIONS.             *
* SEGM                                                           *
* THE SEGM POSITIONAL PARAMETER IDENTIFIES THE                    *
* SEGMENT DECLARATION WITHIN THE CURRENT DATABASE GROUP.          *
* NAME= DATABASE OR SEGMENT NAME                                  *
* REQUIRED                                                         *
* THE NAME= PARAMETER SPECIFIES THE 1 TO 8                        *
```

© 2003, Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
ALPHANUMERIC CHARACTERS FOR THE ASSOCIATED DATABASE OR SEGMENT.

BYTES = 1-255
  OPTIONAL
  THE BYTES= PARAMETER SPECIFIES THE LENGTH TO BE USED WITH THE SEGMENT'S SEQUENCE FIELD.
  THIS PARAMETER IS USED ONLY WHEN A PORTION OF THE SEQUENCE FILED IS TO BE CONVERTED.

START = (1-32767, 1-3825, 1-255)
  OPTIONAL
  THIS PARAMETER IS USED IN CONJUNCTION WITH THE
  THE BYTES= PARAMETER ONLY WHEN A DISTAL PORTION OF THE SEQUENCE FIELD IS TO BE CONVERTED.

EXIT = EXIT NAME
  OPTIONAL
  THE EXIT= PARAMETER SPECIFIES THE 1 TO 8 ALPHANUMERIC CHARACTERS FOR THE SPECIFIED EXIT ROUTINE.

RESTRICTIONS: NONE

MESSAGES:

RC MESSAGE TEXT

4 BEGIN ISSUED ON AN OPEN TABLE DEFINITION, IGNORED.
4 END ISSUED OUTSIDE A TABLE DEFINITION. IGNORED.
8 INVALID POSITIONAL PARAMETER &FUNC
8 $DFSDBUX TABLE NOT ACTIVE. $DFSDBUX BEGIN NEEDED.
8 DBD NAME= OPERAND IS OMITTED OR INVALID
8 SEGМ NAME= OPERAND IS OMITTED OR INVALID
8 BYTES= OPERAND IS MISSING OR INVALID
8 START= SUBPARAMETER (SEG,,) IS MISSING OR INVALID
8 START = SUBPARAMETER (,,KFB,) IS MISSING OR INVALID
8 START = SUBPARAMETER (,,ARG) IS MISSING OR INVALID
8 EXIT = OPERAND IS INVALID

EXAMPLE:

THE FOLLOWING IS AN EXAMPLE OF THE USE OF $DFSDBUX TO CONSTRUCT A SEGMENT REGISTRATION TABLE INSTRUCTING IMS TO CONVERT THE SEQUENCE (KEY) FIELDS OF THE SPECIFIED SEGMENTS WITHIN THEIR RESPECTIVE DATABASES. THIS EXAMPLE IS FOR ILLUSTRATION PURPOSES ONLY.

$DFSDBUX BEGIN
  $DFSDBUX DBD,NAME=DBD00000
  $DFSDBUX SEGM,NAME=SEG00000
  $DFSDBUX DBD,NAME=DBD99999
  $DFSDBUX SEGM,NAME=SEG00000,BYTES=1
  $DFSDBUX SEGM,NAME=SEG99999,BYTES=2,START=(4,12,2)
$DFSDBUX END
* VARIABLE DECLARATIONS *
*--------------------------------------------------------------------*
GBLA &STATE
GBLA &TBLNAME
GBLC &DBDIX, &DBDSEGX
GBLC &DBDNAME(256), &DBDEXIT(256)
GBLA &DBDSEG8(256), &DBDSEG8E(256)
GBLC &SEGNAME(2048)
GBLA &FLDLNG(2048)
GBLC &FLDSEG0(2048), &FLDFBO(2048), &FLDARGO(2048)
LCLA &DBDCNT, &DBDNext
LCLC &DBDLBL
LCLA &SEGCA, &SEGNEXT
LCLC &SEGCLBL
LCLA &SEQLNG, &SEQSEGO, &SEQKFBO, &SEQARGO
LCLB &SEQFLLG0, &SEQFLLG1, &SEQFLLG2, &SEQFLLG3
*--------------------------------------------------------------------*
* POSITIONAL PARAMETERS *
*--------------------------------------------------------------------*
AIF ('&FUNC' EQ 'BEGIN').BEGIN
AIF ('&FUNC' EQ 'DBD').DBD
AIF ('&FUNC' EQ 'SEGM').SEG
AIF ('&FUNC' EQ 'END').END
AIF ('&FUNC' EQ 'DSECT').DSECT
MNOTE 8,'INVALID POSITIONAL PARAMETER &FUNC'
AGO .EXIT
*--------------------------------------------------------------------*
* BEGIN *
*--------------------------------------------------------------------*
BEGIN ANOP
AIF (&STATE NE 1).BEGINX DETECT UNEXPECTED 'BEGIN'
MNOTE 4,'BEGIN ISSUED ON AN OPEN TABLE DEFINITION, IGNORED'
AGO .EXIT
BEGINX ANOP
&STATE SETA 1 SHOW IN DEFINITION
&TBLNAME SETC '&LABLE'
AGO .EXIT
*--------------------------------------------------------------------*
* END *
*--------------------------------------------------------------------*
END ANOP
ENDX ANOP
&STATE SETA 0 SHOW NOT IN DEFINITION
AIF ('&TBLNAME' NE '').ENDNAME
&TBLNAME SETC '$DFSDBUX'
.ENDNAME ANOP
&TBLNAME CSECT,

© 2003. Xephon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
&DBDNCNT SETA 1
&DBDNEXT SETA &DBDNCNT+1
.*
  ENDDBD ANOP
  AIF (&DBDNCNT GT &DBDINDX).EXIT
&DBDLBL SETC 'DBD#'.&DBD_NEXT' AIF (&DBD_NEXT LE &DBDINDX).DBDLBL
&DBDLBL SETC 'Ø'
  DBDLBL ANOP
  D#&DBDNCNT DC A(&DBDLBL),CL8'&DBDNAME(&DBDNCNT)'
  AIF ('&DBDEXIT(&DBDNCNT) EQ 'Ø').ADCON
  DC V(&DBDEXIT(&DBDNCNT))
  AGO .VCON
  ADCON ANOP
  DC A(&DBDEXIT(&DBDNCNT))
  .VCON ANOP
  *
&SEGNCNT SETA &DBDSEGB(&DBDNCNT)
  NEXTSEG ANOP
  AIF (&SEGNCNT LE Ø OR &SEGNCNT GT &DBDSEGE(&DBDNCNT)).NEXTDBD
&SEGNEXT SETA &SEGNCNT+1
&SEGLBL SETC 'SEG#'.&SEGNCNT' AIF (&SEGNCNT LT &DBDSEGE(&DBDNCNT)).SEGLBL
&SEGLBL SETC 'Ø'
  SEGLBL ANOP
  *
  SEG#&SEGNCNT DC A(&SEGLBL),CL8'&SEGNAME(&SEGNCNT)'
  *
&SEQFLO SETB Ø
&SEQFLG SETB Ø
&SEQFLG SETB Ø
&SEQFLG SETB Ø
&SEQLNG SETA &FLDLNG(&SEGNCNT)
&SEQSEGO SETA &FLDSEGO(&SEGNCNT)
&SEQKFB SETA &FLDKFB(&SEGNCNT)
&SEQARGO SETA &FDARGO(&SEGNCNT)
  *
  AIF (&SEQLNG LE Ø).SEQLNG
  &SEQLNG SETA &SEQLNG-1 EXECUTABLE LENGTH
  &SEQFLG SETB 1
  .SEQLNG ANOP
  AIF (&SEQSEGO LE Ø).SEQSEGO
  &SEQSEGO SETA &SEQSEGO-1 DISPLACEMENT OFFSET
  &SEQFLG SETB 1
  .SEQSEGO ANOP
  AIF (&SEQKFB LE Ø).SEQKFB
  &SEQKFB SETA &SEQKFB-1 DISPLACEMENT OFFSET
  &SEQFLG SETB 1

.SEQKFB0 ANOP
  AIF (&SEQARG0 LE 0). SEQARG0
&SEQARG0 SETA &SEQARG0-1 DISPLACEMENT OFFSET
&SEQFLG3 SETB 1
.SEQARG0 ANOP
  DC B'&SEQFLG0&SEQFLG1&SEQFLG2&SEQFLG3.0000', AL1(&SEQLNG)
  DC Y('&SEQSEG0, &SEQKFB0, &SEQARG0)

,*
&SEGCNT SETA &SEGCNT+1
AGO .NEXTSSEG
.NEXTDBD ANOP
*
&DBDCNT SETA &DBDCNT+1
&DBDNEXT SETA &DBDCNT+1
AGO .ENDDBD

*--------------------------------------------------------------------*
*      DBD                                                           *
*--------------------------------------------------------------------*
DBD ANOP
  AIF (&STATE EQ 1). DBDNAME DETECT MISSING 'BEGIN'
  MNOTE 8, '$DFSDBUX TABLE NOT ACTIVE. $DFSDBUX BEGIN NEEDED.'
  AGO .EXIT

DBDNAME ANOP
  AIF ('&NAME' EQ '' OR K'&NAME GT 8). DBDERR
&DBDINDX SETA &DBDINDX+1
&DBDNAME(&DBDINDX) SETC '&NAME'
AGO .EXITNAM

DBDERR MNOTE 8, 'DBD NAME= OPERAND IS OMITTED OR INVALID'
AGO .EXIT

*--------------------------------------------------------------------*
*      EXIT KEYWORD                                                  *
*--------------------------------------------------------------------*
EXITNAM ANOP
  &DBDEXIT(&DBDINDX) SETC '0'
  AIF ('&EXIT' EQ ''). EXIT
  AIF (K'&EXIT GT 8). EXITERR
&DBDEXIT(&DBDINDX) SETC '&EXIT'
AGO .EXIT

EXITERR MNOTE 8, 'EXIT= OPERAND IS INVALID'
AGO .EXIT

*--------------------------------------------------------------------*
*      SEG                                                           *
*--------------------------------------------------------------------*
SEG ANOP
  AIF (&STATE EQ 1). SEGNAME DETECT MISSING 'BEGIN'
  MNOTE 8, '$DFSDBUX TABLE NOT ACTIVE. $DFSDBUX BEGIN NEEDED.'
  AGO .EXIT

SEGNAME ANOP
  AIF ('&NAME' EQ '' OR K'&NAME GT 8). SEGERR
&DBDSEGX SETA &DBDSEGX+1
ORG UX$EXIT
*
UX$FLAG DS XL1                  PROCESSING OPTIONS
UX$FLGØ EQU X'80'               ALTERNATE FIELD LENGTH
UX$FLG1 EQU X'40'               ALTERNATE SEGMENT OFFSET
UX$FLG2 EQU X'20'               ALTERNATE KEY FEEDBACK OFFSET
UX$FLG3 EQU X'10'               ALTERNATE ARGUMENT OFFSET
UX$FLDL DS XL1                  FIELD EXECUTABLE LENGTH
UX$SEGO DS H                    FIELD OFFSET INTO SEGMENT
UX$KFBØ DS H                    FIELD OFFSET INTO KEY FEEDBACK
UX$ARGØ DS H                    FIELD OFFSET INTO SSA ARGUMENT
AGO   .EXIT
.*--------------------------------------------------------------------*
.*      COMMON EXIT                                                   *
.*--------------------------------------------------------------------*
.EXIT    ANOP
MEND
DFSDBUX1 TITLE 'DFSDBUX1 - DATA CONVERSION EXIT'
-------------------------------------------------------------------- *
* MODULE NAME : DFSDBUX1                                               *
* ENTRY POINT : DFSDBUX1                                              *
* FUNCTION : LOGICALLY DESCENDING KEYS                                *
* MODULE ATTRIBUTES: REENTRANT                                        *
* REGISTERS AT ENTRY                                                 *
*   RØ      STATUS CODE                                              *
*    IN - START OF DL/I CALL                                          *
*    OUT - END OF DL/I CALL                                          *
*   R1     PST ADDRESS                                               *
*   R3     PCB ADDRESS                                               *
*   R5     PDIR ADDRESS                                              *
*   R6     SCD ADDRESS                                               *
*   R7     DMBXBLCK ADDRESS                                          *
*   R9     JCB ADDRESS                                               *
*   R10    SDB ADDRESS                                               *
*   R13    SAVE AREA                                                 *
*   R14    RETURN ADDRESS                                            *
*   R15    ROUTINE ENTRY POINT ADDRESS                               *
* REGISTERS AT EXIT  :      RESTORED                                 *
* REGISTER USAGE :   R12 EXECUTION BASE                               *
*                     R11 REGISTRATION BASE                            *
*                     R8 LEVEL TABLE BASE                              *
*                     R7 PHYSICAL SEGMENT DESCRIPTOR BASE              *
*                     R5 SEGMENT FIELD DESCRIPTION BASE               *
*                     R4 SSA FIELD DESCRIPTION BASE                   *
***********************************************************************
GBLA &DFARELN
IMSRELSE
EJECT
DFSDBUX1 CSECT
AIF ('&DFARELN' GT '6').UX1V61
COPY ASMMSP                  CONCEPT 14 DEFINITIONS

*=====================================================================*
*        PROCESS REGISTERED DATABASES                                 |
*=====================================================================*

    L     R11, JCBWKR55            GET A(DATABASE ENTRY)
    I F    (LTR, R11, R11, Z)         GOT A(DATABASE ENTRY) ?
        L     R11, =V($DFSDBUX)  GET A(REGISTRATION TABLE)
        USING UX$, R11
        STRTSRCH UNTIL = (ICM, R11, 15, UX$NEXT, Z)
        EXITIF   (CLC, DBPCBDBD, EQ, UX$NAME)
        ST    R11, JCBWKR55    A(DATABASE ENTRY)
ENDLOOP  ,

ENDSRCH  ,
B     EXIT        DO NOT COME BACK
ENDIF  ,

*=====================================================================*
*        PROCESS REGISTERED SEGMENTS                                  |
*=====================================================================*

    LA    R15, Ø                   CLEAR
    L     R8, JCBLEVTB             GET A(LEVEL TABLE)
    USING LEV, R8
    DO    WHILE=(LTR, R8, R8, NZ)
        MVI   SAVER8, Ø          SET B(FLAG)
        L     R11, JCBWKR55    GET A(DATABASE ENTRY)
        LA    R11, UX$SEG M     SET A(SEGMENT ENTRY)
        L     R10, LEVSDB      GET A(SEGMENT DESCRIPTION)
        I F    (CL, R8, EQ, =CL4'IN'), ANDIF,      X
            (TM, LEVF1, LEVDATA, Z)
L R10, LEVNU$DB  A(SEGMENT DESCRIPTION)
ENDIF,
GOT SEGMENT DATA ?
IF (LTR, R10, R10, NZ) GOT A(SEGMENT DESCRIPTION) ?
L R7, SDBPSDB  GET A(PHYSICAL DESCRIPTION)
USING DMBPSDB, R7
L R5, DMBFDBA  GET A(FIELD DESCRIPTION)
USING FDB, R5
IF (TM, FDBDCENF, FDBKEY, O)
* +----------------------------------------------------------+
* | PROCESS REGISTERED SEGMENTS   |    +----------+
* +----------------------------------------------------------+
DO UNTIL =(ICM, R11, 15, UX$NEXT, Z)
IF (CLC, SDBSYM, EQ, UX$NAME)
* +---------------------------------
| PROCESS REGISTERED SEGMENTS      |
* +---------------------------------
IF (ICM, R4, 15, LEVFLD, NZ)
USING FLD, R4
ST R11, SAVER11 SET A(SEGMENT ENTRY)
* +-----------------------------+
* | CONVERT REGISTERED FIELDS    |
* +-----------------------------+
DO UNTIL =(TM, FLDMBR, FLDMEMRP, O)
LA R4, FLDE$ENG(, R4)
L R11, SAVER11 GET A(SEGMENT ENTRY)
* +---------------------------------
| GET CALL PROCESSING           |
* +---------------------------------
IF (CLI, PSTFUNCH, EQ, C'G')
NI SAVER8, 255 - 128
DO UNTIL=(LTR, R11, R14, Z)
IF (CLC, SDBSYM, EQ, UX$NAME)
* +----------------------------------+
| QUALIFIED CALL PROCESSING        |
* +----------------------------------+
LH R6, FDBOFFST GET S(OFFSET)
ST R6, SAVER6 SET S(OFFSET)
IC R15, FDBFLENG GET F(FIELD LENGTH)
LA R2, I(R15, R6) GET S(OFFSET)
ST R2, SAVER2 SET S(OFFSET)
* +----------------------------------+
IC R15, FDBFLENG GET F(FIELD LENGTH)
L R14, LEVSSA GET A(SSA)
AH R14, FLDSSAOF ADD S(OFFSET)
LH R6, FLDSEGOF GET S(OFFSET)
LA R2, I(R15, R6) SET S(OFFSET)
IF (TM, UX$FLAG, UX$FLGD, O)
IC R15, UX$FLDL GET F(FIELD LENGTH)
AH R14, UX$ARGO ADD S(OFFSET)
IF (TM, UX$FLAG, UX$FLG1, O)
ST R2, SAVER2
LH R6, UX$SEGG0
IF \( (R15, R6, GE, FLDSEGOF) \), AND IF, \( X \)   
EX \( R15, ONESCOMP \)  
OI \( SAVER8, 128 \)  
ENDIF, WITHIN BOUNDS?  
B NEXTARG  
ENDIF, ALTERNATE SEGMENT OFFSET?  
ENDIF, GOT ALTERNATE FIELD LENGTH?  
*  
IF \( (R15, R6, GE, SAVER6) \), AND IF, \( X \)   
EX \( R15, ONESCOMP \)  
OI \( SAVER8, 128 \)  
ENDIF, WITHIN BOUNDS?  
*  
ENDIF, GOT REGISTERED SEGMENT?  
NEXTARG   
DOEXIT (ICM, R14, 15, UX$NEXT, Z)  
ENDDO, GOT A(SEGMENT REGISTRATION) ?  
*  
IF \( (TM, FLDMBR, FLDMEMLT+FLDMEMGT, M) \), AND IF, \( (TM, SAVER8, 128, O) \)   
XI \( FLDMBR, FLDMEMLT+FLDMEMGT \)  
OI \( SAVER8, 64 \)  
ENDIF, GOT RELATIONAL OPERATORS  
*  
UPDATE CALL PROCESSING  
*  
ELSE, NOT A GET CALL  
IC \( R15, FDBFLENG \) GET F(FIELD LENGTH)  
L \( R14, PSTIPARM \) GET A(PARM)  
L \( R14, 8(, R14) \) GET A(I/O)  
L \( R2, LEVSSA \) GET S(OFFSET)  
LA \( R14, 0(R2, R14) \) SET A(SEGMENT)  
LH \( R2, FDBOFFST \) GET S(OFFSET)  
IF \( (TM, UX$FLAG, UX$FLG1, O) \)  
LH \( R2, UX$SEGO \) S(OFFSET)  
ENDIF, GOT ALTERNATE SEGMENT OFFSET?  
LA \( R14, 0(R2, R14) \) GET A(FIELD)  
EX \( R15, ONESCOMP \)  
IF \( (CL, R0, EQ, =CL4'OUT') \)  
L \( R14, SDBKEYFD \) GET A(KFB)  
IF \( (TM, UX$FLAG, UX$FLG2, O) \)  
AH \( R14, UX$KFBO \)  
ENDIF, EX \( R15, ONESCOMP \)  
ENDIF, GOT A GET CALL?
ENDDO, GOT RIGHT PARENTHESIS?
IF (CL, RØ, EQ, =CL4'OUT'), ANDIF,
(CLI, PSTFUNCH, EQ, C'G')
L R11, SAVER11
DO UNTIL=(LTR, R11, R14, Z)
IF (CLC, SDBSYM, EQ, UX$NAME)
IC R15, FDBFLENG GET F(LENGTH)
L R14, SDBKEYFD GET A(KFB)
IF (TM, UX$FLAG, UX$FLG0, O)
LA R2, 1(R15, R14)
IC R15, UX$FLDL F(LENGTH)
AH R14, UX$KFBO S(OFFSET)
IF (CR, R14, GE, R2)
B SKIPKFB
ENDIF, WITHIN BOUNDS?
ENDIF, ALTERNATE FIELD LENGTH?
EX R15, ONESCOMP

SKIPKFB DS ØH
IF (TM, LEVF1, LEVDATA, O)
L R14, PSTIPARM A(PARM)
L R14, 8(R14) A(I/O)
AH R14, LEVUSEOF S(OFFSET)
LH R2, FDBOFFST S(OFFSET)
IF (TM, UX$FLAG, UX$FLG1, O)
LH R2, UX$SEGO
ENDIF,
LA R14, Ø(R2, R14) A(FIELD)
EX R15, ONESCOMP
ENDIF, GOT SEGMENT DATA?

ENDIF, GOT REGISTERED SEGMENT?
DOEXIT (ICM, R14, 15, UX$NEXT, Z)
ENDDO, GOT A(SEGMENT REGISTRATION)?
ENDIF, GOT A GET CALL?

*--------------------------------------------------------------------+
*        UNQUALIFIED CALL PROCESSING                                  |
*--------------------------------------------------------------------+
ELSE, GOT UNQUALIFIED CALL
IF (CL, RØ, EQ, =CL4'IN'), ANDIF,
(CLI, PSTFUNCH, EQ, C'G')
B SKIPSEG

IC R15, FDBFLENG GET F(LENGTH)
IF (TM, LEVF3, LEVDATA, O), ORIF,
(TM, LEVF3, LEVPseudo, O)
L R14, SDBKEYFD GET A(KEY FEEDBACK)
IF (TM, UX$FLAG, UX$FLG0, O)
  LA R2, 1(R15, R14) A(KFB)
  IC R15, UX$FLDL GET F(LENGTH)
  AH R14, UX$KFBO ADD S(OFFSET)
  IF (CR, R14, GE, R2)
    B SKIPKFBA
  ENDF, WIthin BOUNDS ?
ENDF, GOT ALTERNATE FIELD LENGTH ?
EX R15, ONESCOMP SET C(KEY FEEDBACK)
*
IF (TM, LEVF3, LEVPSUDO, O)
  B SKIPSEG
ENDF, GOT REAL SEGMENT ?
ENDF, GOT KEY FEEDBACK DATA ?
SKIPKFBA DS ØH
*
| CONVERT SEGMENT AREA |
*
* |
+--------------------------------+
* |
* |
* |
IF (TM, LEVF1, LEVDATA, O), ORIF, X
    (CLI, PSTFUNCH, EQ, C' I'), ORIF, X
    (CLI, PSTFUNCH, EQ, C' A')
L R14, PSTSEG GET A(SEGMENT)
IF (LTR, R14, R14, Z)
  L R14, PSTIPARM A(PARM)
  L R14, 8(, R14) A(I/O)
  IF (CLI, PSTFUNCH, EQ, C'A')
    LA R14, IOSEG-I0AREA(, R14)
  ENDF, GOT ASRT ?
  AH R14, LEVUSEOF S(OFFSET)
  ENDF, GOT A(SEGMENT) ?
LH R14, FDBOFFST GET S(OFFSET)
IF (TM, UX$FLAG, UX$FLG1, O)
  LH R2, UX$SEGO S(OFFSET)
ENDF, GOT ALTERNATE SEGMENT OFFSET ?
  LA R14, Ø(R2, R14) SET A(FIELD)
EX R15, ONESCOMP
ENDF, GOT SEGMENT DATA ?
SKIPSEG DS ØH
ENDDO, GOT QUALIFIED CALL ?
*
ENDF, GOT REGISTERED SEGMENT ?
ENDDO, GOT A(SEGMENT REGISTRATION) ?
*
IF (TM, SAVER8, 64, O) GOT RELATIONAL OPERATORS ?
  XC LEVMAX, LEVMI N SWAP
  XC LEVMI N, LEVMAX MINIMUM
  XC LEVMAX, LEVMI N MAXIMUM
ENDF, 
*
ENDF, GOT KEY DEFINITION ?
ENDIF,                 GOT A(SEGMENT DESCRIPTION) ?
*  
DOEXIT (TM, LEVF1, LEVLAST, O)  GOT LAST LEVEL ?
DOEXIT (TM, LEVF3, LEVQLAST, O)  GOT LAST QUALIFIED LEVEL ?
LA    R8, LEVLLEN(, R8)        SET A(LEVEL TABLE)
ENDDO,                       GOT A(LEVEL TABLE) ?
*
+--------------------------------+
|  EGRESS CALL                |
+--------------------------------+

L    R11, JCBWK5S             GET A(DATABASE ENTRY)
IF    (ICM, R15, 15, UX$EXIT, NZ), ANDIF,                          X
       (CL, RØ, EQ, =CL4'OUT')
       BAL    R14, CALLEXIT
ENDIF,
*
===============================================
|        COMMON EXIT PROCESSING                  |
===============================================
EXIT     DS    ØH
LEAVE RESTORE=(14, 12), RC=Ø
SPACE 3
ONESCOMP XC    Ø(Ø, R14), =256X'FF'
*
===============================================
|        CALL EXIT ROUTINE                       |
===============================================
CALLEXIT SAVE  (14, 12)
L    R14, SAVELAST            GET A(SA)
LM    RØ, R10, SAVERO-SAVEAREA(R14)
L    R13, SAVENEXT            PUSH SAVESET
*
   CALL (15)                      CALL REGISTERED EXIT
*
L    R13, SAVELAST            POP SAVESET
RETURN (14, 12),, RC=Ø
SPACE 3
*****************************************************************************
|        LITERAL POOL...                            |
*****************************************************************************
LTORG ,
*****************************************************************************
|        PATCH AREA...                             |
*****************************************************************************
PATCH    DC    32S(*)              << PATCH AREA >>
DROP ,
EJECT
*****************************************************************************
*
*****************************************************************************
*/

© 2003. Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
SDBBASE=Ø,
LEVBASE=Ø,
FLDBASE=Ø,
DMBBASE=Ø,
FDBBASE=Ø,
CALLER=IMS.

DFSURGUF,
REQUATE SAVE=YES

$DFSDBUX DSECT

***********************************************************************
*        SEGMENT REGISTRATION TABLE                                   *
***********************************************************************

$DFSDBUX BEGIN
$DFSDBUX DBD, NAME=PHYSICAL
$DFSDBUX SEGM, NAME=ROOTSEGM
$DFSDBUX SEGM, NAME=CHLDSEGM
$DFSDBUX DBD, NAME=SECINDEX
$DFSDBUX SEGM, NAME=IDXSEGM, BYTES=2, SRCH=FIELD
$DFSDBUX SEGM, NAME=IDXSEGM, BYTES=7, START=(8,8,8) /CK
$DFSDBUX SEGM, NAME=IDXSEGM, BYTES=9, START=(2Ø,255,255) SYMB
$DFSDBUX DBD, NAME=RELATED
$DFSDBUX SEGM, NAME=ROOTSEGM
$DFSDBUX SEGM, NAME=CHLDSEGM
$DFSDBUX DBD, NAME=PRIINDEX
$DFSDBUX SEGM, NAME=INDXSEGM
$DFSDBUX DBD, NAME=LOGICAL
$DFSDBUX SEGM, NAME=ROOT
$DFSDBUX SEGM, NAME=CHLD
$DFSDBUX SEGM, NAME=STEP -- INTERSECTION DATA ---
$DFSDBUX SEGM, NAME=STEP, BYTES=5, START=(6,3825,255)
$DFSDBUX SEGM, NAME=CUSN
$DFSDBUX END

AIF ('&DFARELN GT '6').UX$V61
CHANGEID NAME=$DFSDBUX&SYSDATE&SYSTIME, BASE=R11, X
CSECTNM=$DFSDBUX, RMODE=ANY, AMODE=31, X
LINKAGE=SPEC, CHAIN=NO, BRANCH=NO, SAVE=NO, BREG=NOSET
AGO .UX$V61X

.AUX$V61X ANOP ,
CHANGEID NAME=$DFSDBUX&SYSDATE&SYSTIME, BASE=R11, X
CSECTNM=$DFSDBUX, RMODE=ANY, AMODE=31, COPYRIGHTYEAR=NONE, X
LINKAGE=SPEC, CHAIN=NO, BRANCH=NO, SAVE=NO, BREG=NOSET

.AUX$V61X ANOP ,
CHANGEID IDEND=YES

END DFSDBUX1

Scott Heronimus
Senior Product Developer
BMC Software (USA) © BMC Software 2003
Finding CSECTs within LPA load modules in virtual storage

In the course of development of certain types of system application, the ability to programmatically locate CSECT addresses within LPA-resident modules can sometimes prove useful. Unfortunately, the information that would facilitate such location, ie the displacement of a given CSECT within the load module, while present in the linkage editor control information for the module, is not retained in virtual storage when the module is actually loaded. Thus, it becomes necessary to refer back to the linkage editor CESD (Composite External Symbol Dictionary) control records, which form a portion of the load module’s contents in its LPALST library on DASD. From these, the desired CSECT can be located, and its load module displacement extracted. This value, when added to the output of a system service call that identifies the module’s virtual storage load address, precisely identifies the virtual storage address of the afore-mentioned CSECT.

FNDCSCT is a statically or dynamically-called routine that performs the above function. It is passed the load module and CSECT names as parameters, and, upon completion of a successful location operation, returns the associated virtual storage address in register 0 and a return code of 0 in register 15. WTO-type error messages and a non-zero return code in register 15 are generated upon recognition of any conditions that preclude successful completion. Note that the location technique employed will result in the appearance of one or more IEC141I 013-18 exception messages if the desired module is not found in the first of two or more LPALST libraries. These do not signal error conditions if the module is found in a subsequent LPALST library. If the module is not found in any LPALST library, then IEC141I 013-18 messages will appear for all such libraries, and a further ‘FNDCSCT007E Load module loadmodulename was not found' message will signal the error condition.
FNDCSCT should be link-edited as a stand-alone load module into any desired load library. No specific linkage editor attributes need be assigned.

Calling sequences for FNDCSCT are as follows.

Static call:
```
call  FNDCSCT,(loadmod,csect)
ltr r15,r15
bnz errorrtn
lr rx,r0  retain result value
loadmod dc cl8'loadmodulename'
csect dc cl8'CSECTname'
```

Dynamic call:
```
link ep=FNDCSCT,(loadmod,csect)
ltr r15,r15
bnz errorrtn
lr rx,r0  retain result value
loadmod dc cl8'loadmodulename'
csect dc cl8'CSECTname'
```

CODE

```
fndcsct amode 31
fndcsct rmode 24
fndcsct csect
r0 equ 0
r1 equ 1
r2 equ 2
r3 equ 3
r4 equ 4
r5 equ 5
r6 equ 6
r7 equ 7
r8 equ 8
r9 equ 9
r10 equ 10
r11 equ 11
r12 equ 12
r13 equ 13
r14 equ 14
r15 equ 15
stm r14,r12,12(r13)  entry linkage
lr r12,r15
using fndcsct,r12
st r13,savearea+4
```
l a  r15, savearea
st  r15,8(r13)
l r  r13,r15
b  fcsc0020          branch to start
return  ds  0h
xr  r15,r15           exit linkage
l  r13,4(r13)
l  r14,12(r13)
lm  r1,r12,24(r13)
br  r14              return
fcsc0020  ds  0h
lr  r9,r1             save pal pointer
l  r3,cvtprtr         CVT pointer
icm  r3,15,cvtsmext-cvt(r3)  CVT extension
bnz  fcsc0050         it's there
wto  'FNDCSCT001E No CVTX address found'
la  r15,8
b  return+2
fcsc0050  ds  0h
icm  r3,15,cvteplps-cvtsvstg(r3)  LPAT address
bnz  fcsc0100         it's there
wto  'FNDCSCT002E No LPAT address found'
la  r15,8
b  return+2
fcsc0100  ds  0h
clc  =cl4'LPAT',0(r3)    really the LPAT?
be  fcsc0150           YES
wto  'FNDCSCT003E LPAT ID check failed'
la  r15,8
b  return+2
fcsc0150  ds  0h
icm  r4,15,4(r3)       number of LPAT entries
bnz  fcsc0200         more than none
wto  'FNDCSCT004E No LPAT entries present'
la  r15,8
b  return+2
fcsc0200  ds  0h
l  r2,0,(r9)          get member name address
mvc  membrtu+6(8),0(r2)  move member name to txt unit
la  r3,9,(r3)        point to first LPAT entry
fcsc0250  ds  0h
cli  0(r3),0         end of entries?
be  fcsc0900         yes, module not found
mvc  dsnamtu+6(44),0(r3)  no, move dsnam to text unit
mvi  alcverb,s99vrbal  prime dynalloc fields
la  r0, dsnamtu
sta  r0, alctual1
la  r0, membtru
sta  r0, alctual2
la  r0, statstu
st  r0, alctua3
la  r0, rtdntu
st  r0, alctua4
oi  alctua4, x'80'
la  r1, alcrbptr

dynalloc
ltr  r15, r15   allocation ok?
bz  fcsc0400   yes
st  r15, allocrc
mvc  wto00250+37(44), dsnamtu+6   move data set name
trt  wto00250+37(44), trtable   search for terminating blank
mvi  0(r1), c'('   member name preparation
lr  r5, r1   retain address of blank
mvr  1(8, r5), membrtu+6   move member name
trt  1(8, r5), trtable   search for terminating blank
mvi  0(r1), c')'   end of member name

cnop  0, 4
wto00250   wto 'FNDCSCT005E Error allocating
mvc  dfdaplp, =a(alcrb)   initialize DAIRFAIL parms
mvc  dfrcp, =a(allocrc)
la  r1, =a(0)
st  r1, dfjeff02
la  r1, =x'4032'
st  r1, dfidp
xc  dfcpplp, dfcpplp
mvc  dfbufp, =a(dfbufs)
la  r1, dfparams

link ep=IKJEFF18   invoke DAIRFAIL
ltr  r2, r15   ok?
bz  fcsc0300   yes
wto 'FNDCSCT006E DAIRFAIL error - return code set to DAIRFAI*
L return code'
lr  r15, r2
b  return+2

fcsc0300  ds  0h
lh  r5, dfbufl1   extract the DAIRFAIL message
sh  r5, =h'5'   set/check message length
ch  r5, =h'112'
bnh  fcsc0320
lh  r5, =h'112'

fcsc0320  ds  0h
ex  r5, exmvc1   move it to the wto

cnop  0, 4
wto00300   wto 'FNDCSCT005E

clc  dfbufl2, =a(0)   any second level message?
be  fcsc0340   no
lh  r5, dfbufl2   yes, extract as well
sh  r5, =h'5'                     set/check message length
ch  r5, =h'112'
bnh  fcsco330
l  r5, =h'112'
fcsco330  ds  0h,
ex  r5, exmvc2      move it to wto
cnop  0, 4
wto00330  wto  'FNDCST005E'

fcsco340  ds  0h
l  r15, allocrc
b  return+2
fcsco400  ds  0h
mvc  library+dcbddnam-ihadcb(8), rtddntu+6  move ddname
open  (library,(INPUT)), mode=31, open the library
tm  library+(dcboflgs-ihadcb), dcbofopn ok?
bo  fcsco1000  yes, module has been located
bal  r14, fcsco7000  no, unallocate this library
la  r3, 45,(r3)    move to next LPAT library
b  fcsco250      recycle
fcsco900  ds  0h
l  r3, 0(, r9)     get load module name addr
mvc  wto00900+32(8), 0(r3) move load module name
 cnop  0, 4
wto00900  wto  'FNDCST007E Load module xxxxxxxx was not found'
la  r15, 8
b  return+2
fcsco1000  ds  0h
get  library
lr  r3, r1 retain its address
cli  0(r3), x'20' CESD record?
bne  fcsco1000 no, get another one
la  r4, 0
icm  r4, 3, 6(r3) get record length
la  r3, 8(, r3)
la  r4, 0(r3, r4) point past last byte
sh  r4, =h'8' back off sufficiently
l  r5, 4(, r9) get passed CSECT name addr
fcsco1150  ds  0h
clr  r3, 4 past upper search limit?
bni  fcsco1000  yes, go get another record
clc  0(8, r3), 0(r5) no, CSECT names match?
bne  fcsco1500 no
tm  8(r3), x'0f' yes, type=SD?
bf  fcsco2000 yes, what we're looking for
fcsco1500  ds  0h
la  r3, 1(, r3) next byte
b  fcsco1150 reiterate
fcsco2000  ds  0h

© 2003. Xerphon UK telephone 01635 33848, fax 01635 38345. USA telephone (303) 410 9344, fax (303) 438 0290.
la r4,0
icm r4,b'0111',9(r3)  load the displacement
 closures library  close the library
bal r14,fcsc7000  unallocate it
l r8,0(r9)  point to load module name
csvquery inepname=(r8),search=LPA,ouloadpt=loadpt look for it
ltr r2,r15  find it?
bz fcsc2050  yes
ch r2,=h'8'  module not found?
be fcsc2020  yes
wto 'FNDCSCT008E CSVQUERY error - return code set to CSVQUER*
Y return code'
l r15,r2
b return+2
fcsc2020 ds 0h
wto 'FNDCSCT009E CSVQUERY could not find the requested load *
module'
la r15,8
b return+2
fcsc2050 ds 0h
l r0,loadpt  load module load point
alr r0,r4  add CSECT displacement
b return
************************
* Unallocation routine *
************************
fcsc7000 ds 0h
st r14,r14save  save return address
mvc ddnamtu+6(8),rtddntu+6  prime unallocation ddname
mvi alcverb,s99vrbun  request unallocation
la r0,ddnamtu  prime text unit pointers
st r0,alcual
oi alcual,x'80'
la r1,alcrbptr  unallocate
dynalloc
mvc rtddntu+6(8),=cl8' '  reinitialize return ddname
l r14,r14save
br r14
************************
* DCB ABEND exit *
************************
fcs8000 ds 0h
lr r3,r1  retain parameter list addr
lr r4,r14  retain return addr
mvi 3(r3),4  ignore the abend
lr r14,R4  restore return addr
br r14
************************
* DCB EODAD routine *
************************
fcsc9000 ds Oh
  l  r2,4(r9)     load CSECT name address
mvc  wto09000+51(8),0(r2)     move CSECT name
cnop 0,4
wto09000 wto 'FNDSCSCT010E No CESD record found for CSECT xxxxxxxx'
close library     close the library
bal  r14,fcsc7000     unallocate it
la  r15,8
b  return+2

*************************
* Executed instructions *
*************************
exmvc1 mvc wto00300+20(0),dfbuft1
exmvc2 mvc wto00330+20(0),dfbuft2

*************
* Data Area *
*************
savearea dc 18f'0'
dsnamtu dc al2(dal dsnam),al2(1),al2(44),cl44'
membrtu dc al2(dal membr),al2(1),al2(08),cl08'
statstu dc al2(dal stats),al2(1),al2(01),x'01'8'
rtddntu dc al2(dal rtddn),al2(1),al2(08),cl08'
ddnamtu dc al2(dun ddnam),al2(1),al2(08),cl08'
library dcb ddname=dummy,macrf=GL,dsorg=PS,recfm=U,lrecl=0,
  blksize=32760,devd=DA,eodad=fcs9000,exlst=exlst
exlst dc 0f'0',x'11',al3(fcs8000)
r14save dc a(0)
loadpt dc a(0)     module load point
allocrc dc f'0'     allocation return code
alcrbptr dc x'80',al3(alcrb)     request block pointer
alcrb ds 0f     request block
alcrbln dc al1(20)     request block length
alcrverb dc al1(0)     verb code
alcflag1 ds 0al2     flags
alcflag11 dc al1(0)     first flags byte
alcflag12 dc al1(0)     second flags byte
alcrsc dc 0al4     reason code fields
alcerrdc dc al2(0)     error reason code
alcinfo dc al2(0)     information reason code
alctxtpc dc a(alctupl)     tupl address
alcrsv01 ds f     reserved
alcflag2 ds 0al4     authorized functions flags
alcflag21 dc al1(0)     first flags byte
alcflag22 dc al1(0)     second flags byte
alcflag23 dc al1(0)     third flags byte
alcflag24 dc al1(0)     fourth flags byte
alctupl ds 0f     text unit pointer list
alctua1 dc a(0)     text unit address 1
alctua2 dc a(0)     text unit address 2
alctua3 dc a(0)     text unit address 3
alctua4 dc a(0)

ikjefddf

ttable dc 256x'0'

org ttable+c' '

dc c' '

org ,

**********

* DSects *

**********

cvt dsect=YES, list=NO
dcbd dsorg=PS, devd=DA
iefzb4d0
iefzb4d2
end

Joel Riemer
946512 (Canada) © Xephon 2003
Embarcadero has announced Versions 7.1 of DBArtisan and Rapid SQL, including enhanced database platform support with particular focus on DB2 Universal Database and new support for OS/390.

DBArtisan is for managing Oracle, Sybase, Microsoft SQL Server, and DB2, enabling administrators to concurrently manage multiple databases from a single console.

Rapid SQL is an integrated development environment that enables developers to create, edit, version, tune, and deploy server-side objects residing on DB2, Microsoft SQL Server, Oracle, and Sybase databases.

With extended support in the new versions for DB2 UDB Enterprise-Extended Edition (EEE), DBAs and database developers don’t have to leave the graphical consoles of DBArtisan or Rapid SQL to administer or develop applications in their partitioned environments.

Version 7.1 provides increased support for OS/390, including a new interface for cross-referencing DBRMs and plans and packages, along with associated SQL statements.


---

IBM has announced Version 1.5 of its XML Toolkit for z/OS and OS/390, based on cross-platform, open source code, and containing a C++ Parser, a Java Parser, and a Java Processor, based on the Apache Software Foundation Xerces and Xalan software.

The XML C++ Parser (XML4C V5.0.0) is designed for enhanced performance via new DOM C++ bindings, provides the ability to prepare and cache grammars, supplies an experimental subset of DOM Level 3, and can optionally exploit z/OS Unicode Services.

The XML Java parser (XML4J V4.1.3) gets a new API for post validation info set and provides the ability to prepare and cache grammars.

Finally, the XSLT Java Processor (LotusXSL-Java Version 2.4.3) provides a prototype for DOM Level 3 xpath, standardized EXSLT extension support, and updated parser support.

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

---

IBM has announced Application Support Facility for z/OS V3R3, which allows users to create documents based on pre-defined templates, text, and data. Users leveraging the Document Composition feature can define the document layout and formatting using IBM Document Composition Facility. V3R3 gets improved document creation functions and combines the V3R2 base function with its Document Composition feature and provides enhancements for administrators.

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