MVS update

224

May 2005

In this issue

3 Considerations when placing page datasets on DASD with PAV enabled
4 Deleted datasets at a glance
9 Making the transition from Assembler to C on the mainframe – revisited
23 No source, no worry – generating source code from the load module
33 Get the most out of subchannels: a STSCH illustration
41 SMS status check REXX routine
44 Syslog analysis using LOGLYZER
75 MVS news

© Xephon Inc 2005

A tc PUBLICATION

MVS Update

Published by

Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor

Trevor Eddolls
E-mail: trevore@xephon.com

Publisher

Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues

A year's subscription to *MVS Update*, comprising twelve monthly issues, costs $505.00 in the USA and Canada; £340.00 in the UK; £346.00 in Europe; £352.00 in Australasia and Japan; and £350.00 elsewhere. In all cases the price includes postage. Individual issues, starting with the January 2000 issue, are available separately to subscribers for £29.00 ($43.50) each including postage.

MVS Update on-line

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

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 $160 (£100 outside North America) per 1000 words and $80 (£50) per 100 lines of code for the first 200 lines of original material. The remaining code is paid for at the rate of $32 (£20) per 100 lines. To find out more about contributing an article, without any obligation, please download a copy of our *Notes for Contributors* from www.xephon.com/nfc.

© Xephon Inc 2005. 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.

Printed in England.

Considerations when placing page datasets on DASD with PAV enabled

We recently experienced some issues with page dataset usage and on investigation discovered some interesting information.

The initial problem manifested itself as high auxiliary storage usage; however, on issuing a **D ASM** from the operator console we noticed that 10 of our 20 page datasets were actually 0% utilized. This obviously confused us, but on doing a little research we identified the devices that were not being used – they were non-PAV type devices. The datasets we had placed on PAV devices were being over used.

We found an APAR OA04644, which described our problem. What IBM has done is recode the Auxiliary Storage Manager from z/OS 1.3 onwards. The recoding allows a separate queue to be created for PARTE control blocks that represent page datasets residing on Parallel Access Volumes. This works as follows: if a page out occurs, the Auxiliary Storage Manager will make a search for which page dataset to place the page onto. First of all it searches the PAV device queue, before looking at other device types. If a PAV device is located with a response time of less than the average of all other device types, it will be chosen as the candidate for the page-out. This results in PAV devices getting more pages than non-PAV devices. This, according to IBM, is intentional.

The actual implementation of the separate queue for PAV devices introduces a potential risk to systems that page a lot. We overcame the problem by moving the page datasets on the non-PAV volumes to PAV volumes, and everything is balanced again. However, if you are not aware of this change, you may find that your paging environment is not acting the way you expect it to.

IBM has actually now updated its documentation to reflect the

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

fact that any page datasets residing on PAV devices will be examined first when Auxiliary Storage Manager is searching for a page dataset to page out to. The bottom line is that when the Auxiliary Storage Manager selects a dataset, any paging datasets that reside on Parallel Access Volume devices are examined first because of the reliability and performance characteristics of these devices.

Because of this, preference will be given to PAV devices – and it is normal to see higher utilization on the PAV-resident datasets when compared with non-PAV resident datasets.

John Bradley
Systems Programmer
Meerkat Computer Services (UK)
© Xephon 2005

Deleted datasets at a glance

PROBLEM ADDRESSED

Every now and then it happens that someone from a site's large user base asks the 'perennial' question: "Where is my dataset? It used to be in that catalog, on that volume, but it is not there any more. Who deleted it?"

During the course of recent reviewing and restructuring of DR procedures, the very same question occurred more frequently than usual, and prompted me to search for a quick, simple, and easy-to-use solution that would supply the straightforward answer.

SOLUTION PROPOSED

In a search for a solution I have looked at the SMF manual to find out whether there is any record that would allow me to determine who deleted or renamed a dataset. It came as no

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

surprise that actually there are such SMF records. At this point, I would suggest that you first check whether you are collecting these SMF records at all. They are fairly voluminous, and lots of installations don't bother. When enabled by the SMFPRM*xx* TYPE parameter, SMF creates type 17 and 18 records, which are written each time a dataset is deleted or renamed. Record type 17 is written when a non-temporary DASD dataset or a temporary DASD dataset is deleted/ scratched. This record contains the dataset name, number of volumes, and volume serial numbers. The job name, time, and date that the reader recognized the JOB card (for this job) constitute the job's identification.

However, you should note that you will not see the userid (or RACF ID) in the type 17 records of the user who actually deleted the dataset. You may get tempted to look for a RACF ID at the supplied field SMF17UID, but that field is not supposed to be the RACF ID. If fact, the SMF manual states that the content of this field is taken from the SMF Common Exit Parameter Area, not from the USER= parameter on the job statement. In particular it is copied from the 8 bytes starting at offset 20. This area is initialized to blanks and may be updated by SMF exits such as IEFUJI, IEFUJV, IEFUSI, and IEFACTRT. The SMF CEPA is mapped by the JMR (Job Management Record), which is mapped by the IEFJMR macro. If you want this to contain the RACF ID of the job, it would probably be easiest to write an IEFUJI exit to take the ID from the ACEE and put it into the field in the JMR. You can find the type 17 description in macro IGGSMF17 in SYS1.MACLIB.

On the other hand, it may happen that SMF type 17 records are not being produced during a scratch DASD dataset because the check on the dataset name in IGG0290D was not consistent with the SMF manual's description of a TEMPORARY or SYSTEM-generated dataset name. In other words, DADSM was interpreting a non-temporary dataset name as a temporary dataset name: IGG0290D was checking for 'SYS' in positions 1–3, '.T' in positions 9–10, and '.' in

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

position 17. Thus, a dataset name like SYSX.SMF.TYPE110.* was considered when writing SMF records during a scratch. If you experience that kind of problem, take a look at the APAR database as to find a PTF for APAR OW43581, which corrects the SMF type 17 fields. Installing APAR OW41664 corrupts the SMF type 17 fields JOB, READTIME, and LOCLINFO, and the DSNAME has 'SYS' in columns 1–3. New APAR OW43581 corrects the problem created when the strengthening of DADSM's name checking by OW41664 destroyed the register used to build the type 17 SMF record (dataset SCRATCH).

Also please note that you may use the REC parameter, in the SMFPRM*xx* parmlib member, to specify that record type 17 is to be collected. REC(ALL) specifies that record type 17 is to be written for both temporary and non-temporary datasets. REC(PERM) specifies that SMF is to write record type 17 only for non-temporary datasets; it is not to be written for temporary datasets (ie datasets having names that start with SYSyyddd.Thhmmss, either from DSN=&&datasetname or from the absence of any dataset name).

It may happen that a particular dataset has not been deleted at all, but renamed. SMF writes record type 18 each time a non-VSAM dataset defined by a DD statement (either explicitly or implicitly) is renamed. This record contains the old dataset name, the new dataset name, the number of volumes, and the volume serial numbers.

A detailed description of the layout of SMF type 17 and 18 records can be obtained from the manual *MVS System Management Facilities (SMF)* – SA22-7630-03.

CODE

Based on the record description obtained from the manual mentioned above, a simple report writer was written. The code is a two-part stream. In the first part (DUMP178) SMF records 17 and 18 are dumped/copied from SMF datasets to a file, which can be used as a base of archived records.

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

In the second part (DS1718), the captured records are formatted and four reports are produced. These reports are grouped into two sets: one set reports all deleted datasets, while the second set provides a summary of renamed datasets.

The dataset deletion report set displays the job or system task performing the deletion, along with the dataset name, volume serial of the dataset, as well as the date and time the deletion took place.

The dataset rename report set displays the job or system task performing the renaming, along with the old dataset name, new dataset name, volume serial of the dataset, and the date and time the rename took place.

```
//*------------------------------------------------------------
//* The SMF extract job.
//* This job will extract the SMF records type 17 & 18 from
//* selected SMF dataset (weekly or current)
//*------------------------------------------------------------
//DUMP178   EXEC PGM=IFASMFDP,REGION=ØM
//INDD      DD DSN=your.smf.ds,DISP=SHR
//OUTDD     DD DSN=&&SMF1718,DISP=(NEW,PASS),
//          SPACE=(CYL,(25,5)),DCB=(your.smf.weekly.ds)
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
    INDD(INDD,OPTIONS(DUMP))
    OUTDD(OUTDD,TYPE(17,18))
/*
//DS1718   EXEC PGM=ICETOOL
//TOOLMSG  DD SYSOUT=*
//DFSMSG   DD SYSOUT=*
//RAWSMF   DD DSN=&&SMF1718,DISP=SHR
//SMF1718  DD DSN=&&TEMPA,SPACE=(CYL,(9,9)),UNIT=SYSDA
//SMF17    DD DSN=&&TEMP7,SPACE=(CYL,(9,9)),UNIT=SYSDA
//SMF18    DD DSN=&&TEMP8,SPACE=(CYL,(9,9)),UNIT=SYSDA
//SYS17    DD DSN=&&TEMPS,SPACE=(CYL,(9,9)),UNIT=SYSDA
//SYS18    DD DSN=&&TEMPT,SPACE=(CYL,(9,9)),UNIT=SYSDA
//JOBDEL   DD SYSOUT=*
//JOBREN   DD SYSOUT=*
//SYSDEL   DD SYSOUT=*
//SYSREN   DD SYSOUT=*
//TOOLIN   DD *
COPY FROM(RAWSMF)  TO(SMF1718) USING(SMFI)
COPY FROM(SMF1718) TO(SMF17) USING(SMF7)
DISPLAY FROM(SMF17) LIST(JOBDEL)                   -
  TITLE('Job deleted datasets') DATE(4MD/)         -
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
  HEADER('Lpar')        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('Jobname')     ON(19,8,CH)                    -
  HEADER('Volser')      ON(95,6,CH)                    -
  HEADER('Dataset')     ON(45,44,CH)
COPY FROM(SMF1718) TO(SMF18) USING(SMF8)
DISPLAY FROM(SMF18) LIST(JOBREN)                       -
  TITLE('Job renamed datasets') DATE(4MD/)             -
  HEADER('Lpar')        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('Jobname')     ON(19,8,CH)                    -
  HEADER('Volser')      ON(139,6,CH)                   -
  HEADER('Old ds name') ON(45,44,CH)                   -
  HEADER('New ds name') ON(88,44,CH)
COPY FROM(SMF1718) TO(SYS17) USING(SMF9)
DISPLAY FROM(SYS17) LIST(SYSDEL)                       -
  TITLE('System tasks deleted datasets') DATE(4MD/)    -
  HEADER('Lpar')        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('Jobname')     ON(19,8,CH)                    -
  HEADER('Volser')      ON(95,6,CH)                    -
  HEADER('Data set')    ON(45,44,CH)
COPY FROM(SMF1718) TO(SYS18) USING(SMFT)
DISPLAY FROM(SYS18) LIST(SYSREN)                       -
  TITLE('System tasks renamed datasets') DATE(4MD/)    -
  HEADER('Lpar')        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('Jobname')     ON(19,8,CH)                    -
  HEADER('Volser')      ON(139,6,CH)                   -
  HEADER('Old ds name') ON(45,44,CH)                   -
  HEADER('New ds name') ON(88,44,CH)
/*
//SMFICNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,17,OR,6,1,BI,EQ,18)
//SMF7CNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,17,AND,
  (19,4,CH,NE,C'DFSM',AND,
   19,4,CH,NE,C'IXGL',AND,
   19,4,CH,NE,C'DBTD',AND,
   19,4,CH,NE,C'DSND'))
//SMF8CNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,18,AND,
  (19,4,CH,NE,C'DFSM',AND,
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
   19,4,CH,NE,C'IXGL',AND,
   19,4,CH,NE,C'DBTD',AND,
   19,4,CH,NE,C'DSND'))
//SMF9CNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,17,AND,
  (19,4,CH,EQ,C'DFSM',OR,
   19,4,CH,EQ,C'IXGL',OR,
   19,4,CH,EQ,C'DBTD',OR,
   19,4,CH,EQ,C'DSND'))
//SMFTCNTL DD *
  OPTION SPANINC=RC4,VLSHRT
  INCLUDE COND=(6,1,BI,EQ,18,AND,
  (19,4,CH,EQ,C'DFSM',OR,
   19,4,CH,EQ,C'IXGL',OR,
   19,4,CH,EQ,C'DBTD',OR,
   19,4,CH,EQ,C'DSND'))
/*
```

Mile Pekic
Systems Programmer (Serbia and Montenegro) © Xephon 2005

Making the transition from Assembler to C on the mainframe – revisited

PASSING PARAMETERS AND ARGUMENTS

This article extends on a previous article in *MVS Update* (see issue 216, September 2004) that discussed the general problems of migrating from Assembler to C. The current article discusses the particular problems associated with passing parameters to and from C programs and functions. For completeness, some parts are taken from the original article.

PASSING ARGUMENTS

All common mainframe programming languages (Assembler, C, COBOL, PL/I) differentiate between the form with which

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

arguments are passed to a main program (namely the program that is initially invoked, such as with the EXEC PGM= JCL statement) and to a subprogram (namely a program invoked with a CALL statement or function call). This discussion ignores any 'exotic' calling conventions, such as XPLINK.

A related topic concerns the returning of arguments. Unlike traditional Assembler, C is a stack-based language; this has consequences when arguments are returned. The stack contains temporary data maintained by the run-time environment (Language Environment). Each program that is Language Environment-conformant takes space from the stack when called and returns the space when it exits. For want of a better term, 'traditional Assembler' refers to Assembler programs that conform to the calling conventions that go back to the original OS, ie the Assembler program saves the caller's parameters in a save area provided by the caller, creates a static save area for use by lower-level called programs. Any data areas required by the Assembler program are located in static memory, typically in the same CSECT as the program code. Only in exceptional circumstances are data areas allocated dynamically, for example to provide space for very large tables. Assembler programs that use dynamic storage areas, for example Language Environment-conforming programs and re-entrant programs, are, in effect, stack-based and are subject to similar restraints as C programs.

Main program

C programs differ from Assembler programs in not only the form of the arguments but also the method used to pass arguments. Both Assembler (also COBOL and most other mainframe languages) and C have different conventions for the arguments depending on whether a main program or a subprogram (function) is called.

Whereas an Assembler (COBOL, etc) main program receives the address (in register 1) of a pointer to the length-prefixed argument list, a C main program intrinsically receives two

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

parameters: the number of passed arguments and the array of pointers to the passed arguments (the invoked program name is passed as the first argument followed by the calling argument automatically parsed into blank-delimited subarguments; each argument is passed as a 0-delimited character string). This mimics the behaviour of personal computer compilers and so simplifies the porting of applications. The parsing can be suppressed by specifying the NOARGPARSE compiler option, in which case the passed array of pointers contains just two entries: the invoked program name and the specified invocation parameter, each of which is passed as a 0-delimited character string.

The C behaviour for passed arguments also has a significant difference when compared with other common programming languages (Assembler, COBOL, etc): namely, the passed arguments are automatically converted to lower case (even for the NOARGPARSE compiler option). There are, however, means of countering this behaviour.

Note: the conversion to lower case is performed character-by-character. This means any hexadecimal codes that happen to coincide with upper-case characters will also be converted, for example X'01C2' (namely, X'01',C'B') would be converted automatically to X'0192' (X'01',C'b').

Case 1 (standard behaviour)

For example, the JCL statement:

```
// EXEC PGM=TESTPROG,PARM='ALPHA  BETA'
```

would cause the following parameters to be generated (note: to better illustrate the behaviour, two blanks are specified between ALPHA and BETA):

Assembler:

```
+-----+
¦     ¦?--- register 1
+-----+
   ¦
   ¦
```

```
   ?
   +--------------+
   ¦11¦ALPHA  BETA¦
   +--------------+
```

C:

```
    +-----+
argv¦  3  ¦
    +-----+

    +-----------------+
argc¦     ¦     ¦     ¦
    +-----------------+
       ¦     ¦     ¦
       ¦     ¦     ¦    +----------+
       +---------------?¦testprog¦#¦
             ¦     ¦    +----------+
             ¦     ¦
             ¦     ¦    +-------+
             +---------?¦alpha¦#¦
                   ¦    +-------+
                   ¦
                   ¦    +------+
                   +---?¦beta¦#¦
                        +------+
```

where # represents the 0-delimiter ('\0', namely X'00').

The following C code fragment lists the passed parameters:

```
int i;

int main(int argc, char *argv[]) {

  for (i=0; i < argc; i++)
    printf("argv: %s\n", argv[i]);
}
```

Case 2 (behaviour with the NOARGPARSE compiler option)

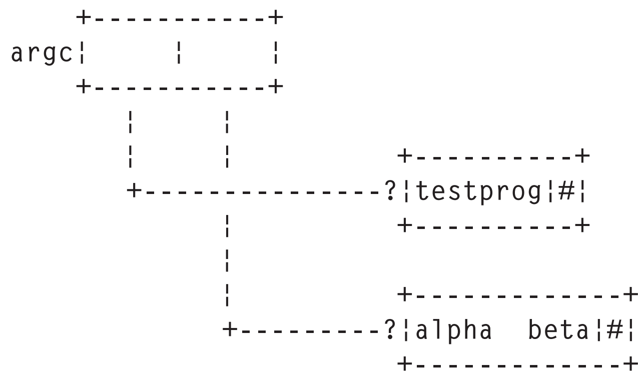
For example, the JCL statement:

```
// EXEC PGM=TESTPROG,PARM='ALPHA  BETA'
```

would cause the following parameters to be generated:

```
    +-----+
argv¦  2  ¦
    +-----+
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.



where # again represents the 0-delimiter (‘\0’).

Obtaining the true EXEC parameter

The [0] element from the __osplist built-in array (defined in <stdlib.h>) can be used to obtain the contents of register 1 when the program was invoked, namely, the address of a pointer to the EXEC parameter. This is the EXEC parameter as specified by the caller, ie it is not converted. The __osplist built-in array is briefly mentioned in the *Language Environment Programming Reference*.

```
#include <stdlib.h>
#include <stdio.h>

/* the following structure declares the form of the EXEC parameter */
struct {
  short vl;
  char vd[101];
} *pv;

main() {
  pv=(void *)__osplist[0];
  pv=(void *)pv;
  printf("vl:%hd\n",pv->vl); /* EXEC parameter length */
  printf("vd:%s\n",pv->vd);  /* EXEC parameter data   */
}
```

Subprograms

Whereas, in Assembler, arguments are normally passed by reference, in C, arguments are implicitly passed by value. This means that the address-of operator (&) must be explicitly applied to pass an argument by reference (exception: strings

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

are always passed by reference). To ensure that the arguments are passed in the correct form, the calling program should include a prototype declaration of the called function (subprogram).

Calling program (code fragment):

```
void FUNC(int, int *); /* function prototype */
int x1 = 1, x2 = 2;

FUNC(x1, &x2); /* invoke function */
Called function
void FUNC(int i1, int *pi2) {
  printf("p1: %d\n", i1);
  printf("p2: %d\n", *pi2);
}
FUNC() makes the following display:
p1: 1
p2: 2
```

Calling legacy programs

Many programs make use of existing services, which can be classified as being legacy programs. In C, functions are called rather than programs. Although, as standard, such functions are invoked using C calling conventions, conventional legacy programs (subroutines) with the operating system calling convention can also be called when they have been identified with the #pragma linkage (pgmname,OS) instruction (*pgmname* is the name of the associated program).

The following code fragments show how a legacy program (here ISPLINK, the program interface for ISPF services) is invoked dynamically from C and Assembler. The two versions have the same functionality.

Note: for simplicity, the code to check that the program was loaded is omitted (null pointer returned from fetch() or zero address from the LOAD macro).

C code fragment:

```
#pragma linkage (OSFUNC,OS)
typedef int OSFUNC();
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
int rc;                   /* return code */
OSFUNC *fptr;

char sn[] = "VCOPY ";     /* service name */
char option[] = "MOVE ";  /* option */
char vnl[] = "(VN1 VN2)"; /* name list    */
int  vla[2] = {8, 8};     /* length array */
char vna[2][8];           /* value array  */

fptr = (OSFUNC *)fetch("ISPLINK");
rc =  (*fptr)(sn,vnl,&vla,&vna,option);
```

The equivalent Assembler code fragment:

```
         LOAD  EP=ISPLINK
         LR    15,Ø      pass entry point address
         CALL  (15),(SN,VNL,VLA,VNA,MOVE),VL

SN       DC    CL8'VCOPY '   service name
OPTION   DC    CL8'MOVE '    option
VNL      DC    C'(VN1 VN2)'
VLA      DC    F'8,8'
VNA      DS    2CL8
```

Special attention must be paid to the form of the arguments passed to the invoked program in C. The address-of operator (&) must be applied to the two arrays (vla and vna) to pass the required pointer. Determining the correct form of the passed arguments is the most difficult part of calling legacy programs from C.

Returning arguments

Because local variables in a C program (and normally re-entrant Assembler programs) are contained in the stack, such variables are no longer available after the program has finished execution (or more correctly, cannot be guaranteed to be available – this is a danger, because, depending on when a variable is used, it might still be 'present') and so they cannot be used as the returned result. Only a value that can be returned in a register (eg short, int, float, char [a single character, but not a string – even if the string consists of just a single character], pointer, or a direct value that can be expressed as such) can be used as a direct return value, eg the field specified on the return instruction (obviously a pointer

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

may not contain the address of a local [non-static] variable). Instead, a static variable (ie a variable with static storage duration), a variable allocated in the heap (eg allocated with malloc()–), or a variable passed from the calling program (obviously as a pointer) must be used. Another possibility is the use of a global variable, namely a variable that is defined outside all program blocks.

The following demonstration programs all have the same functionality. The calling program passes an integer (the value 3) to the called function. The called function increments this passed value by 1 and returns this new value (4) to the calling program.

To keep the techniques as general and practical as possible, it is assumed that the calling program and the called function are separate compile units that are statically linked together. However, with the exception of referencing global variables, the techniques can also be used for dynamically loaded programs. Similarly, to avoid over-complication, the shown code fragments, although complete, are often trivial, but suffice to illustrate the concepts.

Calling program (result used directly):

```
int FØ2(int); /* function prototype */


void main()

{

  if (FØ2(3) == 4) puts("four");

}
```

Called function:

```
int FØ2(int i)
{
  int n; /* define local variable */
  n = i; /* transfer caller's argument to local variable */
  n++;   /* increment */

  return n; /* return the local variable */
}
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

Because the calling value does not save the returned result, the function call can only be used immediately in an instruction.

Furthermore, in this particular simple case, the use of a local variable is not necessary. Because a copy of the caller's variable is passed, this copy could be incremented and returned.

Called function (alternative):

```
int FØ2(int i)
{
  i++;   /* increment copy of caller's argument */

  return i; /* return the incremented copy */
}
```

Calling program (result returned as a pointer):

The called function can be written in several ways:

```
int *FØ2(int); /* function prototype */

void main()
{
  int *pi;
  pi = FØ2(3);
}
```

Called function (result defined as a static variable):

```
int *FØ2(int i)
{
  static int n; /* static local variable */
  n = i; /* transfer caller's argument to local variable */
  n++;

  return &n; /* return address of local variable */
}
```

Note: if rather than **static int n;** merely **int n;** was written, the calling program would access the wrong data field, but because the address is probably still valid, no error would be indicated.

Called function (explicit memory allocation [heap] with malloc()):

```
#include <stdlib.h> /* header for malloc() */
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
int *FØ2(int i)
{
  int *pn;

  pn = malloc(sizeof(*pn)); /* allocate heap memory */
  *pn = i; /* transfer caller's argument to the allocated heap memory */
  (*pn)++;
  return pn; /* return the address of the allocated heap memory */
}
```

Called function (result defined as a global variable):

```
int n; /* <global variable, defined outside all program blocks */

void FØ2(int i, int *pi)
{
  n = i;
  n++;
  return &n;
}
```

Calling program directly accesses an external (global) variable:

If the accessed variable is defined as a global variable (as in this case), it can be accessed directly when it is declared as being external in the calling program.

Because good software engineering practice deprecates the use of global variables (unless their use is unavoidable), such a usage should be used with discretion.

```
#include <stdio.h>

extern int n;

int *FØ2(int);

void main()
{
  FØ2(3);
  printf("n:%d\n",n); /* display the result */
}
```

This is equivalent to EXTERN variables in Assembler, for example:

- L 1,=V(N) – address of external variable

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

- L 0,0(1) – load contents of external variable into register 0.

Calling program (result returned directly):

```
int FØ2(int); /* prototype */

void main()
{
  int i;
  i = FØ2(3);
}
1.4.3.1 Called function
int FØ2(int i)
{
  int n; /* local variable */
  n = i;
  n++;

  return n;
}
```

This form can be used here because the result can be passed as a register (in this particular case, as a general purpose register, but floating-point registers can also be used when appropriate).

Calling program (result returned in a variable passed by the calling program):

```
void FØ2(int, int *); /* prototype */

void main()
{
  int i; /* <i> will contain the returned value */
  FØ2(3, &i);
}
```

In this case, the calling program passes the address of a field (defined locally within the calling program) in which the called function returns its result.

Called function:

```
void FØ2(int i, int *pi)
{
  int n; /* work field */
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
  n = i;
  n++;
  *pi = n;

  return; /* no explicit return value */
}
```

Scope of the returned result

The form with which a variable is defined determines its scope (possibly better known as its lifetime). The following list summarizes the various scopes. To distinguish between the calling program (main program) and the called function, the terms ‘program’ and ‘function’, respectively, are used.

-	function result – the function result is transient, but can be used in the appropriate instruction.
-	function invocation – if nothing is specified, the variable is in the stack which exists only while the function is being invoked.
static	function lifetime – the variable exists until the function is deleted from memory (implicitly or explicitly).
malloc()	program lifetime – the variable exists while the program is loaded or until the variable is explicitly deleted from memory.
file scope	program lifetime – the variable exists while the program is loaded. The variable is global and can be accessed directly with the extern qualifier. This is not good software engineering practice.

The programmer should be aware of the consequences of explicitly allocated variables (via malloc()), ie extensive use without the appropriate housekeeping can lead to memory leakage. Memory leakage occurs when memory is allocated but never freed after use. Because explicit memory allocation is relatively time-consuming, it should be used with caution when a large number of variables need to be allocated.

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

FORWARDING A PARAMETER LIST

It is sometimes necessary to forward a parameter list, for example to call another subprogram. In Assembler, it suffices just to forward the contents of register 1; namely, the pointer to the parameter list. Because C normally needs to build the call at compilation time, a variable parameter list – processed with argv[] (main program) or va_arg() (subprogram) – must be converted into a fixed parameter list (eg an array) and then forwarded to the called subprogram, but even in this case the full flexibility of Assembler is not possible.

Assembler program

Because the Assembler program does not need to know the format of the parameter list, the parameter list address can simply be forwarded. No differentiation needs to be made between a main program and a subprogram:

```
FO1      CSECT
         BAKR  14,Ø   save caller's registers
         BASR  12,Ø   load base register
         USING *,12   set base register
         L     2,1    save caller's parmlist
...
* call subprogram
         L     1,2    reload caller's parmlist
         CALL  FO2    note: no parameters
...
         PR    ,      program return
```

For simplicity, the BAKR/PR instruction-pair is used to save and restore the caller's registers, although the usual calling conventions can also be used.

C program

Here is an example of the procedure involved. The first function, fo1(), is called with a variable parameter list, fo1(int, ...), where the second and following arguments are strings. (The number of the following arguments must be known, here it is specified as the first argument.) The arguments are passed to the second function, fo2(), as a fixed parameter list,

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

fo2(int, char *pca[]). To simplify the code, the array has a fixed length.

```
#include <stdio.h>
#include <stdarg.h>

int fo1(int, ...);
int fo2(int, char *pca[]);

int main()
{
  fo1(2, "alpha", "beta");
  return Ø;
}

int fo1(int n, ...) {
  va_list argptr;
  char *pc;
  int i;
  #define N 1Ø /* max. number of parms */
  char *pca[N];

  printf("N:%d\n",n); /* number of varparms */
  va_start(argptr, n);
  for (i=Ø; (i < n) & (i < N); i++) {
    pc = va_arg(argptr, char *);
    puts(pc);
    pca[i] = pc;
  }
  va_end(argptr);

  fo2(n,pca);
  return n;
}

int fo2(int n, char *pca[]) {
  char *pc;
  int i;

  printf("N:%d\n",n); /* number of parms */
  for (i=Ø; i < n; i++) {
    pc = pca[i];
    puts(pc);
  }
  return n;
}
```

Anthony Rudd
Systems Programmer (Germany)
© Xephon 2005

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

No source, no worry – generating source code from the load module

Source code loss can strike anyone. Back-up devices may have failed or been misconfigured, or perhaps back-ups were not made at all. The loss may represent recent updates or additions – even the entire application; or other demands may require the modification of an old application for which the source code has been misplaced

It was the belief for years that you should re-write programs from scratch should the original source code become lost. Fortunately, there is now an alternative for recovering source code from object code. This article describes a formal technique for the recovery of COBOL and Assembler source code statements from the load module.

THE LOAD MODULE ANALYSIS

First let us understand the load module and the information it can provide us with. Only load modules contain accurate information about the code running on a system. Load modules contain the machine instructions that are actually executed when a program runs. Each load module is composed of a number of CSECTs (control sections) containing the main program instructions, run-time routines supplied with compilers, and subroutines previously created by users. In addition to executable machine instructions, each CSECT also contains information – such as languages, compiler release levels, and compiler and linkage-editor options – that identifies what it does and how it was created. A load module begins with a source module written by a programmer in a high-level programming language. The source module must then be submitted to a compiler for the language the source is written in (there is a unique compiler for each programming language). The compiler changes the programming language into machine language – the machine instructions a processor can execute

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

– and puts them in an object module. The object module is then fed into the linkage editor, which transforms it into a load module.

When submitting a program to a compiler, programmers can specify parameters or options. The parameters selected can change the machine instructions generated from the source code. In most cases, a compiler inserts information about itself, such as version and release level, as well as information identifying what options were selected when the program was compiled into the load module. The format and location of this information may vary depending on the compiler and even the release of the compiler used. Like compilers, the linkage editor also has parameters that can be selected to vary the way a program executes. It also allows users to add IDR (Identification Record) data to load modules. And like most compilers, the linkage editor inserts identifying information about itself, parameters selected, and IDR data added into load modules as it creates them.

The linkage editor also inserts additional CSECTs into load modules for CALL statements that invoke statically linked system run-time routines or user-created subroutines to perform frequently used functions. Run-time routines are supplied with compilers or as part of the Language Environment (LE). Subroutines are user written and must have been previously compiled. Even load modules that dynamically invoke run-time routines from a resident library may have some statically linked routines, such as ‘bootstrap’ modules. Each subroutine contains its own identifying information about the compiler that created it.

DISASSEMBLING

Disassembling, or source code recovery, is based on the following concepts or steps:

1 Delink

2 Disassembly

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

3 Decompile

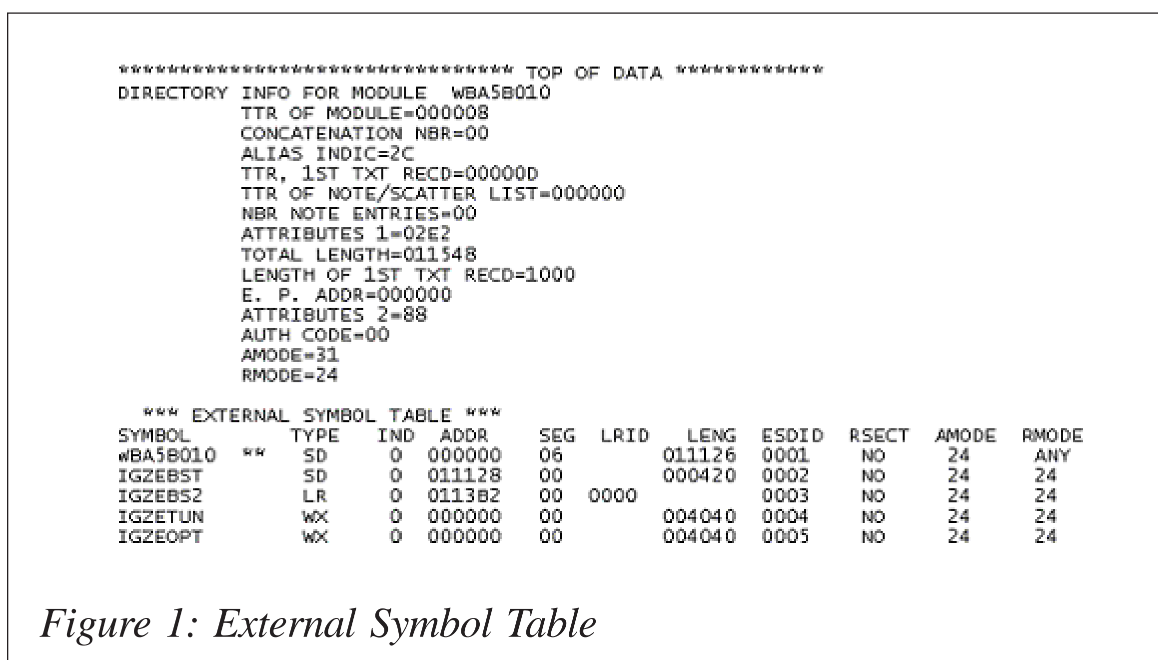
   – pattern matching

   – operand analysis

4 Validation

Step 1: delink

The first step in recovering source code from an executable program is to break down the executable into its core components, known as control sections or CSECTs. This process identifies the individual modules that make up the executable. They include run-time modules provided by the operating system to incorporate system-level services such as CICS and/or DB2.Each of these CSECTs is intertwined in the executable, so in order to focus on one individual CSECT, the executable must first be delinked. From here, the basic 'program' can be disassembled. CSECT information can be obtained from the External Symbol Table in the SYSPRINT card from the spool after the first run of the disassembler on the load module. This is illustrated in Figure 1.



```
********************************* TOP OF DATA ************
DIRECTORY INFO FOR MODULE  WBA5B010
          TTR OF MODULE=000008
          CONCATENATION NBR=00
          ALIAS INDIC=2C
          TTR, 1ST TXT RECD=00000D
          TTR OF NOTE/SCATTER LIST=000000
          NBR NOTE ENTRIES=00
          ATTRIBUTES 1=02E2
          TOTAL LENGTH=011548
          LENGTH OF 1ST TXT RECD=1000
          E. P. ADDR=000000
          ATTRIBUTES 2=88
          AUTH CODE=00
          AMODE=31
          RMODE=24

  *** EXTERNAL SYMBOL TABLE ***
SYMBOL          TYPE   IND  ADDR     SEG  LRID    LENG   ESDID   RSECT  AMODE  RMODE
WBA5B010  **     SD      0  000000   06          011126  0001     NO     24     ANY
IGZEBST          SD      0  011128   00          000420  0002     NO     24     24
IGZEBS2          LR      0  0113B2   00   0000           0003     NO     24     24
IGZETUN          WX      0  000000   00          004040  0004     NO     24     24
IGZEOPT          WX      0  000000   00          004040  0005     NO     24     24
```

Figure 1: External Symbol Table

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

Step 2: disassembly

The compiled ‘program’ is really a representation of ones and zeros, called binary. An example of the internals of a binary program file are shown below:

```
00000101 00011111 00001000 00000000 00000000 00000000 11010010 00010011
01100001 01100000 01100001 01001000 01011000 01000000 11010010 00110000
01000001 11100000 10111000 01101000 11010010 00000011 01000000 00100000
11000000 01001000 01011000 11110000 01000000 00001100 00000101 00011111
00000100 11000010 00000000 00000000 01000111 11110000 10111000 01110100
01011000 10110000 11000001 00111000 01000111 11110000 10110110 11111100
11010010 00000011 11010010 01100000 11010010 01100100 01000001 00000000
10111000 10000110 01010000 00000000 11010010 01100100 01000111 11110000
10110001 10001000 11010010 00000011 11010010 01100100 11010010 01100000
11010010 01011111 10100000 00000000 10000000 00000000 01001000 00000000
11000001 00001000 01011000 01000000 11010010 00101100 01000001 11100000
10111000 10101000 01011000 11110000 01000000 00001100 00000101 00011111
00001000 00000000 00000000 00000000 01011000 10110000 11000001 00111000
```

Programs represented in this manner, while accurate, are extremely difficult to inspect and interpret. At a somewhat higher level, the same program can be viewed from a byte-oriented or hexadecimal level of granularity, as shown below:

```
47F0F028  00C3C5C5  000000B0  00000014  47F0F001  98CEAC00
00000000  00000000  90ECD00C  4110F038  98EFF04C  07FF0000
00000748  000000AE  00000000  00000498  00000E38  000000CA
E2C1D4D7  D3C5F340  F2F0F0F5  F0F1F2F0  F1F4F3F1  F0F4F0F2
0000076C  E0E86C4C  00800000  0288000A  80204000  00000000
00000018  00008000  40404040  0007E2C1  D4D7D3C5  F3400500
0000FFFF  FFB20000  00000000  00380000  00080000  00060000
0D600000  00050000  00000000  00000000  00000000  00000000
00000108  00000434  00000476  00000498  FFFFFFFC  00001000
C940D4C1  D2C540C9  E340D7D6  E2E2C9C2  D3C5E2E8  E2D6E4E3
```

While somewhat easier to decipher, hexadecimal representation of a program is still at too low a level for most programmers. The same code can be further represented at a higher level – Assembler. The previous examples are shown below as a set of Assembler language statements and operands:

```
00040C  04    00000000                       DC     F'0'
000410  04    00000000                       DC     F'0'
000414  02    0000                           DC     H'0'
000416  08    D7D6E260C9404040               DC     C'POS-I    '
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
ØØØ41E  Ø8    4Ø4Ø4Ø4Ø4Ø4Ø4Ø4Ø                 DC    C'        '
ØØØ426  Ø8    4Ø4Ø4Ø4Ø4Ø4Ø4Ø4Ø                 DC    C'        '
ØØØ42E  Ø6    4Ø4Ø4Ø4Ø4Ø4Ø                     DC    C'      '
ØØØ434  Ø4    582Ø9128            AØØØ434     L     R2,296(,R9)
ØØØ438  Ø6    D2Ø12ØØØAØØC                    MVC   Ø(2,R2),AØØØ114
ØØØ43E  Ø6    D2Ø12ØØ8AØØC                    MVC   8(2,R2),AØØØ114
ØØØ444  Ø6    D2Ø72Ø1ØAØ32                    MVC   16(8,R2),AØØØ13A
ØØØ44A  Ø6    D2Ø32Ø18AØ14                    MVC   24(4,R2),AØØØ11C
ØØØ45Ø  Ø6    D2Ø32Ø2ØAØ14                    MVC   32(4,R2),AØØØ11C
ØØØ456  Ø6    D2Ø32Ø28AØ14                    MVC   4Ø(4,R2),AØØØ11C
ØØØ45C  Ø6    D2Ø72Ø3ØAØ2A                    MVC   48(8,R2),AØØØ132
ØØØ462  Ø6    D2Ø32Ø38AØ14                    MVC   56(4,R2),AØØØ11C
ØØØ468  Ø4    92ØE2Ø4Ø                        MVI   64(R2),X'ØE'
ØØØ46C  Ø4    92ØF2Ø48                        MVI   72(R2),X'ØF'
ØØØ47Ø  Ø4    583ØDØ8C                        L     R3,14Ø(,R13)
ØØØ474  Ø2    Ø7F3                            BR    R3
ØØØ476  Ø4    92F18ØØØ            AØØØ476     MVI   Ø(R8),C'1'
ØØØ47A  Ø6    D2Ø18ØØ8AØBC                    MVC   8(2,R8),AØØØ1C4
ØØØ48Ø  Ø6    D2Ø28Ø1ØAØB5                    MVC   16(3,R8),AØØØ1BD
```

The code illustrated in the above three examples differs in the format in which the information is presented. Disassembly of the code, no matter what the original source language, is the first concept in source code recovery. The disassembler produces Assembler language source statements and a pseudo-listing using object code as input. You can use the Assembler language source file and listing to understand the program, debug it, and recover the lost source code.

We use DISASM, which is a one-pass disassembler that produces an Assembler language source program from a CSECT within a load module. Control cards permit specification of areas containing no instructions; allow base registers to be provided so that symbolic labels may be created during disassembly, and definition of DSECTs to be used during disassembly. Control statements permit the specification of areas containing instructions, data or uninitialized data areas, provide base registers so that symbolic labels are created during disassembly, and define the DSECTs used during disassembly.

Registers are denoted thus:

- Access Registers are denoted by A0, A1,...A15.

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
-------------------------------------------------------------------------
  //DISASM    EXEC PGM=ASMDASM,PARM='options'                          1
  //SYSLIB    DD DSN=user.loadlib,DISP=SHR                             2
  //SYSPRINT DD SYSOUT=A,DCB=BLKSIZE=121                               3
  //SYSPUNCH DD DSN=user.command.asm,DISP=(,CATLG),                    4
  //             UNIT=SYSDA,DCB=BLKSIZE=3200,
  //             SPACE=(TRK,(5,2),RLSE)
  //SYSIN DD *
  module_name csect_name                                               5
  other control statements
  .
  .
  .
  //*COPYLIB DD DSN=user.copy.name,DISP=SHR                            6
-------------------------------------------------------------------------
          Sample Disassembler MVS JCL
```

Figure 2: JCL statements used

- Control Registers are denoted by C0, C1,...C15.
- Floating Point Registers are denoted by F0, F1,...F15.
- General Purpose Registers are denoted by R0, R1,...R15.
- Vector Registers are denoted by V0, V1,...V15.

The disassembler provides informational comments for recognized SVCs, and for various branch instructions to aid in creating a documented source program. You invoke the disassembler as a batch program using JCL. Figure 2 shows the JCL statements you can use.

Step 3: decompile

Pattern matching

The next phase in the recovery of COBOL is that every COBOL verb will generate a distinct pattern of machine instructions that can be analysed and then programmatically stored, retrieved, and applied to the programs to be recovered. These patterns may be as simple as a single machine instruction or they may encompass many dozens of machine

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

instructions. Just as solving a jigsaw puzzle requires a sharp eye for subtle differences in the shapes of the puzzle's pieces, source code recovery requires an intimate knowledge of COBOL verbs and the patterns of machine code that they produce. For source code recovery, COBOL compiler research begins by creating source code files that provide examples of all the various programming elements in all their flavours, particularly Procedure Division verbs. Next, these research files are compiled with the PMAP option or, in the case of COBOL II, the LIST option. Finally, the compile listings are analysed to determine what verbs generate what particular patterns of machine instructions.

The PERFORM verb provides a good example of this type of research. The source code version of a simple PERFORM verb followed by the LIST option of the compile listing for that same PERFORM is shown below:

```
PERFORM 1ØØØ-OPEN-FILES THRU 1ØØØ-EXIT.

D2 Ø3 D 214 D 234       MVC   214(4,13),234(13)           PSV=2 VN=Ø2
41 ØØ B Ø32             LA    Ø,Ø32(Ø,11)                 GN=Ø2
5Ø ØØ D 234             ST    Ø,234(Ø,13)                 VN=Ø2
58 BØ C Ø58             L     11,Ø58(Ø,12)                PBL=1
47 FØ B 11A             BC    15,11A(Ø,11)                PN=Ø3
D2 Ø3 D 234 D 214       MVC   234(4,13),214(13)           VN=Ø2 PSV=2
```

Analysing this pattern, and others like it, leads to the following conclusions: the pattern begins and ends with MVC instructions and the operands of these instructions are reversed. Finally, two instructions prior to the last statement in the pattern there is an unconditional branch to the paragraph to be performed. These instructions are distinctive enough for us to conclude that any identical patterns found in a COBOL program we are recovering must be generated by a simple PERFORM. Having done this type of analysis for all different verbs, and for many different COBOL compilers and compiler options, we can develop a DECOMPILER that can readily generate source code from the load module. Below is a simple REXX program to identify the starting and ending offset of a simple PERFORM pattern in the disassembled code:

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/* MACRO TO COLLECT ALL THE MVC STATEMENTS FROM THE DISASSEMBLED CODE */
"ISREDIT MACRO"
ADDRESS ISPEXEC VGET ('OUTPDS') PROFILE
ADDRESS ISREDIT "EXCLUDE ALL"
ADDRESS ISREDIT "FIND ALL ' MVC '"
ADDRESS ISREDIT "DELETE ALL X"
ADDRESS ISREDIT "RESET"
ADDRESS ISREDIT "CRE .ZF .ZL &OUTPDS"
ADDRESS ISREDIT "CANCEL"
EXIT

 /****************************** REXX ******************************/
/*   THIS PROGRAM READS THE FILE WITH MVC STATEMENTS AND IDENTIFIES THE
     PERFORM PATTERN.THE START AND END OFFSET OF THE PERFORM PATTERN IS
     STORED IN PERFORM_BEGIN.I PERFORM_END.I              */

ADDRESS TSO "ALLOC DA("OUTPDS") F(INFILE) SHR REUSE"
ADDRESS TSO "EXECIO * DISKR INFILE (FINIS STEM IN1."
COUNT1 = Ø
   DO I=1 TO IN1.Ø
     DO J=1 TO 2
        IF J=1 THEN
        DO
          X = LEFT(IN1.I,72)         ;           ORIGINAL = X
        END
        ELSE
        DO
          K = I + 1                   ;          X = LEFT(IN1.K,72)
        END
        NO_OF_WORDS            =       WORDS(X)
        COLUMN_OPERAND   =       NO_OF_WORDS
        COLUMN_INST             =       NO_OF_WORDS - 1
        OPERAND                      =        WORD(X,COLUMN_OPERAND)
        INST                     =       WORD(X,COLUMN_INST)
        IF J=1 THEN
          PARSE VALUE OPERAND WITH VAL1 '(' TEMP_1 '),' VAL2 '('
        ELSE
          PARSE VALUE OPERAND WITH VAL3 '(' TEMP_1 '),' VAL4 '('
     END /* J */
     IF VAL1 = VAL4 &  VAL2 = VAL3 THEN
     DO
       TEMP_1 = WORD(ORIGINAL,1)
       IF PROGRAM_START_OFFSET < X2D(TEMP_1) THEN
       DO
          COUNT1 = COUNT1+1
          PERFORM_BEGIN.COUNT1 = TEMP_1
          PERFORM_END.COUNT1 = WORD(X,1)
          A_VAL2.COUNT1 = VAL2
       END
     END
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
  END /* I */

ADDRESS TSO "FREE DD (INFILE)"
```

Operand analysis

The pattern description language does a good job of recovering COBOL PROCEDURE DIVISION verbs, but does not provide much information about other elements in a COBOL program, particularly data items in file descriptions, WORKING STORAGE, and the LINKAGE SECTION. These items are recovered by applying the next concept in source code recovery, which is referred to as operand analysis. Operand analysis is the detailed examination and eventual analysis of the operands of the machine instructions making up the PROCEDURE DIVISION. This examination and analysis allows us to determine the relative placement of a data item and the PICTURE clause that is most appropriate for the data item. Data item recovery relies on analysis of research programs in much the same manner as the PROCEDURE DIVISION verb recovery. The compile listings are analysed and permit the determination of the types of PICTURE clauses that generate specific machine instructions. The following example illustrates that type of research and involves an elementary data item that is defined as a PIC 9:

```
Ø1 NUM1 PIC 999.

COMPUTE NUM1 = NUM1 + 1

F212 D1A8 9ØØØ       PACK 424(2,13),Ø(3,9)        TS2=Ø
96ØF D1A9            OI   425(13),X'ØF'           TS2=1
FA1Ø D1A8 AØ53       AP   424(2,13),83(1,1Ø)      TS2=Ø
F321 9ØØØ D1A8       UNPK Ø(3,9),424(2,13)        NUM1
96FØ 9ØØ2            OI   2(9),X'FØ'              NUM1+2
47FØ B134            BC   15,3Ø8(Ø,11)            GN=6(ØØØ3E8)

-------------------------------------------------------------------------

Ø1 NUM1 PIC 999.
Ø1 NUM2 PIC 999 COMP-4.

MOVE NUM1 TO NUM2.
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
F272 D180 9000        PACK 384(8,13),0(3,9)        TS2=0
960F D187             OI   391(13),X'0F'           TS2=7
4F30 D180             CVB  3,384(0,13)             TS2=0
4030 9008             STH  3,8(0,9)                NUM2
```

Our research has determined that whenever a PIC 9 item is involved in an arithmetic operation the final machine instruction generated will be determined with:

```
OI <  >,X'0F'.
```

Finding this type of machine instruction during source code recovery allows us to conclude that the data item being operated on is an elementary item of type PIC 9. In addition the PACK and UNPK instructions indicate the length in bytes of the item, which can then be expressed as the number of 9s in the PICTURE clause.

```
PACK 424(2,13),0(3,9)
```

Here 3 indicates the length of the NUM1.

The actual operand analysis occurs during both the disassembly and the decompilation of the program being recovered. As each operand is encountered, the probable PICTURE clause value based on the machine instruction acting on the operand is determined and stored for later use during the source code generation phase of the decompiler.

Operand analysis allows you to recover accurately any data item that is referenced within the program.

Step 4: validation

The last step of the source code recovery methodology is simply to prove that the source code has been recovered accurately. In order to do this, we take the resulting recovered source code and compile it using the same version of the compiler (and compile options) that created the original executable program. The compile options that were originally specified are key for validation. The newly-created object module is then disassembled and compared with the disassembly of the original executable. While a 100% identical

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

object module is seldom expected, the technician must compare the original object with the newly-created object and verify that the resulting program is 'functionally' equivalent.

SUMMARY

The process of recovering source code from object code is a formal process that results in source code that is guaranteed to be 100% functionally equivalent to the original code regardless of the original COBOL dialect, the options under which the program was compiled, or the operating system that the application is running on; and the decompiler forms the basis for detecting and recreating the corresponding source code.

T S Laxminarayan
System Programmer
Tata Consultancy Services (India)

© Xephon 2005

Get the most out of subchannels: a STSCH illustration

SUBCHANNEL

A subchannel number is a system-unique half-word value within a channel subsystem. The subchannel is addressed by the basic I/O instructions (SSCH, MMSCH, RSH, etc) and its status and configuration information is maintained in a special channel subsystem block – the subchannel-information block (SCHIB). You can easily retrieve that block using a STORE SUBCHANNEL (STSCH) instruction.

Let's take a look at the SCHIB. It has a twelve-word structure that contains three major fields:

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

- The path management control word (PCMW)
- The subchannel status word (SCSW)
- The model dependent area/measurement block.

Most of the information contained in the SCHIB, particularly the PCMW, is propagated by IOS to UCB. For example the Last Path Used Mask (LPUM), which contains the channel path that was last used for communicating between the channel subsystem and the device, is also maintained in the UCB IOS extension.

To illustrate my point, the following code will scan each subchannel (from 0 to 65535) and issue a STSCH to get the SCHIB and process path masks so it can print a simple path configuration for each device (only operational and last used path). Before issuing a STSCH, general register 1 must contain the subsystem-identification word (SID). This SID has been extended to provide both the CSS ID and MIF ID to which the subchannel is configured, but this information is implicitly provided at run-time by the LPAR hypervisor in a transparent way, so programs don't need to code the entire SID, only the last word (binary number one and a half-word subchannel number).

I/O MEASUREMENT

Regarding I/O measurements, with the advent of z/990 and z/architecture comes a new 64-byte ECMB (replaces the old CMB) and the Extended Measurement Word (EMW). Both these measurement blocks contain device connect, disconnect, pending, busy time, and control unit queueing time, etc.

EMW is an extension of the IRB, and measurement data is provided on an I/O operation basis, while ECMB has a more sampling approach.

In order to access ECMB, the subchannel must have the following bits on in the related SCHIB:

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

- Measurement Mode Enable.
- Measurement Block Format Control (format 1).
- Measurement Block Update Enable.

It can then be pointed to by an absolute storage address – the measurement block address within the SCHIB. A much simpler way to access the ECMB is to get the ALET for the dataspace, the measurement block index from the UCB, and the compute displacement (see in the code, only the SSCH count is printed).

This sample program also gets the Configuration Data Record (CDR) of a device finally to provide a configuration report.

In short, SCHIB and ECMB, plus the UCB, can be easily combined to code an efficient device and path I/O monitor or a configuration report utility.

CSSSCAN

```
CSSSSAN TITLE 'Scan Subchannels '
***********************************************************************
* Name     : CSSSCAN
* Requires : APF
* Function : Scan subchannel (Ø=>X'ffff') and get SCHIB.
*          : Get PATH configuration through LPUM/PAM   SCHIB values
*          : Check Measurements Mode
*          : Access IOS data space to obtain Measurements Block info
*          :  (only the accumulated SSCH count is printed).
*          : Obtain  CDR to inform about manufacturer and model
***********************************************************************
CSSSSAN CSECT
         COPY EQU
CSSSSAN AMODE 31
CSSSSAN RMODE 24
         BAKR  R14,Ø                      Save
         LR    R12,R15                    Using R12
         USING CSSSSAN,R12                Addressability
CONN     EQU   *
         OPEN (OUT,OUTPUT),MODE=31        Open sysout
         PUT   OUT,TITLE                  Put 3  header
         PUT   OUT,TITLE2                 lines
         PUT   OUT,TITLE3
TØØ      EQU   *
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
         L      R5,=XL4'0000FFFF'           Load 65535 X'ffff'
         XR     R6,R6                       Clear R6
**********************************************************************
* Scan Subchannel  0=>65535                                          *
**********************************************************************
T01      EQU    *
         MODESET MODE=SUP,KEY=ZERO          To mode Sup
         STH    R6,SCIW+6                   Store subchannel numb
         L      R1,SCIW+4                   Load Sub Ident Word
         STSCH  SCHIB2                      Get Schib
         LA     R2,SCHIB2                   SCHIB addr
         USING  SCHIB,R2                    addr for SCHIB
         STH    R6,ZSCN                     Edit
         UNPK   ZSCN(5),ZSCN(3)             Subchannel Number
         TR     ZSCN(5),TAB
*
         L      R3,SCHIP                    Select SCH with a
         LTR    R3,R3                       non null Interrup.
         BZ     NEXTSCH                     (Same as UCB)
         MVC    WUCBPTR(4),SCHIP            Save UCB pointer
         TM     SCHFLG2,SCHV                Check Device Validity
         BO     OKDEVNO                     If Device validy bit off
         B      NEXTSCH                     then skip this sch
OKDEVNO  EQU    *
         MVC    WRKDEVD(2),SCHDEVNO         Save to access CDR
         MVC    ZDEVNUM(2),SCHDEVNO         Save Device numbr
         UNPK   ZDEVNUM(5),ZDEVNUM(3)       Edit Device numbr
         TR     ZDEVNUM(5),TAB
GETPATH  EQU    *
         LA     R9,ZCHPTAB                  Load CHPID edit table
         LA     R10,SCHCHPID                Chpids from SCHIB
         MVC    ZCHPTAB(24),=CL24' '        Clear print zone
         MVC    WRKCHPD(2),=H'0'            Zeroes
         LA     R3,8                        Max chpid per device = 8
PATH00   EQU    *
         XR     R8,R8                       Clear R8
**===============================================================*
* Check PAM Path Available Mask
**===============================================================*
         IC     R8,SCHPAM                   Path available Mask byte
         LR     R4,R3                       Compute displacement
         BCTR   R4,0                        within the mask with
         SRL    R8,0(R4)                    8-loop count -1 - shift
         N      R8,=XL4'00000001'           Check Last bit only
         LTR    R8,R8                       If zero this path is
         BZ     NOPATH                      Not available to z/OS
         XR     R8,R8                       Clear
**===============================================================*
* Print Chpid
**===============================================================*
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
         XR    R7,R7                   Clear R7
         IC    R7,Ø(R1Ø)               Get chpid
         ST    R7,PARM                 Store
         UNPK  WKZ(2*L'PARM+1),PARM(L'PARM+1)     format
         TR    WKZ(2*L'PARM),TAB                  Edit chpid number
         MVC   Ø(2,R9),WKZ+6
         MVI   2(R9),C' '
**==============================================================*
* Check PUM 'Last Path Used Mask'
**==============================================================*
         IC    R8,SCHLPUM              Last Used Path Mask byte
         SRL   R8,Ø(R4)                Shift with loop index
         N     R8,=XL4'ØØØØØØØ1'       Last bit only
         LTR   R8,R8                   if null
         BZ    NXTPATH                 Process next chpid
         MVC   ZLAST(2),WKZ+6          edit last used chpid
         MVC   WRKCHPD+1(1),Ø(R1Ø)
         B     NXTPATH                 Process next Chpid
NOPATH   EQU   *
         MVC   Ø(2,R9),=CL2'..'        Indicates not available
NXTPATH  EQU   *
         LA    R9,3(R9)                Next Chp zone in output
         LA    R1Ø,1(R1Ø)              Next Chp in SCHIB TAB
         BCT   R3,PATHØØ               To next chpid
         CLC   ZCHPTAB(24),=CL24' '    if at least one
         BNE   GETCDR                             path found, getcdr
         MVC   ZCHPTAB(24),=CL24'** No Operational Path **'
         B     GETCMB                             Print detail line
**==============================================================*
* GetCDR : Get Configuration data report with noio (<= from z/OS)
**==============================================================*
GETCDR   EQU    *
         GETMAIN RU,LV=65535            GetMain
         LR    R4,R1                    Keep storage addr
          IOSCDR DEVN=WRKDEVD,CHPID=WRKCHPD+1,CDRAREA=(R4),CDRLEN=65535,*
               READ=NOIO,STATUS=CDRSTAT
         LTR   R15,R15                  Indicates No CDR
         BNZ   NOCDR                    if request failed
         CLI   NEDFLAGS-NED(R4),X'CØ'   Process NED only
         BE    NOCDR
         MVC   ZCDR(3),NEDMANUF-NED(R4)     Format like
         MVI   ZCDR+3,C'-'                  IBM-
         MVC   ZCDR+4(6),NEDTYPEN-NED(R4)   IBM-ØØ9Ø32
         MVI   ZCDR+1Ø,C'-'                 IBM-ØØ9Ø32-
         MVC   ZCDR+11(3),NEDMODN-NED(R4)   IBM-ØØ9Ø32-ØØ5
         FREEMAIN RU,LV=65535,A=(R4)        Free Storage
         B     GETCMB
NOCDR    EQU   *
         MVC   ZCDR(16),=CL16'N/A'          Indicates no CDR
FREECDR  EQU   *
```

```
         FREEMAIN RU,LV=65535,A=(R4)        Freemain
**=================================================================*
* GetCMD : Get Configuration data report with noio (<= from z/OS)
**=================================================================*
GETCMB   EQU   *
         LA    R2,SCHIB2                    Restore addr for SCHIB
         TM    SCHFLG2,SCHMM                Is MM Bit on
         BNO   NOMM                         Measurement off- skip
         L     R2,16                        To CVT
         L     R2,CVTOPCTP-CVT(R2)          To RMCT
         L     R3,RMCTCMCT-RMCT(R2)         To CMCT
         CLI   CMCTVERS-CMCT(R3),X'Ø2'      Level 2 => ECMB support
         BNE   NOECMB                       Not z/99Ø support
         CLI   CMCTECMBFLAGS-CMCT(R3),CMCTECMBMODE See OAØ6164
         BNE   NOECMB                       Not z/99Ø support
         IOSCMXA UCBPTR=WUCBPTR,UCBCXPTR=WUCBCXT   To UCB Comm Ext
         L     R4,WUCBCXT                   Save to addr
         L     R4,UCBIEXT-UCBETI(R4)        UCB ios Extension
         LH    R9,UCBMBI-UCBRSTEM(R4)       Get MBI
         LR    R7,R9                        *64 to reach entry
         SLL   R7,6                         ECMB
         L     R1,16                        To CVT
         L     R1,CVTOPCTP-CVT(R1)          To RMCT
         L     R2,RMCTCMCT-RMCT(R1)         To CMCT
         L     R8,CMCTECMBPTR-CMCT(R2)      To ECMB pointer
         LAM   R9,R9,CMCTECMBALET-CMCT(R2)  ALET
         SAC   512                          Switch AR mode
         LA    R9,Ø(R7,R8)                  Get ECMB addr
         USING ECMB,R9
         L     R1,ECMBSschRschCount         Get ssch/rsch count
         SAC   Ø                            Switch AR mode
         CVD   R1,PL8                       Convert and
         MVC   MASK1(1Ø),=X'4Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø2Ø' print ssch count
         ED    MASK1(1Ø),PL8+3              Edit
         MVC   ZSSCH(1Ø),MASK1
         B     WRITEIT                      Print line
NOMM     EQU   *
         MVC   ZSSCH(1Ø),=CL1Ø'MM-off'      Measurement Mode off
         B     WRITEIT
NOECMB   EQU   *
         MVC   ZSSCH(1Ø),=CL1Ø'Not supp '   Not Supported
WRITEIT  EQU   *
         MODESET MODE=PROB,KEY=NZERO        Back to prob mode
         PUT   OUT,ZSCN                     Print detail
         MVI   ZSCN,X'4Ø'                   Raz output zone
         MVC   ZSCN+1(159),ZSCN
NEXTSCH  EQU   *
         LA    R6,1(,R6)                    Next subchannel
         BCT   R5,TØ1                       Loop
         B     EXIT                         Exit
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
         CNOP  Ø,4
EXIT     EQU   *
         PR
         CNOP  Ø,4
**==================================================================*
** SubChannel-Identification Word => bit                            *
** CSSID and MIF ID are provided by LPAR Hypervisor                 *
** see IBM J. RES. & DEV. VOL. 48 NO. 3/4 MAY/JULY 2ØØ4             *
** see z/Architecture Principles of Operation SA22-7832-Ø2          *
**==================================================================*
SCIW     DC   A(Ø)
         DC   XL2'ØØØ1',XL2'ØØ'
SCHIB2   DC   12F'Ø'
OUT       DCB  MACRF=PM,DSORG=PS,RECFM=FB,LRECL=8Ø,DDNAME=OUT
         CNOP  Ø,4
CDRSTAT  DC    X'ØØ'
         CNOP  Ø,4
WUCBPTR  DC    F'Ø'
WUCBCXT  DC    F'Ø'
MASK1    DC    X'4Ø212Ø2Ø2Ø2Ø2Ø2Ø6B2Ø2Ø2Ø'
*==============================================================
* Edit zone                                                    =
*==============================================================
*
TITLE    DC C'Subc Dev  Chpids                       Last Activity'
TITLE1Ø  DC C' Physical                                           '
TITLE2   DC C'Num  Adr  Available                    Used (ECMB)  '
TITLE2Ø  DC C' Description                                        '
TITLE3   DC C'----+----+-----------------------+----+--------'
TITLE3Ø  DC C'+------------+---------------------------------'
*
ZSCN     DC  CL4' ',C' '                          SubChannel Number
ZDEVNUM  DC  CL4' ',C' '                          Device number
ZCHPTAB  DC  CL24' ',C' '                         Chpid Table
ZLAST    DC  CL4' ',C' '                          Last Used Channel
ZSSCH    DC  CL1Ø' ',C' '                         Accumulated SSH count
ZCDR     DC  CL16' ',C' '                         CDR returned from system
TAB      DC    15XL16'ØØ'
         DC    C'Ø123456789ABCDEF'
*=======================================================*
*=======================================================*
WRKCHAN   DS   F
WRKDEVD   DS   H
WRKCHPD   DS   H
PARM     DC    F'Ø'
PARM2    DC    F'Ø'
PL8      DS    PL8
*
          CNOP  Ø,4
RETCODE   DC   F'Ø'
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
RSNCODE   DC   F'Ø'
WRKUCB    DS   CL1ØØ
WKZ       DS   CL8
REALECMB  DS   CL64
         CNOP  Ø,4
         LTORG
         IOSDDEVI
         IEFUCBOB LIST=YES,PREFIX=YES
         IHACDR
         IOSDUCBP
         IHASCHIB DSECT=YES
         IRACMCT
         IRARMCT
         CVT   DSECT=YES
*=======================================================
* ECMB layout                                          =
*=======================================================
ECMB     DSECT
ECMBSschRschCount    DS F Number of SSCH/RSCH
ECMBSampleCount      DS F Number of SSCH/RSCH collected
ECMBConnectTime      DS F Summation of deviceconnect times
ECMBPendingTime      DS F Summation of SSCH/RSCH request pending
ECMBDisconnectTime   DS F Summation of subchannel disconnect times
ECMBCUqueueingTime   DS F Summation of control queueing times
ECMBDeviceActiveOnly DS F Summation of device  active-onlytimes
ECMBDeviceBusyTime   DS F Summation of device busy times
ECMBInitialCmdRespT  DS F Initial commandresponse time
         END
```

SAMPLE OUTPUT

```
 Subc Dev  Chpids                    Last Activity Physical
 Num  Adr  Available                 Used (ECMB)   Description
 ----+----+------------------------+----+--------+------------+----
 ØØØØ Ø1ØØ AD 2E AB 86 33 32 2F 87   87          171 N/A
 ØØØD Ø1ØD AD 2E AB 86 33 32 2F 87   33         1624 IBM-ØØ339Ø-B3C
 ØØ12 Ø112 AD 2E AB 86 33 32 2F 87   2F         1624 IBM-ØØ339Ø-B3C
 Ø1ØØ Ø2ØØ 9F 1C B3 34 C3 8Ø CF C7   9F      1136879 STK-ØØ349Ø-B4Ø
 Ø14Ø Ø421 9A .. .. .. .. .. .. ..        MM-off     N/A
 Ø277 Ø6E8 C6 .. .. .. .. .. .. ..   C6   MM-off     IBM-ØØ2Ø84-CTC
 Ø42C ØF8C F8 .. .. .. .. .. .. ..   F8   MM-off     IBM-ØØ1732-ØØ1
 Ø431 ØFFØ 2B .. .. .. .. .. .. ..   2B   MM-off     IBM-ØØ9Ø32-ØØ5
 ØC1D 428A .. 9C ØC A8 CC AE .. ..   AE        3962Ø HTC-ØØ339Ø-B3C
```

David Harou
Systems Programmer (France)
© Xephon 2005

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

SMS status check REXX routine

We have recently had some issues with adding volumes into SMS, replacing previously replaced volume serial numbers. We found that in most cases these had been left in status QUINEW or DISNEW by the work performed to remove them previously.

This was easily corrected, but did cause some confusion and also meant Help Desk calls were logged.

To avoid this situation I have written a small REXX EXEC that runs weekly and flags any volumes that have been removed, but are not in an ENABLE SMS status. We can then go in and re-enable these so that if volumes are added over old definitions they are immediately enabled.

This routine ensures that if the person re-using the volumes forgets to check the status we do not get Help Desk calls saying volumes cannot be utilized because of the old status.

The REXX EXEC that we run is shown below. Note that the following line of code is not testing for spaces but actually has hex '0000' coded. However, it prints as follows:

```
     IF SUBSTR(DCOLREC1.C,145,4)  = '     '
```

You can code the hex '0000' by entering HEX mode in ISPF edit.

Here's the SMSVOLCK EXEC:

```
/*REXX*/

/*       ******************************************************   */
/*       * PROGRAM NAME: SMSVOLCK.                            *   */
/*       *                                                    *   */
/*       * PROGRAMMER:   JOHN BRADLEY.                        *   */
/*       *                                                    *   */
/*       * PURPOSE:      THIS PROGRAM READS DCOLLECT VL TYPE  *   */
/*       *               RECORDS AND THEN LOOKS TO SEE WHETHER *   */
/*       *               A VOLUME CODED IN A STORAGE GROUP IS *   */
/*       *               NON-EXISTENT. IF IT IS IT CHECKS     *   */
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/*       *                WHETHER IT IS IN ANY STATE OTHER       *   */
/*       *                THAN 'ENABLE'. IF IT IS THEN A         *   */
/*       *                RECORD IS WRITTEN TO A REPORT SO IT    *   */
/*       *                CAN BE ENABLED.                        *   */
/*       ******************************************************* */
/* READ ALL RECORDS FROM DCOLLECT OUTPUT.                            */

TOTAL = Ø                              /* INITIALIZE TOTAL.          */
"EXECIO * DISKR DCOLIN (STEM DCOLREC1. FINIS)"

DO C = 1 TO DCOLREC1.Ø                 /* LOOP FOR NO. OF RECORDS.   */
IF SUBSTR(DCOLREC1.C,5,2) = 'VL' THEN  /* ONLY PROCESS TYPE 'VL'.    */
 DO                                    /* IF IT IS THEN LOOP.        */
  VOLSER   = SUBSTR(DCOLREC1.C,27,6)   /* GET THE VOLSER.            */

/* GET STORAGE GROUP NAME.                                           */

  DVLSTGRP = LEFT(STRIP(SUBSTR(DCOLREC1.C,91,8),T,'ØØ'X),8)
  DVLCSMSS = SUBSTR(DCOLREC1.C,121,1)  /* GET STATUS.                */

  SELECT                               /* START SELECT.              */
    WHEN DVLCSMSS = 'ØØ'X THEN         /* STATUS ØØ SET TO 'NONE'.   */
       DVLCSMSS = "NONE   "
    WHEN DVLCSMSS = 'Ø1'X THEN         /* STATUS Ø1 SET TO 'ENABLE'. */
       DVLCSMSS = "ENABLE "
    WHEN DVLCSMSS = 'Ø2'X THEN         /* STATUS Ø2 SET TO 'QUIESCE'.*/
       DVLCSMSS = "QUIESCE"
    WHEN DVLCSMSS = 'Ø3'X THEN         /* STATUS Ø3 SET TO 'QUINEW'. */
       DVLCSMSS = "QUINEW "
    WHEN DVLCSMSS = 'Ø4'X THEN         /* STATUS Ø4 SET TO 'DISABLE'.*/
       DVLCSMSS = "DISABLE"
    WHEN DVLCSMSS = 'Ø5'X THEN         /* STATUS Ø5 SET TO 'DISNEW'. */
       DVLCSMSS = "DISNEW "
  END                                  /* END SELECT.                */

  IF SUBSTR(DCOLREC1.C,145,4)  = '    ' THEN /* IF HEX ØØØØ UNITADDR.*/
    DO                                 /* LOOP.                      */
      IF DVLCSMSS ¬= 'ENABLE' THEN     /* IF NOT = 'ENABLE'          */
       DO
        OREC =  ' '                    /* BLANK OREC FIELD.          */
        OREC = ' '||VOLSER||'  '||DVLSTGRP||'  '||DVLCSMSS
        QUEUE OREC                     /* OUTPUT RECORD.             */
        TOTAL = TOTAL + 1              /* ADD 1 TO TOTAL.            */
       END                             /* END LOOP.                  */
    END                                /* END LOOP.                  */
 END                                   /* END LOOP.                  */
END                                    /* END LOOP.                  */

IF TOTAL ¬= Ø THEN                     /* RECORDS PROCESSED?         */
 DO                                    /* YES START LOOP.            */
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
  HDR = ' VOLSER  STORGRP   STATUS'    /* SET HEADER.                */
  PUSH HDR                             /* PUSH ONTO STACK.           */
  "EXECIO 1 DISKW DCOLOUT"             /* WRITE HDR.                 */
  HDR = ' ------  -------   ------'    /* SET HEADER.                */
  PUSH HDR                             /* PUSH ONTO STACK.           */
  "EXECIO 1 DISKW DCOLOUT"             /* WRITE HDR.                 */
  HDR = ' '                            /* SET HEADER.                */
  PUSH HDR                             /* PUSH ONTO STACK.           */
  "EXECIO 1 DISKW DCOLOUT"             /* WRITE HDR.                 */
  "EXECIO * DISKW DCOLOUT (FINIS)"     /* ALL RECORDS DONE, WRITE.   */
 END                                   /* END LOOP.                  */
EXIT Ø                                 /* EXIT WITH Ø.               */
```

The JCL required to run the DCOLLECT to obtain input into the REXX and into later steps that run the EXEC itself are shown below:

```
//JXB7884S    JOB   (JXB),'JXB,J.BRADLEY',CLASS=A,MSGCLASS=H
//*       *************************************************************
//*       * GENERATE DCOLLECT RECORDS FROM ACTIVE SCDS.               *
//*       *************************************************************
//*
//*       *************************************************************
//*       * DELETE DATASETS FROM LAST RUN.                            *
//*       *************************************************************
//*
//STEP1       EXEC  PGM=IDCAMS
//SYSPRINT    DD    SYSOUT=*
//SYSIN       DD    *
  DELETE JXB.JB.DCOLIN
  DELETE JXB.JB.DCOLOUT
/*
//*
//*       *************************************************************
//*       * RUN DCOLLECT AGAINST ACTIVE SMS SCDS.                     *
//*       *************************************************************
//*
//STEP2       EXEC  PGM=IDCAMS
//SYSPRINT    DD    SYSOUT=*
//OUTDD       DD    DSN=JXB.JB.DCOLIN,
//               DCB=(RECFM=VB,LRECL=32576,BLKSIZE=Ø,DSORG=PS),
//               SPACE=(CYL,(2Ø,1Ø),RLSE),
//               DISP=(,CATLG,DELETE)
//SYSIN       DD     *
  DCOLLECT -
  OUTFILE(OUTDD) -
  SMSDATA(ACTIVE)
/*
//*
//*       *************************************************************
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
//*      *                                                       *
//*      * EXECUTE SMSVOLCK REXX EXEC.                           *
//*      *                                                       *
//*      *********************************************************
//*
//STEP3       EXEC  PGM=IRXJCL,PARM='SMSVOLCK',REGION=4M
//SYSTSPRT    DD    SYSOUT=*,RECFM=FBA,LRECL=133,BLKSIZE=Ø
//SYSEXEC     DD    DSN=JXB.JB.CNTL,DISP=SHR
//SYSTSIN     DD    DUMMY
//DCOLIN      DD    DSN=JXB.JB.DCOLIN,DISP=SHR
//DCOLOUT     DD    DSN=JXB.JB.DCOLOUT,
//                DISP=(NEW,CATLG,DELETE),
//                SPACE=(CYL,(1,1),RLSE),
//                DCB=(RECFM=FB,LRECL=8Ø,BLKSIZE=Ø,DSORG=PS)
```

Finally, below is an example of the output from the REXX EXEC.

```
VOLSER          STORGRP          STATUS
------------    --------------   ------------

 PRDAØ1         PROD339Ø         DISNEW
 PRDAØ2         PROD339Ø         DISNEW
 PRDAØ3         PROD339Ø         DISNEW
 PRDAØ4         PROD339Ø         DISNEW
 PRDAØ5         PROD339Ø         DISNEW
 PRDAØ6         PROD339Ø         DISNEW
 PRDA17         PROD339Ø         DISNEW
 PRDA18         PROD339Ø         DISNEW
 PRDA2Ø         PROD339Ø         DISNEW
 PRDA21         PROD339Ø         DISNEW
```

John Bradley
Systems Programmer
Meerkat Computer Services (UK)
© Xephon 2005

Syslog analysis using LOGLYZER

I needed to analyse my syslog data. Aside from using SORT and REXX, I couldn't find any tools that really made the job any easier. SORT allowed me to copy and select certain records, as long as what I was looking for was always in exactly the same place. REXX, of course, allows me to do whatever I

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

want. What I wanted was a way to search for data in each logical syslog record.

Each syslog record is actually a Write to Operator (WTO) message. Some WTOs are Multi-Line WTOs (MLWTO). When the MVS WRITELOG command is used, it externalizes the syslog. Then an external writer is used to copy it to a sequential dataset. This process causes many records in the syslog records to get broken across physical records. Things can get complicated when you are searching for data in the continuation records.

To deal with this, I wrote the LOGLYZER REXX EXEC. LOGLYZER will read a sequential syslog and rejoin all the records. It provides search parameters to select and/or reject records based on the value at a specific word offset. Multiple selection criteria can be used to reduce the amount of output. The qualifying records will be copied to the OUTPUT DD. Additionally, a statistical analysis of the messages is always produced after the processing is completed.

In LOGYLZER, each blank-delimited word in a syslog record becomes a 'token'. The token is the number of the word in the record. Using the following $HASP100 syslog record as an example, each word would have the following token value.

```
  (1)   (2)   (3)    (4)     (5)         (6)      (7)       (8)
CSØ1N Ø2ØØØØØ CSØ1  Ø5Ø26 ØØ:ØØ:Ø1.1Ø STC14981 ØØØØØØ91 $HASP1ØØ
  (9)     (1Ø)  (11)
SMFUDUMP  ON STCINRDR
```

So, if I wanted to find all the records that were $HASP100 messages for SMFUDUMP, I would want to find token 8 equal to $HASP100 and token 9 equal to SMFUDUMP. Using my input format, this would be done using the following format:

```
8=$HASP1ØØ 9=SMFUDUMP
```

Token=value (using the '=' sign) is for selection, and token-value (using the '-' sign) is for rejection. All criteria are 'ANDed' together. Each value is 'wildcarded' against the actual value in the syslog record. Therefore, you can use the entire value

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

or just prefixes, suffixes, or substrings of the actual values.

If I want to find all DSNT501I messages (a seven line MLWTO) from DB2 subsystem DB2B with reason code 00C9008E, but not from job MYTEST, in this example DSNT501I message:

```
 (1)    (2)   (3)       (4)     (5)        (6)      (7)       (8)
CS01M 4000000 CS01     05026 20:11:32.54 STC15059 00000090  DSNT501I  -
 (9)   (10)
DB2B DSNILMCL

   (11)      (12)   (13)        (14)                    (15)
RESOURCE UNAVAILABLE 293 CORRELATION-ID=MYTEST   CONNECTION-ID=DB2CALL
 (16)     (17)
LUW-ID=* REASON

 (18)    (19)  (20)    (21)      (22)           (23)
00C9008E TYPE 00000302 NAME CUSDB001.TSCUS   .X'100756'
```

I would use the following search criteria:

```
8=DSNT501I 9=DB2B 18=00C9008E 14-MYTEST
```

This would select all DSNT501I records from DB2B with reason code 00C9008E and ignore records from job MYTEST.

Here is sample JCL for LOGLYZER:

```
//jobcard…
//LOGLYZER EXEC PGM=IKJEFT01,DYNAMNBR=999,
//          PARM='LOGLYZER * 8=DSNT501I 9=DB2B 18=00C9008E 14-MYTEST'
//SYSEXEC  DD   DSN=your.rexx.pds,DISP=SHR
//SYSLOG   DD   DSN=your.syslog.dsn,DISP=SHR,DCB=BUFNO=60
//OUTPUT   DD   SYSOUT=*
//SYSTSPRT DD   SYSOUT=*
//DIAGMSGS DD   SYSOUT=*
//SYSTSIN  DD   DUMMY
```

Here is some sample LOGLYZER output:

```
OPSROZLF ---- LOGLYZER started 26 Jan 2005 19:32:29 on SY01 --- JOB 4346


Search criteria:

Token 8 equals Value DSNT501

SYSLOG used: SYS0.SYSLOG.DAILY.SY01.SMR05026

559,765 records read from SYSLOG in 3.990338 seconds
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
SYSLOG starts at: 26 Jan 2005 00:00:00.01
SYSLOG ends at  : 26 Jan 2005 17:00:00.79

366,256 expanded records generated in 65.883703 seconds

8 records matched search criteria, output is in OUTPUT DD

457 unique Message ID's found


MSGID Counts:


Sorted Descending by Frequency    Sorted Alpha by MSGID

SARSTC22       83662      33%     !AOP0000           23      <1%
$HASP250       17211       6%     $HASP000          123      <1%
$HASP100       11517       4%     $HASP003           20      <1%
IEF196I         7084       2%     $HASP097           38      <1%
DTM1460I        6743       2%     $HASP100        11517       4%
TSS7053I        6111       2%     $HASP112            1      <1%
$HASP373        4953       1%     $HASP119            1      <1%
$HASP395        4937       1%     $HASP125            3      <1%
SVTM052I        4783       1%     $HASP150         1404      <1%
$HASP540        4724       1%     $HASP160          779      <1%
IEF404I         4677       1%     $HASP250        17211       6%
.
.
ARC6018I           1     <1%      WER169I             1      <1%
ARC0415I           1     <1%      WER177I             1      <1%
ARC6019I           1     <1%      WER211B             1      <1%
ARC1540I           1     <1%      WER246I             1      <1%
ARC1541I           1     <1%      WER410B             2      <1%
IKT122I            1     <1%      WER416B             3      <1%

MSGID Total: 248,712


OPSROZLF-LOGLYZER ended 26 Jan 2005 19:33:53 83.7 on SY01 RC=0-JOB 4346
```

LOGLYZER REXX EXEC

```
/*********************************************************************/
/*                               REXX                                */
/*********************************************************************/
/* Purpose: Find strings in the SYSLOG                               */
/*-------------------------------------------------------------------*/
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/* Syntax:  LOGLYZER readlimit token1 token2 token3 token4 token5... */
/*-------------------------------------------------------------------*/
/* Parms: READLIMIT  - The number of records to read ('*' means all) */
/*        TOKENn     - Unlimited number of tokens                    */
/*                                                                   */
/* Notes: TOKENs are a pair of the positional word and the value     */
/* ie     8=IEA630I 10-OPSROZ                                        */
/*                                                                   */
/* Use '=' for an equals comparison or '-' for a not equal compare   */
/* The leading characters of a value can be used for wildcarding     */
/* ie     8=IEA630I 10-OPS                                           */
/*                                                                   */
/* OPS would match OPS, OPSX, OPSM02, OPSROZ, OPSROZXX, etc.         */
/*                                                                   */
/* Optional keyword tokens:     (only support with '=' comparison)   */
/* LPAR=SYSA             (replaced with the target LPAR)             */
/* DATE=TODAY            (replaced with today's julian date)         */
/* DATE=YESTERDAY        (replaced with yesterday's julian date)     */
/* DATE=05085            (any valid julian date)                     */
/* TIME=HH:MM:SS.hh      (any valid time)                            */
/* TIME=10:              (would get 10:00:00.00-10:59:59.99)         */
/* MSGID=DSNT501         (would get all DSNT501I messages)           */
/* TASKID=JOB12345       (can be jobnum, console, internal/instream) */
/* CMD=$C                (can be any command or prefix)              */
/*                                                                   */
/* All WTOs (single or multi-line) are searched as a single record   */
/* Single line WTOs are copied as a single record to the OUTPUT DD   */
/* Multi-line WTOs are copied as separate lines to the OUTPUT DD     */
/*                                                                   */
/* Return Codes:                                                     */
/*         00 - Everything worked                                    */
/*         01 - Records truncated while writing to OUTPUT DD         */
/*         02 - Unexpected record types found in the SYSLOG          */
/*         04 - No records matched the search criteria               */
/*         08 - All parms missing                                    */
/*         10 - Tokens missing                                       */
/*         11 - Parm or Token format error                           */
/*         12 - TSO ALLOC or EXECIO error                            */
/*                                                                   */
/* Sample JCL:                                                       */
/* //Jobcard...                                                      */
/* //LOGLYZER  EXEC PGM=IKJEFT01,                                    */
/* //          PARM='LOGLYZER * 8=IEA630I 10-OPSROZ'                 */
/* //SYSEXEC   DD   DSN=your.exec.pds,DISP=SHR      EXEC PDS         */
/* //SYSLOG    DD   DSN=your.syslog.dsn,DISP=SHR    Syslog Input     */
/* //SYSLOGX   DD   DSN=your.syslogx.dsn,DISP=NEW   Optional Output  */
/* //SYSTSPRT  DD   SYSOUT=*                        TSO/REXX Output  */
/* //OUTPUT    DD   SYSOUT=*                        Matching records */
/* //DIAGMSGS  DD   SYSOUT=*                        Error messages   */
/* //SYSTSIN   DD   DUMMY                           TSO/REXX Input   */
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
/*                                                                   */
/* The SYSLOGX DD is optional and can be used to capture expanded    */
/* SYSLOG records.  Start with an LRECL=1000 and use RECFM=VB.  If   */
/* you get RC=1, it means some records were truncated.               */
/*********************************************************************/
/*                      Change Log                                   */
/*********** @REFRESH BEGIN START    2005/04/03 12:55:22 *************/
/* Standard housekeeping activities                                  */
/*********************************************************************/
 call time 'r'
 parse arg parms
 signal on syntax name trap
 signal on failure name trap
 signal on novalue name trap
 probe = 'NONE'
 vardump = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' 0 'Parms:' parms
 if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
 call modtrace 'START' 0
/*********************************************************************/
/* Set local estoeric names                                          */
/*********************************************************************/
 @vio   = 'VIO'
 @sysda = 'SYSDA'
/*********** @REFRESH END   START    2005/04/03 12:55:22 *************/
/* Set defaults                                                      */
/*********************************************************************/
 version = '1.0.0'
/*********************************************************************/
/* Default starting position of the syslog (record type location)    */
/* N, S, M, D, E, L, O, etc.  Set this to the offset of the record   */
/* type if not position 1 (SMR requires logstart=5)                  */
/*********************************************************************/
 logstart = 1
/*********************************************************************/
/* Get yesterday's julian date (in case it is needed)                */
/*********************************************************************/
 yestjul  = g2j(date('S',(date('B',date('S'),'S')-1),'B'))
/*********************************************************************/
/* Accept, validate, and print the arguments                         */
/*********************************************************************/
 parse arg parms
 if parms = '' then call rcexit 8 'No parms provided'
 parse var parms readlimit parms
 if readlimit <> '*' then
    do
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
     if datatype(readlimit,'W') = Ø then
        call rcexit 11 'READLIMIT must be numeric or "*"'
    end
 if parms = '' then call rcexit 1Ø 'No tokens provided'
 origparm = parms
 tcount = words(parms)
 say
 say execname version 'Search criteria:'
 say
 do i=1 to tcount
    select
      when pos('=',word(parms,1)) <> Ø then
           do
            parse var parms token '=' value parms
            ro = 'equals'
           end
      when pos('-',word(parms,1)) <> Ø then
           do
            parse var parms token '-' value parms
            ro = 'not equal'
           end
      otherwise call rcexit 11,
               'Parm' i 'is invalid, must be n=value or n-value'
    end
    if datatype(token,'W') <> 1 then
       do
        select
          when token = 'LPAR'   then nop
          when token = 'DATE'   then nop
          when token = 'TIME'   then nop
          when token = 'TASKID' then nop
          when token = 'MSGID'  then nop
          when token = 'CMD'    then nop
          when token = 'STRING' then nop
          otherwise call rcexit 11 'Token' i '('token') is invalid'
        end
       end
    if value = '' then
       call rcexit 11 'Value' i 'is missing'
    say 'Token' token ro 'Value' value
 end
 say
/*******************************************************************/
/* Confirm required files are available                            */
/*******************************************************************/
 call ddcheck 'OUTPUT'
 call ddcheck 'SYSLOG'
/*******************************************************************/
/* Identify the file being used                                    */
/*******************************************************************/
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
 if LRC = Ø then
    say 'SYSLOG used:' sysdsname
 else
    say 'SYSLOG used: Can not determine DSN for tape datasets'
 say
/*******************************************************************/
/* Read the original SYSLOG                                        */
/*******************************************************************/
 fcount = Ø
 call tsotrap "EXECIO" readlimit "DISKR SYSLOG (STEM SYSLOG. FINIS"
 readtime = format(time('e'),,1)
 say commafy(syslog.Ø) 'records read from SYSLOG in' readtime 'seconds'
 say
/*******************************************************************/
/* Date and Time range                                             */
/*******************************************************************/
 first = 2
 do forever
    parse var syslog.first . . . lsdate lstime .
    if substr(strip(syslog.first),logstart,1) = 'S' then
       do
        first = first + 1
        iterate
       end
    first = first + 1
    if datatype(lsdate,'W') = 1 then
       do
        say 'SYSLOG starts at:' right(date('N',lsdate,'J'),11) lstime
        leave
       end
 end
 last = syslog.Ø
 do forever
    parse var syslog.last . . . ledate letime .
    if substr(strip(syslog.last),logstart,1) = 'S' then
       do
        last = last - 1
        iterate
       end
    last = last - 1
    if datatype(ledate,'W') = 1 then
       do
        say 'SYSLOG ends at  :' right(date('N',ledate,'J'),11) letime
        leave
       end
 end
/*******************************************************************/
/* Join multi-line messages and load the SYSLOGX stem              */
/*******************************************************************/
 xcount = Ø
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
 msglist = ''
 do l=1 to syslog.Ø
/*******************************************************************/
/* Strip any leading 1s, or spaces                                 */
/*******************************************************************/
    syslog.l = strip(syslog.l,'L','1')
    syslog.l = strip(syslog.l,'L',' ')
/*******************************************************************/
/* Determine whether this is a single or multi-record message      */
/*******************************************************************/
    clrtype = substr(syslog.l,logstart,1)
    clrstyp = substr(syslog.l,logstart+1,1)
    if l < syslog.Ø then
       do
        nextlr = l + 1
        nlrtype = substr(syslog.nextlr,logstart,1)
       end
    else
        nlrtype = 'N'
/*******************************************************************/
/* Parse the 7th, 8th, and 9th tokens                              */
/*******************************************************************/
    parse var syslog.l . . . . . . token7 token8 token9 .
/*******************************************************************/
/* Check whether the task identifier is present                    */
/*******************************************************************/
    taskid = substr(syslog.l,logstart+37,8)
    if taskid = '        ' then taskid = 'N/A'
/*******************************************************************/
/* Determine how to process each record type                       */
/*******************************************************************/
    select
/*******************************************************************/
/* Single records                                                  */
/*******************************************************************/
      when clrtype = 'N' & nlrtype <> 'S' then
           do
            xcount = xcount + 1
            syslogx.xcount = syslog.l
            startx.xcount = l
            endx.xcount = l
/*******************************************************************/
/* Determine which token to use as the msgid                       */
/*******************************************************************/
            select
              when clrstyp = ' ' & taskid = 'N/A' then
                   call msgctr token7
              when clrstyp = 'C' then
                   do
                    syslogx.xcount = syslog.l
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
                msgid.xcount = 'COMMAND'
               end
          when clrstyp <> ' ' then
               call msgctr token7
          otherwise call msgctr token8
        end
       end
/*******************************************************************/
/* L records - Label line of a multiple line message               */
/*******************************************************************/
      when clrtype = 'L' then
       do
        xcount = xcount + 1
        syslogx.xcount = syslog.l
        msgid.xcount = 'LABEL'
       end
/*******************************************************************/
/* Bypassed records (L and W)                                      */
/*******************************************************************/
      when clrtype = 'W' then
       do
        xcount = xcount + 1
        syslogx.xcount = syslog.l
/*******************************************************************/
/* Determine which token to use as the msgid                       */
/*******************************************************************/
        select
          when clrstyp = ' ' & taskid = 'N/A' then
               call msgctr token7
          when clrstyp <> ' ' then
               call msgctr token7
          otherwise call msgctr token8
        end
       end
/*******************************************************************/
/* Non MLWTO multi-line records                                    */
/*******************************************************************/
      when clrtype = 'N' & nlrtype = 'S' then
       do
        xcount = xcount + 1
        parse var syslog.nextlr . remainder
        syslogx.xcount = strip(syslog.l) strip(remainder)
        startx.xcount = l
        endx.xcount = l + 1
/*******************************************************************/
/* Determine which token to use as the msgid                       */
/*******************************************************************/
        select
          when clrstyp = ' ' & taskid = 'N/A' then
               call msgctr token7
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
            when clrstyp <> ' ' then
                 call msgctr token7
            otherwise call msgctr token8
          end
         end
/*********************************************************************/
/* MLWTO 'M' records                                                 */
/*********************************************************************/
      when clrtype = 'M' then
           do
            xcount = xcount + 1
            syslogx.xcount = strip(syslog.1)
            startx.xcount = 1
            endx.xcount = 1
/*********************************************************************/
/* Determine which token to use as the msgid                         */
/*********************************************************************/
            select
              when clrstyp = ' ' & taskid = 'N/A' then
                   call msgctr token7
              when clrstyp <> ' ' then
                   call msgctr token7
              otherwise call msgctr token8
            end
           end
/*********************************************************************/
/* MLWTO 'D' records                                                 */
/*********************************************************************/
      when clrtype = 'D' then
           do
            parse var syslog.1 . . . remainder
            syslogx.xcount = syslogx.xcount strip(remainder)
           end
/*********************************************************************/
/* MLWTO 'E' records                                                 */
/*********************************************************************/
      when clrtype = 'E' then
           do
            parse var syslog.1 . . . remainder
            syslogx.xcount = syslogx.xcount strip(remainder)
            endx.xcount = 1
           end
/*********************************************************************/
/* Eliminate the 'S' continuation records                            */
/*********************************************************************/
      when clrtype = 'S' then nop
/*********************************************************************/
/* Eliminate the 'O' records                                         */
/*********************************************************************/
      when clrtype = 'O' then
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
          do
           xcount = xcount + 1
           syslogx.xcount = syslog.l
           msgid.xcount = 'LOG'
          end
/*********************************************************************/
/* Eliminate the 'X' records                                         */
/*********************************************************************/
      when clrtype = 'X' then
          do
           xcount = xcount + 1
           syslogx.xcount = syslog.l
           call msgctr token8
          end
/*********************************************************************/
/* Eliminate the '+' records                                         */
/*********************************************************************/
      when clrtype = '+' then nop
/*********************************************************************/
/* Unxpected conditions                                              */
/*********************************************************************/
      otherwise
          do
           EXITRC = 2
           say '===>' syslog.l
          end
    end
 end
/*********************************************************************/
/* Report number of expanded records                                 */
/*********************************************************************/
 say
 exptime = format((time('e') - readtime),,1)
 say commafy(xcount) 'expanded records generated in' exptime 'seconds'
 say
/*********************************************************************/
/* Load SYSLOGX if SYSLOGX DD is found                               */
/*********************************************************************/
 if listdsi("SYSLOGX" "FILE") = Ø then
    do
     say 'SYSLOGX DD found, writing' commafy(xcount) 'expanded SYSLOG',
         'records to' sysdsname
     ERC = tsotrap(1 "EXECIO * DISKW SYSLOGX (STEM SYSLOGX. FINIS")
     if ERC = 1 then
        do
         say 'LRECL='syslrecl 'is too small for longest record, some',
             'records were truncated'
        end
    end
/*********************************************************************/
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/* Search the expanded records                                      */
/*******************************************************************/
 ocount = Ø
 do i=1 to xcount
/*******************************************************************/
/* Look for the tokens                                              */
/*******************************************************************/
    parms = origparm
    do j=1 to tcount
/*******************************************************************/
/* Determine whether this is an equal or not equal token pair       */
/*******************************************************************/
       if pos('=',word(parms,1)) <> Ø then
          do
           parse var parms token '=' value parms
           select
/*******************************************************************/
/* Select all records for the specified julian date                 */
/*******************************************************************/
             when token = 'DATE' then
                  do
                   msgdate = substr(syslogx.i,logstart+19,5)
                   if value = 'TODAY' then
                      value = date('J')
                   if value = 'YESTERDAY' then
                      value = yestjul
                   if msgdate <> value then
                      leave j
                  end
/*******************************************************************/
/* Select all records for the specified time                        */
/*******************************************************************/
             when token = 'TIME' then
                  do
                   msgtime = substr(syslogx.i,logstart+25,11)
                   if abbrev(msgtime,value) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all records for the specified Message ID                  */
/*******************************************************************/
             when token = 'MSGID' then
                  do
                   if abbrev(msgid.i,value) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all records with the jobnum, console ID, etc              */
/*******************************************************************/
             when token = 'TASKID' then
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
                  do
                   taskid = substr(syslogx.i,logstart+37,11)
                   if abbrev(taskid,value) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all records for a specific LPAR (can be any substring)     */
/*******************************************************************/
             when token = 'LPAR' then
                  do
                   lparid = substr(syslogx.i,logstart+1Ø,4)
                   if pos(value,lparid) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all command records containing the text string             */
/*******************************************************************/
             when token = 'CMD' then
                  do
                   cmdrec = substr(syslogx.i,logstart,1)
                   cmdind = substr(syslogx.i,logstart+1,1)
                   if cmdrec <> 'N' then
                      leave j
                   if cmdind <> 'C' then
                      leave j
                   cmdlen = length(syslogx.i) - (logstart+56)
                   cmdtxt = substr(syslogx.i,logstart+56,cmdlen)
                   if pos(value,cmdtxt) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all records containing the text string                     */
/*******************************************************************/
             when token = 'STRING' then
                  do
                   if pos(value,syslogx.i) = Ø then
                      leave j
                  end
/*******************************************************************/
/* Select all records starting with the text string in the nth word  */
/*******************************************************************/
             when abbrev(word(syslogx.i,token),value) = Ø then
                  leave j
             otherwise nop
           end
          end
       if pos('-',word(parms,1)) <> Ø then
          do
           parse var parms token '-' value parms
           if abbrev(word(syslogx.i,token),value) = 1 then
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
             leave j
          end
/*******************************************************************/
/* Make sure the message has the number of tokens in the pair       */
/*******************************************************************/
       if words(syslogx.i) < token then
          leave j
       else
          if j < tcount then
             iterate
          else
/*******************************************************************/
/* Put the matches in the output stem                               */
/*******************************************************************/
             do
              ortype = substr(syslogx.i,logstart,1)
              select
/*******************************************************************/
/* Write 'N' records to the output stem (expanded)                  */
/*******************************************************************/
                when ortype = 'N' then
                     do
                      ocount = ocount + 1
                      output.ocount = syslogx.i
                      fcount = fcount + 1
                     end
/*******************************************************************/
/* Write 'M' records to the output stem (not expanded)              */
/*******************************************************************/
                when ortype = 'M' then
                     do
                      do k=startx.i to endx.i
                         ocount = ocount + 1
                         output.ocount = syslog.k
                      end
                      fcount = fcount + 1
                     end
                 otherwise say '===> Unexpected ORTYPE' ortype syslogx.i
              end
             end
    end
 end
/*******************************************************************/
/* Load OUTPUT                                                      */
/*******************************************************************/
 EXITRC = tsotrap(1 "EXECIO * DISKW OUTPUT (STEM OUTPUT. FINIS")
/*******************************************************************/
/* Drop the SYSLOGX. and OUTPUT. stems                              */
/*******************************************************************/
 drop syslogx. output.
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
/********************************************************************/
/* Summary                                                          */
/********************************************************************/
 select
   when fcount = Ø then
        do
         EXITRC = 4
         say fcount 'records matched search criteria, RC='EXITRC
        end
   when fcount = 1 then
        say fcount 'record matched search criteria,',
           'output is in OUTPUT DD'
   otherwise
        say commafy(fcount) 'records matched search criteria,',
           'output is in OUTPUT DD'
 end
 say
 say commafy(words(msglist)) 'unique Message ID''s found'
 say
 say
/********************************************************************/
/* Message summary                                                  */
/********************************************************************/
 say 'MSGID Counts:'
 say
 msgtotal = Ø
 do i=1 to words(msglist)
    interpret,
   'sortin.i = left(translate(word(msglist,i),''-'',''_''),12)',
             'right('word(msglist,i)',7)'
    parse var sortin.i msgid msgcnt
    msgtotal = msgtotal + msgcnt
 end
/********************************************************************/
/* Sort the MSGLIST descending by count                             */
/********************************************************************/
 call stemsort(2Ø 'SORT FIELDS=(14,7,CH,D)')
/********************************************************************/
/* Add percentages                                                  */
/********************************************************************/
 do i=1 to sortout.Ø
    parse var sortout.i msgid msgcnt .
    msgid = left(msgid,12)
    msgcnt = strip(msgcnt,'T','ØØ'x)
    msgpct = msgcnt/msgtotal*1ØØ
    msgcnt = right(commafy(msgcnt),9)
    if msgpct < 1 then
       sortin.i = msgid msgcnt '    <1%'
    else
       sortin.i = msgid msgcnt '  ' right(trunc(msgpct),3)'%'
```

```
   msgcnt.i = sortin.i
 end
/*******************************************************************/
/* Sort the MSGLIST ascending by MSGID                             */
/*******************************************************************/
 call stemsort(3Ø 'SORT FIELDS=(1,12,CH,A)')
/*******************************************************************/
/* Print the sorted MSGLIST in MSGCNT and MSGID sequence           */
/*******************************************************************/
 say
 say 'Sorted Descending by Frequency      Sorted Alpha by MSGID'
 say
 do i=1 to sortout.Ø
    say msgcnt.i '     ' strip(sortout.i,'T','ØØ'x)
 end
 say
 say 'MSGID Total:' commafy(msgtotal)
 say
/*******************************************************************/
/* Shutdown                                                        */
/*******************************************************************/
 shutdown: nop
/*******************************************************************/
/* Put unique shutdown logic before the call to stdexit            */
/*********** @REFRESH BEGIN STOP     2ØØ2/Ø8/Ø3 Ø8:42:33 *************/
/* Shutdown message and terminate                                  */
/*******************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2ØØ2/Ø8/Ø3 Ø8:42:33 *************/
/* Non-refreshable subroutines                                     */
/* MSGCTR   - Maintain counters for each unique MSGID              */
/*******************************************************************/
/* MSGCTR   - Maintain counters for each unique MSGID              */
/*******************************************************************/
 msgctr: arg msgid
/*******************************************************************/
/* Deal with imbedded '-<>', leading '+', '*', '@' and trailing ':'  */
/*******************************************************************/
         msgid = strip(translate(msgid,'_  ','-<>'))
         if left(msgid,1) = '+' then
            msgid = strip(strip(msgid,'L','+'))
         if left(msgid,1) = '*' then
            msgid = strip(strip(msgid,'L','*'))
         if left(msgid,1) = '@' then
            msgid = strip(strip(msgid,'L','@'))
         if left(reverse(msgid),1) = ':' then
            msgid = strip(strip(msgid,'T',':'))
/*******************************************************************/
/* Filter things that are not message IDs, but set a bogus msgid.x   */
/*******************************************************************/
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
         trash = 'NO'
         select
           when verify(msgid,'0123456789+','M') = 0 then trash = 'YES'
           when verify(msgid,'().,/*=?_"','M') <> 0 then trash = 'YES'
           when left(msgid,1) = '#' then trash = 'YES'
           when left(msgid,1) = '-' then trash = 'YES'
           when left(msgid,2) = '+-' then trash = 'YES'
           when left(msgid,1) = '$' & left(msgid,5) <> '$HASP' then
                trash = 'YES'
           when pos("'",msgid) <> 0 then trash = 'YES'
           when datatype(left(msgid,1)) = 'NUM' then trash = 'YES'
           when length(msgid) < 6 then trash = 'YES'
           when length(msgid) > 11 then trash = 'YES'
           otherwise nop
         end
         if trash = 'YES' then
            do
             msgid.xcount = '$TRASH$'
             return
            end
/**********************************************************************/
/* If the message is not already on the MSGLIST, add it               */
/**********************************************************************/
         if pos(msgid,msglist) = 0 then msglist = msglist msgid
/**********************************************************************/
/* Load the MSGID stem                                                */
/**********************************************************************/
         msgid.xcount = msgid
/**********************************************************************/
/* Set the MSGID VAR to 0, if this is the first occurrance            */
/**********************************************************************/
         select
           when symbol(msgid) = 'LIT' then interpret msgid '=' 0
           when symbol(msgid) = 'BAD' then return
           otherwise nop
         end
/**********************************************************************/
/* Increment the counter                                              */
/**********************************************************************/
         interpret msgid '=' msgid '+' 1
         return
/*********** @REFRESH BEGIN SUBBOX   2004/03/10 01:25:03 *************/
/* 26 Internal Subroutines provided in LOGLYZER                       */
/* Last Subroutine REFRESH was 4 Apr 2005 02:31:14                    */
/*                                                                    */
/* RCEXIT   - Exit on non-zero return codes                           */
/* TRAP     - Issue a common trap error message using rcexit          */
/* ERRMSG   - Build common error message with failing line number     */
/* STDENTRY - Standard Entry logic                                    */
/* STDEXIT  - Standard Exit logic                                     */
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine whether a required DD is allocated           */
/* DDLIST   - Returns number of DD's and populates DDLIST variable   */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* QDSN     - Make sure there are only one set of quotes             */
/* UNIQDSN  - Create a unique dataset name                           */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* TSOQUIET - Trap all output from a TSO command and ignore failures */
/* SAYDD    - Print messages to the requested DD                     */
/* JOBINFO  - Get job related data from control blocks               */
/* COMMAFY  - Add commas to a number                                 */
/* STEM     - Set a value into a stem and increment the number       */
/* STEMSORT - Sort a stem variable (caller must strip nulls)         */
/* VIODD    - EXECIO a stem into a sequential dataset                */
/* G2J      - Convert a Gregorian date to a Julian data              */
/* PTR      - Pointer to a storage location                          */
/* STG      - Return the data from a storage location                */
/* MODTRACE - Module Trace                                           */
/*********** @REFRESH END   SUBBOX   2004/03/10 01:25:03 *************/
/*********** @REFRESH BEGIN RCEXIT   2005/04/03 12:58:07 *************/
/* RCEXIT   - Exit on non-zero return codes                          */
/*-------------------------------------------------------------------*/
/* EXITRC   - Return code to exit with (if non zero)                 */
/* ZEDLMSG  - Message text for it with for non zero EXITRCs          */
/*********************************************************************/
 rcexit: parse arg EXITRC zedlmsg
         EXITRC = abs(EXITRC)
         if EXITRC <> 0 then
            do
             trace 'o'
/*********************************************************************/
/* If execution environment is ISPF then VPUT ZISPFRC                */
/*********************************************************************/
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                     address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/*********************************************************************/
/* If a message is provided, wrap it in date, time, and EXITRC       */
/*********************************************************************/
             if zedlmsg <> '' then
                do
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
```

 © 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
                call msg zedlmsg
               end
/*********************************************************************/
/* Write the contents of the Parentage Stack                         */
/*********************************************************************/
             stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/*********************************************************************/
/* Write to MSGDD if background and MSGDD exists                     */
/*********************************************************************/
             if tsoenv = 'BACK' then
               do
                if subword(zedlmsg,9,1) = msgdd then
                   do
                    say zedlmsg
                    signal shutdown
                   end
                else
                   do
                    call saydd msgdd 1 zedlmsg
                    call saydd msgdd 1 stacktitle
                   end
               end
             else
/*********************************************************************/
/* Write to the ISPF Log if foreground                               */
/*********************************************************************/
               do
                zerrlm = zedlmsg
                address ISPEXEC "LOG MSG(ISRZØØ3)"
                zerrlm = center(' 'stacktitle' ',78,'-')
                address ISPEXEC "LOG MSG(ISRZØØ3)"
               end
/*********************************************************************/
/* Unload the Parentage Stack                                        */
/*********************************************************************/
             do queued()
               pull stackinfo
               if tsoenv = 'BACK' then
                  do
                   call saydd msgdd Ø stackinfo
                  end
               else
                  do
                   zerrlm = stackinfo
                   address ISPEXEC "LOG MSG(ISRZØØ3)"
                  end
             end
/*********************************************************************/
/* Print the VARDUMP values (if present)                             */
/*********************************************************************/
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
             if vardump <> 'NONE' then
                do
                 if tsoenv = 'BACK' then
                    do
                     say
                     say 'Selected variable values:'
                     say
                     do vd=1 to words(vardump)
                        interpret "say word(vardump,vd) '='",
                                  word(vardump,vd)
                     end
                     say
                    end
                end
             else
                do
                 zerrlm = 'Selected variable values:'
                 address ISPEXEC "LOG MSG(ISRZ003)"
                 do vd=1 to words(vardump)
                    interpret "zerrlm = word(vardump,vd) '='",
                              word(vardump,vd)
                    address ISPEXEC "LOG MSG(ISRZ003)"
                 end
                end
/**********************************************************************/
/* Put a terminator in the ISPF Log for the Parentage Stack          */
/**********************************************************************/
             if tsoenv = 'FORE' then
                do
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZ003)"
                end
/**********************************************************************/
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
/**********************************************************************/
             signal shutdown
            end
         else
            return
/*********** @REFRESH END   RCEXIT   2005/04/03 12:58:07 *************/
/*********** @REFRESH BEGIN TRAP     2004/12/13 14:00:48 *************/
/* TRAP     - Issue a common trap error message using rcexit         */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                     */
/**********************************************************************/
 trap: trace 'off'
       traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       if trap = 'YES' & tsoenv = 'BACK' then
          do
           trap = 'NO'
           traplinemsg = msg
           say traplinemsg
           signal on syntax name trap
           signal on failure name trap
           signal on novalue name trap
           say
           say center(' Trace of failing instruction ',78,'-')
           trace 'i'
           interpret trapline
          end
       if trap = 'NO' & tsoenv = 'BACK' then
          do
           say center(' Trace of failing instruction ',78,'-')
           say
          end
       if tsoenv = 'FORE' then
          call rcexit 666 msg
       else
          call rcexit 666 traplinemsg
/*********** @REFRESH END   TRAP     2004/12/13 14:00:48 *************/
/*********** @REFRESH BEGIN ERRMSG   2002/08/10 16:53:04 *************/
/* ERRMSG   - Build common error message with failing line number    */
/*-------------------------------------------------------------------*/
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
/*********************************************************************/
 errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2005/01/30 07:49:49 *************/
/* STDENTRY - Standard Entry logic                                   */
/*-------------------------------------------------------------------*/
/* MSGDD    - Optional MSGDD used only in background                 */
/*********************************************************************/
 stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/*********************************************************************/
/* Start-up values                                                   */
/*********************************************************************/
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
           EXITRC = Ø
           MAXRC = Ø
           trap = 'YES'
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           headoff = 'NO'
           hcreator = 'NO'
           keepstack = 'NO'
           lpar = mvsvar('SYSNAME')
           jobname = mvsvar('SYMDEF','JOBNAME')
           zedlmsg = 'Default shutdown message'
           lower = xrange('a','z')
           upper = xrange('A','Z')
/*********************************************************************/
/* Determine environment                                             */
/*********************************************************************/
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
               signal off failure
              "ISPQRY"
               ISPRC = RC
               if ISPRC = Ø then
                  do
                   ispfenv = 'YES'
/*********************************************************************/
/* Check if HEADING ISPF table exists already, if so set HEADOFF=YES */
/*********************************************************************/
                   call ispwrap "VGET (ZSCREEN)"
                   if tsoenv = 'BACK' then
                      htable = jobinfo(1)||jobinfo(2)
                   else
                      htable = userid()||zscreen
                   TBCRC = ispwrap(8 "TBCREATE" htable "KEYS(HEAD)")
                   if TBCRC = Ø then
                      do
                       headoff = 'NO'
                       hcreator = 'YES'
                      end
                   else
                      do
                       headoff = 'YES'
                      end
                  end
               signal on failure name trap
              end
/*********************************************************************/
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

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

```
                end
            end
/*****************************************************************/
/* If foreground, save ZFKA and turn off the FKA display          */
/*****************************************************************/
           else
              do
               fkaset = 'OFF'
               call ispwrap "VGET (ZFKA) PROFILE"
               if zfka <> 'OFF' & tsoenv = 'FORE' then
                  do
                   fkaset = zfka
                   fkacmd = 'FKA OFF'
                  call ispwrap "CONTROL DISPLAY SAVE"
                  call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKACMD)"
                  call ispwrap "CONTROL DISPLAY RESTORE"
                  end
              end
/*****************************************************************/
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/*********** @REFRESH END   STDENTRY 2005/01/30 07:49:49 *************/
/*********** @REFRESH BEGIN STDEXIT  2004/08/02 06:06:40 *************/
/* STDEXIT  - Standard Exit logic                                 */
/*---------------------------------------------------------------*/
/* ENDTIME  - Elapsed time                                        */
/* Note: Caller must set KEEPSTACK if the stack is valid          */
/*****************************************************************/
 stdexit: module = 'STDEXIT'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
/*****************************************************************/
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC       */
/*****************************************************************/
          EXITRC = max(EXITRC,MAXRC)
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' & sysvar('SYSNEST') = 'NO' &,
             headoff = 'NO' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
/*****************************************************************/
/* Make sure this isn't a MSGDD missing error then log to MSGDD   */
/*****************************************************************/
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
             if msgdd <> '' & subword(zedlmsg,9,1) <> msgdd then
                do
                 call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                 call saydd msgdd Ø endmsg
                end
            end
/********************************************************************/
/* If foreground, reset the FKA if necessary                        */
/********************************************************************/
         else
            do
             if fkaset <> 'OFF' then
                do
                 fkafix = 'FKA'
                 call ispwrap "CONTROL DISPLAY SAVE"
                 call ispwrap "DISPLAY PANEL(ISPBLANK) COMMAND(FKAFIX)"
                 if fkaset = 'SHORT' then
                    call ispwrap "DISPLAY PANEL(ISPBLANK)",
                                 "COMMAND(FKAFIX)"
                 call ispwrap "CONTROL DISPLAY RESTORE"
                end
            end
/********************************************************************/
/* Clean up the temporary HEADING table                             */
/********************************************************************/
         if ispfenv = 'YES' & hcreator = 'YES' then
            call ispwrap "TBEND" htable
/********************************************************************/
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there    */
/********************************************************************/
         call modtrace 'STOP' sigl
         if queued() > Ø then pull . . module . sigl . sparms
         if queued() > Ø then pull . . module . sigl . sparms
         if tsoenv = 'FORE' & queued() > Ø & keepstack = 'NO' then
            pull . . module . sigl . sparms
/********************************************************************/
/* if the Parentage Stack is not empty, display its contents        */
/********************************************************************/
         if queued() > Ø & keepstack = 'NO' then
            do
             say queued() 'Leftover Parentage Stack Entries:'
             say
             do queued()
                pull stackundo
                say stackundo
             end
             EXITRC = 1
            end
/********************************************************************/
/* Exit                                                             */
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
/*********************************************************************/
          exit(EXITRC)
/*********** @REFRESH END   STDEXIT  2004/08/02 06:06:40 *************/
/*********** @REFRESH BEGIN MSG      2002/09/11 01:35:53 *************/
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/*-------------------------------------------------------------------*/
/* ZEDLMSG  - The long message variable                              */
/*********************************************************************/
 msg: module = 'MSG'
      parse arg zedlmsg
      if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/*********************************************************************/
/* If this is background or OMVS use SAY                             */
/*********************************************************************/
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/*********************************************************************/
/* If this is foreground and ISPF is available, use SETMSG           */
/*********************************************************************/
         do
          if ispfenv = 'YES' then
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
             address ISPEXEC "SETMSG MSG(ISRZ000)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2002/09/11 01:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2004/11/09 22:48:36 *************/
/* DDCHECK  - Determine whether a required DD is allocated           */
/*-------------------------------------------------------------------*/
/* DD       - DDNAME to confirm                                      */
/*********************************************************************/
 ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
/*********************************************************************/
/* Allow sysreason=3 & 22 to verify SYSOUT and tape DD statements    */
/*********************************************************************/
          if LRC <> Ø & sysreason <> 3 & sysreason <> 22 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/*********** @REFRESH END   DDCHECK  2ØØ4/11/Ø9 22:48:36 *************/
/*********** @REFRESH BEGIN DDLIST   2ØØ2/12/15 Ø4:54:32 *************/
/* DDLIST   - Returns number of DDs and populates DDLIST variable    */
/*-------------------------------------------------------------------*/
/* N/A      - None                                                   */
/*********************************************************************/
 ddlist: module = 'DDLIST'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         call outtrap 'lines.'
         address TSO "LISTALC STATUS"
         call outtrap 'off'
         ddnum = Ø
/*********************************************************************/
/* Parse out the DDNAMEs and concatenate into a list                 */
/*********************************************************************/
         ddlist = ''
         do ddl=1 to lines.Ø
            if words(lines.ddl) = 2 then
               do
                parse upper var lines.ddl ddname .
                ddlist = ddlist ddname
                ddnum = ddnum + 1
               end
            else
               do
                iterate
               end
         end
/*********************************************************************/
/* Return the number of DDs                                          */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
        call modtrace 'STOP' sigl
        interpret 'trace' tracelvl
        return ddnum
/*********** @REFRESH END   DDLIST   2002/12/15 04:54:32 *************/
/*********** @REFRESH BEGIN DDDSNS   2002/09/11 00:37:36 *************/
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*-------------------------------------------------------------------*/
/* TARGDD   - DD to return DSNs for                                  */
/*********************************************************************/
 dddsns: module = 'DDDSNS'
        if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
        parse arg sparms
        push trace() time('L') module 'From:' sigl 'Parms:' sparms
        call modtrace 'START' sigl
        arg targdd
        if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
        x = outtrap('lines.')
        address TSO "LISTALC STATUS"
        dsnnum = 0
        ddname = '$DDNAME$'
/*********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/*********************************************************************/
        do ddd=1 to lines.0
           select
              when words(lines.ddd) = 1 & targdd = ddname &,
                   lines.ddd <> 'KEEP' then
                   dddsns = dddsns strip(lines.ddd)
              when words(lines.ddd) = 1 & strip(lines.ddd),
                   <> 'KEEP' then
                   dddsn.ddd = strip(lines.ddd)
              when words(lines.ddd) = 2 then
                   do
                    parse upper var lines.ddd ddname .
                    if targdd = ddname then
                       do
                        fdsn = ddd - 1
                        dddsns = lines.fdsn
                       end
                   end
              otherwise iterate
           end
        end
/*********************************************************************/
/* Get the last DD                                                   */
/*********************************************************************/
        ddnum = ddlist()
```


© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

```
         lastdd = word(ddlist,ddnum)
/*********************************************************************/
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
/*********************************************************************/
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
/*********************************************************************/
/* Return the number of DSNs in the DD                               */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/*********** @REFRESH END   DDDSNS   2002/09/11 00:37:36 *************/
/*********** @REFRESH BEGIN QDSN     2002/09/11 01:15:23 *************/
/* QDSN     - Make sure there are only one set of quotes             */
/*-------------------------------------------------------------------*/
/* QDSN     - The DSN                                                */
/*********************************************************************/
 qdsn: module = 'QDSN'
       if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
       parse arg sparms
       push trace() time('L') module 'From:' sigl 'Parms:' sparms
       call modtrace 'START' sigl
       parse arg qdsn
       qdsn = "'"strip(qdsn,"B","'")"'"
       pull tracelvl . module . sigl . sparms
       call modtrace 'STOP' sigl
       interpret 'trace' tracelvl
       return qdsn
/*********** @REFRESH END   QDSN     2002/09/11 01:15:23 *************/
/*********** @REFRESH BEGIN UNIQDSN  2004/09/01 18:03:04 *************/
/* UNIQDSN  - Create a unique dataset name                           */
/*-------------------------------------------------------------------*/
/* PARM     - N/A                                                    */
/*********************************************************************/
 uniqdsn: module = 'UNIQDSN'
          if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
/*********************************************************************/
/* Compose a DSN using userid, exec name, job number, date and time  */
/*********************************************************************/
          jnum = jobinfo(1) || jobinfo(2)
          udate = 'D'space(translate(date('O'),'','/'),0)
          utime = 'T'left(space(translate(time('L'),'',':.'),0),7)
```

© 2005. Reproduction prohibited. Please inform Xephon of any infringement.

```
         uniqdsn = userid()'.'execname'.'jnum'.'udate'.'utime
         if sysdsn(qdsn(uniqdsn)) = 'OK' then
            do
/*********************************************************************/
/* Wait 1 seconds to ensure a unique dataset (necessary on z990)     */
/*********************************************************************/
             RC = syscalls('ON')
             address SYSCALL "SLEEP 1"
             RC = syscalls('OFF')
             uniqdsn = uniqdsn()
            end
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return uniqdsn
/*********** @REFRESH END   UNIQDSN  2004/09/01 18:03:04 *************/
```

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

Robert Zenuk
Systems Programmer (USA)

© Xephon 2005

© 2005. Xephon USA telephone (214) 340 5690, fax (214) 341 7081.

MVS news

Computer Associates has announced new versions of Unicenter CA-Spool Print Management r11 (CA-Spool), Unicenter CA-View r11 (CA-View), and Unicenter CA-Deliver r11 (CA-Deliver).

CA-Spool provides management of all printing and spooling tasks throughout the enterprise. It features a Web interface and enhancements to critical processing points. Mainframe reports can be transformed to PDF, HTML, and RTF formats.

CA-Deliver is an online report management system that automates distribution, tracking, and printing. CA-View is an automated system that allows immediate online viewing of mainframe output.

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

***

Informatica has announced the latest version of PowerCenter for Mainframe, which provides a single data integration platform to help organizations access, transform, and integrate data from a large variety of systems and deliver that information to other transactional systems, real-time business processes, and people.

PowerCenter for Mainframe provides a unified data integration platform that leverages the business-critical aspects of mainframe computing while providing the same time-to-value and cost-of-ownership advantages that PowerCenter brings to non-mainframe environments, the company claims. These include PowerCenter's metadata-driven architecture for defining integration tasks once and deploying them anywhere on or off the mainframe, and Informatica PowerExchange's ability to access real-time, changed-only, and batch data.

For further information contact:
URL: www.informatica.com/products/powercenter/default.htm.

***

IBM has announced an expanded suite of new Tivoli software products that automate real-time systems management and monitoring capabilities for eServer zSeries. The Tivoli OMEGAMON software suite allows organizations to proactively detect, isolate, and repair application performance problems anywhere within their IT infrastructure.

This enhanced product suite integrates Candle's technology, creating the Tivoli systems management solution for zSeries customers.

The IBM Tivoli OMEGAMON XE solution includes products that monitor and manage zSeries operating systems and subsystems including z/OS Unix System Services and Parallel Sysplex, z/VM, Unix System Services, CICS, DB2, IMS, zSeries networks, storage, WebSphere Application Server (WAS), WebSphere Integration Brokers, and WebSphere MQ.

For further information contact:
URL: www.ibm.com/ondemand.

***

StreamFoundry has announced its IT application suite, Central Management System (CMS) for problem, change, and request management.

The product reduces the overall processing requirements of the network, server, and database environments.

For further information contact:
URL: www.streamfoundry.com/product.html.