211

MVS

April 2004

In this issue

3 Easy dynamic allocation!

9 Porting a Java Web application to z/OS

24 zSeries CPC capping

28 Keeping track of non-reusable ASIDs

48 REXX routine to count lines of COBOL code – part 2

62 JCL tips – part 1

72 Defining page datasets for a new partition without STEPCAT

75 MVS news

update

MVS Update

Published by

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

North American office

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

Subscriptions and back-issues

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

MVS Update on-line

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

Editor

Trevor Eddolls

Disclaimer

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

Contributions

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

Printed in England.

Easy dynamic allocation!

INTRODUCTION

The use of the dynamic allocation SVC (99) is an arduous and error prone process. To help you with SVC 99, there is an IBM utility, BPXWDYN, which is an undocumented/semi-documented service used by USS.

BPXWDYN is a text interface to a subset of the SVC 99 (dynamic allocation) and SVC 109 (dynamic output) services. BPXWDYN supports dataset allocation, unallocation, concatenation, and adding and deleting output descriptors. BPXWDYN covers only a subset of SVC 99 parameters, but probably supports everything you need including USS path allocation.

The primary intent of BPXWDYN is to make dynamic allocation and dynamic output services easily accessible to programs running outside of a TSO environment; however, it also functions in a TSO environment.

BPXWDYN documentation is available at ftp://ftp.software.ibm.com/s390/zos/tools/bpxwdyn/bpxwdyn.html.

This article describes how to use BPXWDYN with REXX and Assembler.

BPXWDYN SERVICE

BPXWDYN calling convention

BPXWDYN is an IBM service located in SYS1.LINKLIB. It is designed to be called directly from REXX, but may also be called by any program using a parameter list.

Several parameter list formats are supported:

- REXX external function parameter list.
- Conventional MVS variable-length parameter string.

REXX external function parameter list

The REXX external function parameter list allows REXX programs to call the BPXWDYN program as a function or subroutine.

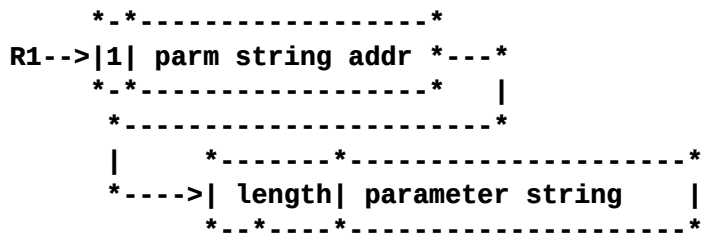
It must be called with a single string parameter:

```
if BPXWDYN("alloc dd(sysin) da('my.dataset') shr")<>0 then
   call allocfailed
```

Conventional MVS variable-length parameter string

The conventional MVS variable-length parameter string is the same parameter list as the one generated by ADDRESS LINKMVS with one parameter and JCL with EXEC PGM=xxxxxxxx,PARM=.

This parameter list form is simple to use by any Assembler program:



Note that this is a single item variable-length parameter list. The high bit is on in the parameter address word and length is a half word.

BPXWDYN syntax

BPXWDYN accepts four request types:

- ALLOC – for dynamic allocation.
- FREE – for dynamic unallocation of a DD or to free an output descriptor.
- CONCAT – to concatenate several DDnames.
- OUTDES – to create an output descriptor.

For each request type, it is possible to specify additional parameters (keys). There are too many options to be included in

this article; you should read the BPXWDYN documentation page on the IBM FTP server to get more details.

ALLOC sample:

```
CALL BPXWDYN("ALLOC DD(DD1) DA(MYUSER.BPXWDYN) NEW CATALOG
                    UNIT(339Ø) CYL SPACE(1,1)
                    DSORG(PS) RECFM(F,B) LRECL(8Ø) BLKSIZE(2344Ø)")
SAY "ALLOC   - RC = " RESULT
```

ALLOC HFS sample:

```
CALL BPXWDYN("ALLOC DD(DD3) PATH('/tmp/bpxwdyn.txt')
                    PATHOPTS(OWRONLY,OCREAT) PATHMODE(SIRWXU,SIRGRP)
                    PATHDISP(KEEP,DELETE) msg(wtp)")
SAY "ALLOC   - RC = " RESULT
```

FREE sample:

```
CALL BPXWDYN("FREE  DD(DD1)")
SAY "FREE    - RC = " RESULT
```

CONCAT sample:

```
CALL BPXWDYN("ALLOC DD(DD1) DA(MYUSER.MCAT.DLB$Ø1) SHR")
SAY "ALLOC   - RC = " RESULT

CALL BPXWDYN("ALLOC DD(DD2) DA(MYUSER.MCAT.DLB$Ø2) SHR")
SAY "ALLOC   - RC = " RESULT

CALL BPXWDYN("CONCAT DDLIST(DD1,DD2) MSG(WTP)")
SAY "CONCAT  - RC = " RESULT
```

OUTDES sample:

```
CALL BPXWDYN("OUTDES(MYDESC) COPIES(4) DEPT(MYDEPT) DEST(MYDEST)")
SAY "OUTDES  - RC = " RESULT
```

BPXWDYN return codes

When BPXWDYN is called as a REXX function or subroutine, the return code can be accessed in RESULT or as the value of the function.

When called as a program, the return code is available in R15.

BPXWDYN returns the following codes:

- 0 – successful.

- 20 – invalid parameter list.
- -21 to -9999 – key error.
- -100*nn* – message processing error. IEFDB476 returned code *nn*.
- >0 – dynamic allocation or dynamic output error codes.

For more information about error codes, you should see the BPXWDYN documentation page on the IBM FTP server.

SAMPLE REXX PROGRAM USING BPXWDYN SERVICES

```
/* REXX */
 CALL BPXWDYN("ALLOC DD(DD1) DA(MYUSER.BPXWDYN) NEW CATALOG
                    UNIT(339Ø) CYL SPACE(1,1)
                    DSORG(PS) RECFM(F,B) LRECL(8Ø) BLKSIZE(2344Ø)")
 SAY "ALLOC   - RC = " RESULT
 LINEO.Ø = 2
 LINEO.1 = "TEST DATA LINE 1"
 LINEO.2 = "TEST DATA LINE 2"
 "EXECIO * DISKW DD1 (STEM LINEO."
 "EXECIO Ø DISKW DD1 (FINIS"
 CALL BPXWDYN("FREE  DD(DD1)")
 SAY "FREE    - RC = " RESULT
 CALL BPXWDYN("ALLOC DD(DD2) DA(MYUSER.BPXWDYN) OLD  DELETE")
 SAY "ALLOC   - RC = " RESULT
 CALL BPXWDYN("FREE  DD(DD2)")
 SAY "FREE    - RC = " RESULT
 CALL BPXWDYN("OUTDES(MYDESC) COPIES(4) DEPT(MYDEPT) DEST(MYDEST)")
 SAY "OUTDES  - RC = " RESULT
 CALL BPXWDYN("ALLOC DD(DD3) SYSOUT(A) OUTDES(MYDESC)")
 SAY "ALLOC   - RC = " RESULT
 LINEO.Ø = 2
 LINEO.1 = "TEST DATA LINE 1"
 LINEO.2 = "TEST DATA LINE 2"
 "EXECIO * DISKW DD3 (STEM LINEO."
 "EXECIO Ø DISKW DD3 (FINIS"
 CALL BPXWDYN("FREE  DD(DD3)")
 SAY "FREE    - RC = " RESULT
 CALL BPXWDYN("ALLOC DD(DD1) DA(MYUSER.MCAT.LØ1) SHR")
 SAY "ALLOC   - RC = " RESULT
 CALL BPXWDYN("ALLOC DD(DD2) DA(MYUSER.MCAT.LØ2) SHR")
 SAY "ALLOC   - RC = " RESULT
 CALL BPXWDYN("CONCAT DDLIST(DD1,DD2) MSG(WTP)")
 SAY "CONCAT  - RC = " RESULT
 CALL BPXWDYN("FREE  DD(DD1)")
```

```
SAY "FREE    - RC = " RESULT
CALL BPXWDYN("FREE  DD(DD2)")
SAY "FREE    - RC = " RESULT
CALL BPXWDYN("ALLOC DD(DD3) PATH('/tmp/bpxwdyn.txt')
                    PATHOPTS(OWRONLY,OCREAT) PATHMODE(SIRWXU,SIRGRP)
                    PATHDISP(KEEP,DELETE) msg(wtp)")
SAY "ALLOC   - RC = " RESULT
LINEO.Ø = 2
LINEO.1 = "TEST DATA LINE 1"
LINEO.2 = "TEST DATA LINE 2"
"EXECIO * DISKW DD3 (STEM LINEO."
"EXECIO Ø DISKW DD3 (FINIS"
CALL BPXWDYN("FREE  DD(DD3)")
SAY "FREE    - RC = " RESULT
```

SAMPLE ASSEMBLER PROGRAM USING BPXWDYN SERVICES

```
BPXWDYSO CSECT
BPXWDYSO AMODE 31
BPXWDYSO RMODE ANY
*
         SAVE  (14,12)
         BASR  R12,Ø
         USING *,R12                    R12 = BASE REGISTER
*
         GETMAIN R,LV=WORKL
*
         ST    R1,8(R13)
         ST    R13,4(R1)
         LR    R13,R1
         USING WORK,R13
*
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø'IN BPXWDYSO ROUTINE'
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
*
         LOAD  EP=BPXWDYN
         LTR   R15,RØ
         BZ    LOAD_ERROR
*
         OI    PARMLISTa,X'8Ø'       END OF PARMLIST
         LA    R1,PARMLISTa          LOAD PTR TO PARMLIST
         BALR  R14,R15               CALL BPXWDYN
         LTR   R15,R15
         BNZ   ALLOC_ERROR
*
```

```
         B     RETURN
*
*        LINK    EP=SHOWREGS
*
LOAD_ERROR EQU *
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø'BPXWDYSO - ERROR DURING LOAD'
         B     ISSUE_MSG
ALLOC_ERROR EQU *
         MVC   WTOA(WTOL),WTOLIST
         MVC   WTOM,=CL8Ø'BPXWDYSO - ERROR DURING ALLOC'
*
ISSUE_MSG EQU *
*
         L     R2,=A(WTOMLEN)
         STH   R2,WTOML
         LA    R2,WTOMSG
         WTO   TEXT=(R2),MF=(E,WTOLIST)
*
RETURN   L     R13,4(R13)              RESTORE R13
         L     R1,8(R13)
         FREEMAIN R,LV=WORKL,A=(R1)
         L     R14,12(R13)
         LM    RØ,R12,2Ø(R13)
         SR    R15,R15                 SET UP RC
         BSM   Ø,R14                   RETURN TO MVS AND USE RC=R15
*
WTOLIST  WTO   TEXT=,ROUTCDE=11,MF=L
WTOL     EQU   *-WTOLIST
*
PARMLISTa DC   AL4(LENGTH)
LENGTH   DC    AL2(TEXTLEN)
TEXT     DC    C'ALLOC FI(DD2) DSN(MYUSER.TEST) NEW CATALOG '
         DC    C'UNIT(339Ø) CYL SPACE(5,1) '
         DC    C'DSORG(PS) RECFM(F,B) LRECL(8Ø) BLKSIZE(2344Ø)'
TEXTLEN  EQU   *-TEXT
*
WORK     DSECT
SAVEAREA DS    18F
WTOA     DS    CL(WTOL)
*
WTOMSG   DS    ØF
WTOML    DS    H
WTOM     DS    CL8Ø
WTOMLEN  EQU   *-WTOM
*
WORKL    EQU   *-WORK
*
         REGISTER
```

```
*
          END
```


Porting a Java Web application to z/OS

INTRODUCTION

Scaling is the ability of a system to handle increasing demands at an acceptable performance level. When the number of clients is not very large, PC platforms may be a satisfactory solution. However, when the number of clients is growing rapidly, a possible solution is to port the same application to z/OS, which is intended for such a load.

jPOS is a Java-based, production grade, ISO 8583 library/framework that can be used to implement financial interchanges, protocol converters (for example from HYPERCOM ISO 8583 to SPDH and *vice versa*, etc), payment gateways, credit card verification clients and servers (from merchant, issuer, or acquirer point of view), etc. In other words, jPOS is a Java-based financial transaction library/framework that can be customized and extended in order to implement any particular financial interchange. You can use http://www.jpos.org/ to get more information about jPOS.

Our aim was to port a Java Web application (jPOS-based application) developed for PC platforms, called TMS (Terminal Management System), onto a z/OS platform. TMS is a modular software system for EFT/POS (Electronic Funds Transfer with Point Of Sale) network management and POS financial transactions management. TMS is a system through which EFT/POS devices connect to a bank authorization system or to many bank authorization systems. TMS also initializes EFT/POS devices in such a way that they are able to accept national cards and cards compliant with international technical standards, issued by any bank.

The purpose of this article is to provide all the practical information necessary to successfully port a jPOS-based application to z/OS. Particularly, we will present our experience with an IBM z/800 mainframe model 0A2 with two processors. The system has 8GB RAM memory. We assume that WebSphere Application Server (WAS) Version 4.0.1 and DB2 Version 7.1 for z/OS (PTF level 502) are already installed.

TMS DESCRIPTION

TMS is a modular and expandable system. The basic features and functions of TMS are:

- A simple graphical user interface.
- Management of EFT/POS networks with PIN-PAD or without PIN-PAD. A PIN-PAD is a small keyboard that contains numeric keys and it is used to provide better security. PIN is an acronym for a Personal Identification Number, which is entered into the keyboard pad to verify account information for a transaction in a payment system.
- ISO 8583 (1987) compliant.
- Credit, debit, and loyalty transactions support.
- Connection to concentrators by using HTTP.
- Floor limits support.
- Downloading of BIN (Bank Identification Number) tables and EFT/POS parameters.
- Routing of financial transactions.
- Hot-list management.
- Management of cards with a negative balance.
- PAN (Personal Account Number) control (Luhn formula).
- Connection to a host by using TCP/IP protocol.
- Centralized management of all POS parameters in order to

control all POS devices systematically.

- Multi-banking.
- Multi-merchant.
- Integration with Web CMS (Card Management System).
- Mobile recharge.

PROBLEMS

During the process of porting the TMS (jPOS-based application) to z/OS we met the following problems:

- General problems in Java code on z/OS. For example, ternary if-else operator does not always work properly.
- The z/OS platform represents character strings in EBCDIC format and it is really the biggest challenge when writing platform-independent Java applications.
- How to prepare a Java jPOS-based Web application for deployment on z/OS.
- How to deploy a Java jPOS-based Web application on z/OS.
- How to run a Java jPOS-based Web application on z/OS.

SOLUTIONS

Solutions for general problems with Java code on z/OS

General problems in Java code on z/OS can be solved easily. For example:

- Ternary *if-else* operator
- Boolean-exp? true : false

Boolean values of true and false do not always work properly in a z/OS environment, so they have to be replaced with the following *if-else* statements:

```
if (boolean-exp)
 return true;
else
 return false;
```

After the replacement, the code works correctly.

Solutions for problems in porting jPOS to z/OS

The class ISO87Bpackager is used for packaging ISO 8583 messages in the TMS application. Most platforms to which Java has been ported are ASCII based. Bearing in mind that the z/OS platform is EBCDIC based, whenever it is necessary to get the value of a field that is represented as an ASCII value in an ISO 8583 message, it is necessary to convert that value to EBCDIC and to create a string under that array of bytes. It can be done in the following way:

```
ISOPackager packager = new ISO87BPackager();
ISOMsg isoMsg = new ISOMsg();
isoMsg.setPackager(packager);
isoMsg.unpack(isobytes);

String fieldValue = (String) isoMsg.getValue(fieldNumber);
byte[] ebcdic = ISOUtil.asciiToEbcdic(fieldValue.getBytes());
String fieldValueOnHost = new String(ebcdic);
```

The *isobytes* is an array of bytes representing an ISO 8583 message. ISO 8583 messages are composed of fields. The number of fields is variable for different types of ISO 8583 messages. The *fieldValue* is an ASCII value, which you want to convert into the corresponding EBCDIC value. The *fieldNumber* is the number of the required field. For example, if you want the Merchant ID field, then *fieldNumber* = 42.

Similarly, whenever it is necessary to set a value for a field that is represented as an ASCII value in an ISO 8583 message, it is first necessary to convert the string to ASCII and after that to set the value in the ISO 8583 message. It can be done in the following way:

```
ISOPackager packager = new ISO87BPackager();
ISOMsg isoMsg = new ISOMsg();
isoMsg.setPackager(packager);
isoMsg.unpack(isobytes);
```

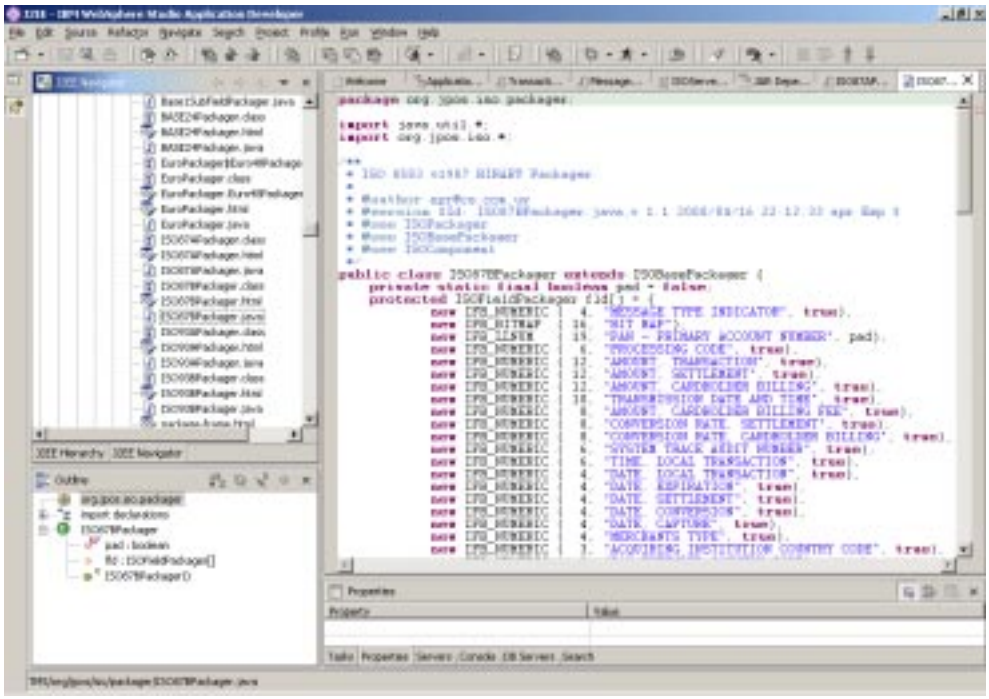


Figure 1: Studio Application Developer for Windows

```
String ascii = ISOUtil.ebcdicToAscii(fieldValue.getBytes());
```

Figure 2: The window for selecting an export destination

Figure 3: EAR file resources and name

```
isoMsg.set(new ISOField(fieldNumber, ascii));
```

As in the previous example, the *isobytes* is an array of bytes, which represents an ISO 8583 message, the *fieldValue* is an EBCDIC value that we want to convert into an ASCII value, and the *fieldNumber* is the number of the required field.

Preparing a Java jPOS-based Web application for deployment on z/OS

A Java jPOS-based Web application can be prepared for deployment on z/OS platform by using any PC with IBM WebSphere Studio Application Developer (we worked with Version 5.0.0 for Windows) installed. First, in WebSphere Studio create a new Enterprise Application Project. We called it WebTms. Then it is necessary to import Java source classes from a file system as shown in Figure 1 (**Select File/Import/File system** and enter the name of the directory where the Java classes are located). After that you

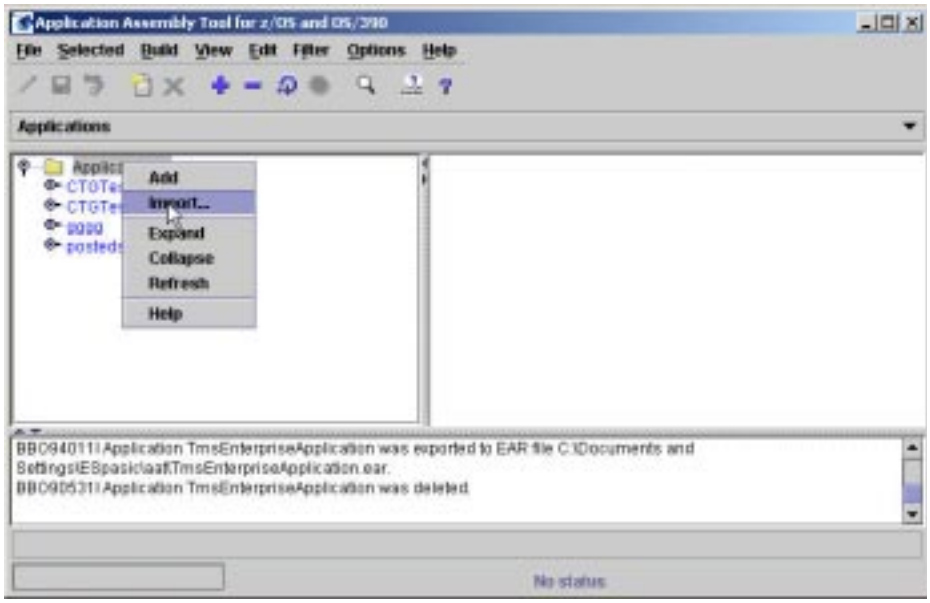


Figure 4: The Application Assembly Tool main window

should click on the button *J2EE Navigator* (below the window on the upper-left side of the screen) in order to see the tree of Java classes.

It is necessary to export the Enterprise Application project into an EAR (Enterprise Archive) file in order to deploy a Web application on WebSphere Application Server on z/OS. So, the next step is to create an EAR file. Select **File/Export**. The window is shown in Figure 2.

Select *EAR file* option, click on the *Next* button and you will see the window shown in Figure 3. Enter the resource name and the

Figure 5: Importing EAR file

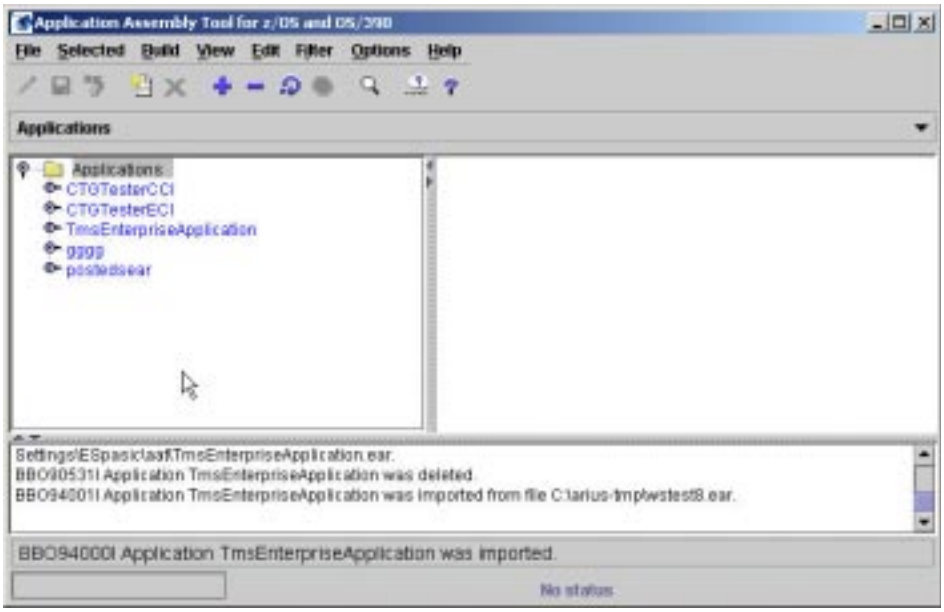


Figure 6: Application Assembly Tool after import

location to which you want to export the EAR file.

Click on the button *Finish* and the EAR file will be generated.

Using the Application Assembly Tool for z/OS and OS/390

We used the Application Assembly Tool for z/OS and OS/390 in order to deploy the TMS (jPOS-based application). The main

Figure 7: Deploying Tms Enterprise Application

Figure 8: Exporting Tms Enterprise Application

window of the Application Assembly Tool for z/OS and OS/390 is shown in Figure 4.

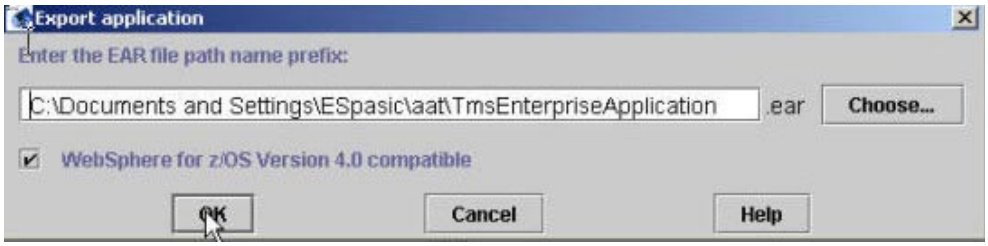


Figure 9: Entering EAR file path name

Click on *Applications* and press the right mouse button. Then click on *Import*. You will see the window shown in Figure 5.

Enter the EAR file path name and click the *OK* button. After that the EAR file will be imported. A successful import is shown in Figure 6.

The next step is validation of the application, which is imported. Select *TmsEnterpriseApplication* and press the right mouse button. Then click on *Validate* and the application will be validated.

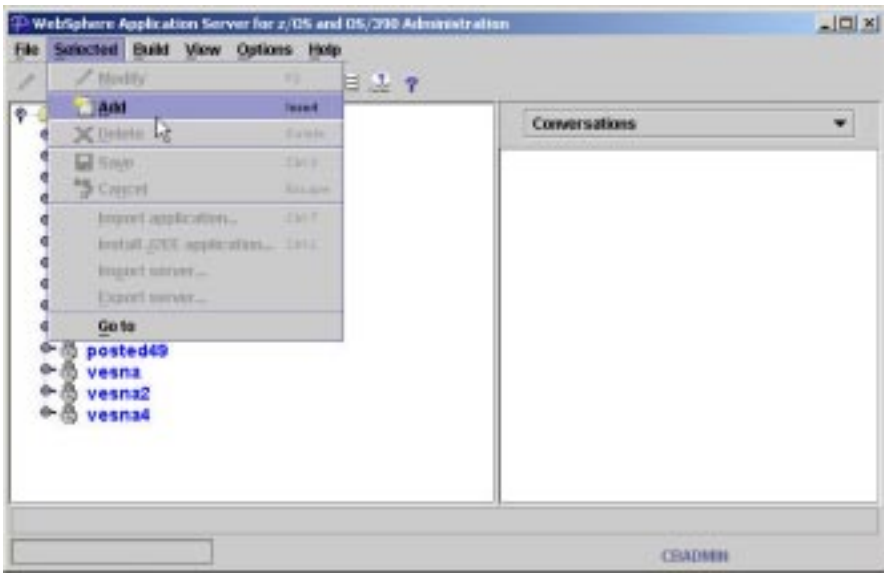


Figure 10: Adding EAR file

The next step is the application deployment in the Application Assembly Tool. Click on *TmsEnterpriseApplication* and press the right mouse button. Then click on *Deploy* as shown in Figure 7.

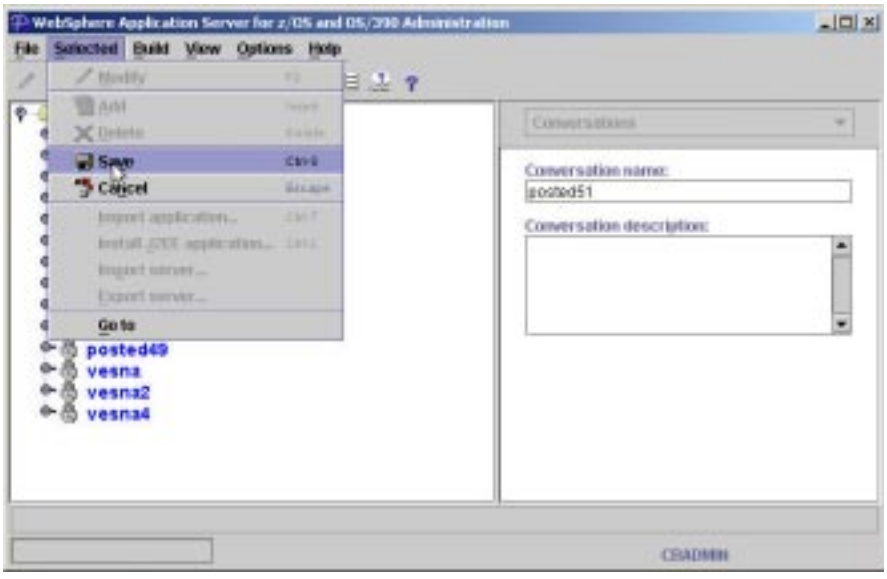


Figure 11: Saving EAR file

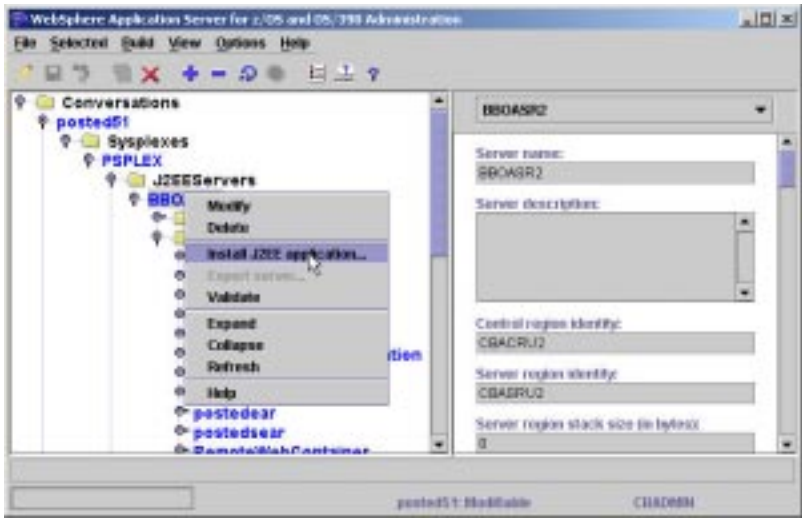


Figure 12: Installing J2EE application on z/OS

The next step is the application export from the Application Assembly Tool. Click on *TmsEnterpriseApplication* and press the right mouse button. Then click on *Export* as shown in Figure 8.

Enter the full path of EAR file for Tms Enterprise Application as shown in Figure 9.

The EAR file prepared using the Application Assembly Tool for z/OS and OS/390 is now ready for deployment on WebSphere Application Server for z/OS and OS/390. This process will now be described in detail.

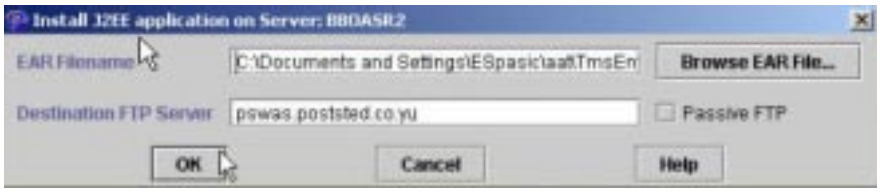


Figure 13: Entering EAR file name and destination server

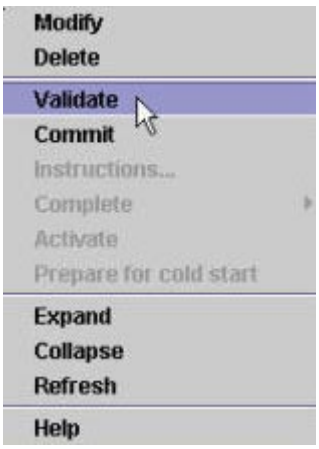


Figure 14: Validating Tms Enterprise Application

Click on the *Selected* option in the main menu bar and then click on the *Add* option as shown in Figure 10.

For *Conversion name* enter some value (for example *posted51*)

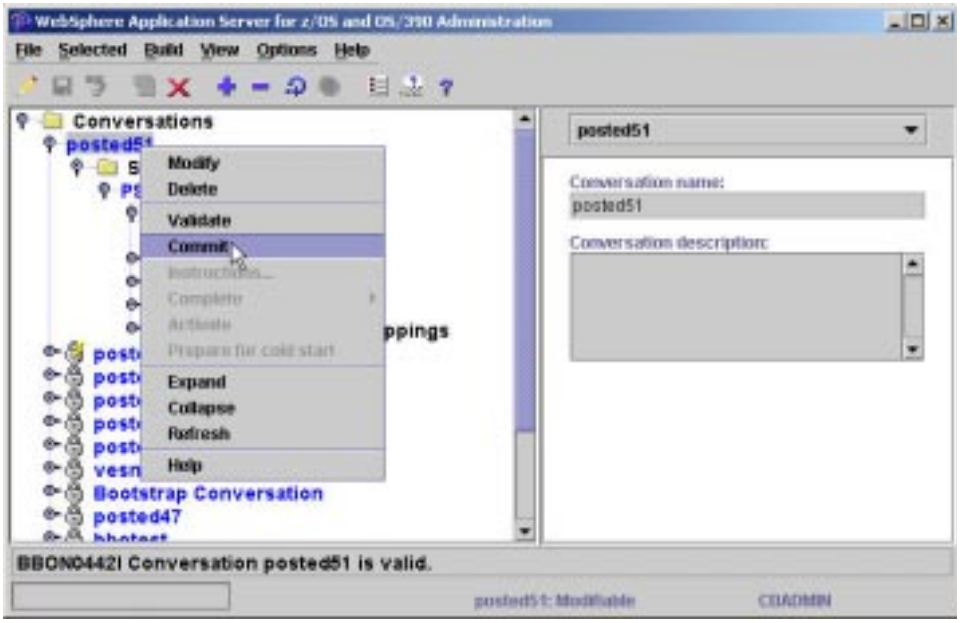


Figure 15: Committing Tms Enterprise Application

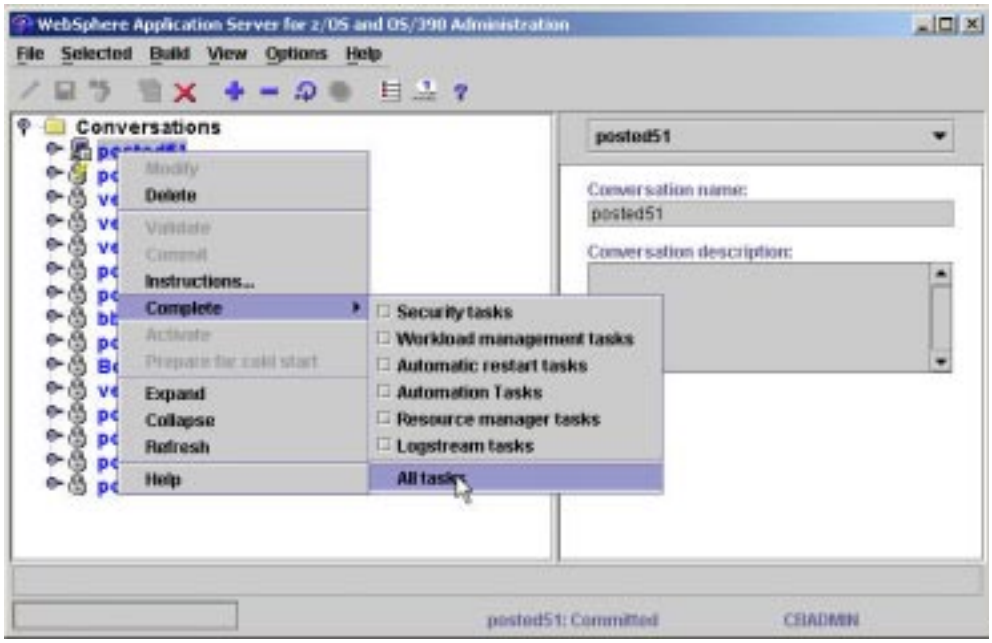


Figure 16: Completing all tasks

on the right side of the screen as shown in Figure 11.

Click on the *Selected* option in the main menu bar and then click

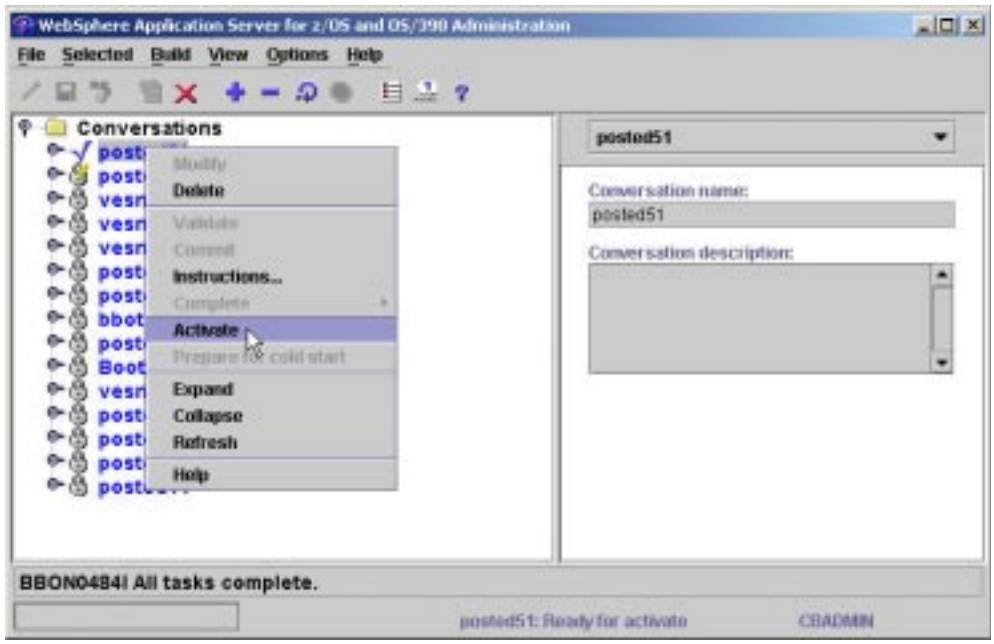


Figure 17: Activating Tms Enterprise Application

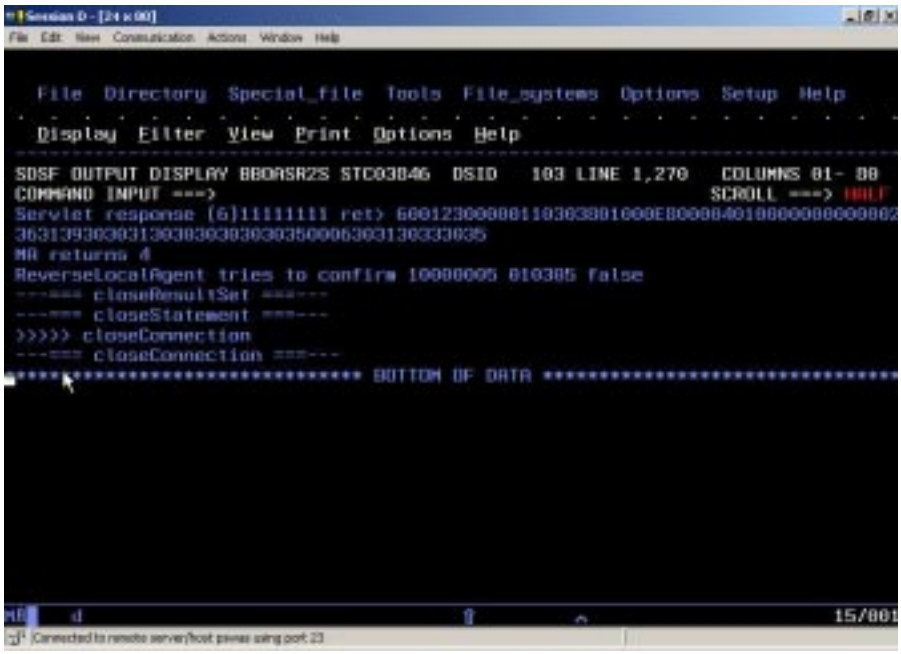


Figure 18: Completion of an authorization transaction

on the *Save* option.

Click on J2EEServer name, press the right mouse button, and choose *Install J2EE application* option (Figure 12). The window for entering the EAR file name and destination FTP server will appear, as shown in Figure 13.

Enter the EAR file name and destination FTP server and press the *OK* button.

Select *posted51* and press the right mouse button. Click on *Validate* as shown in Figure 14.

Then click on *Commit* as shown in Figure 15.

After that, you have to perform the final operation: select *posted51*. Press the right mouse button, and choose **Complete/ All tasks** as shown in Figure 16. After that, the application deployment on z/OS is completed.

RUNNING A jPOS-BASED WEB APPLICATION ON z/OS

In this part of the article the starting of a TMS application on WebSphere Application Server on z/OS is described.

In WebSphere Application Server for z/OS and OS/390 Administrator click on *posted51*, press the right mouse button, and click on *Activate* as shown in Figure 17.

After that, the TMS application is running. Figure 18 shows completion of an authorization transaction on TMS generated on a POS terminal.

FURTHER READING

- *The Name Game: WebSphere z/OS JNDI Naming Concepts*, Kenneth J Muckenhaupt, IBM Design Center for e-transaction processing, 2002.
- *Java 2 on the OS/390 and z/OS Platforms*.
- *Java programming code page considerations – ASCII vs EBCDIC and others*, Frank Petrik, Paschalis Kaltsatis and John Peck, IBM Java Security for S/390 project, 2002. http://www-1.ibm.com/servers/eserver/zseries/software/java/.
- *Java Security on z/OS: An Introduction*, Tom Benjamin, http://www-1.ibm.com/servers/eserver/zseries/zos/racf/pdf/share_03_2002_java_security_zos.pdf.
- IBM Redbook, *Writing Optimized java Applications for z/OS*, http://www.redbooks.ibm.com/redbooks/pdfs/sg246541.pdf.
- IBM Redbook, *e-business Cookbook for z/OS Volume III: Java Development*, http://www.redbooks.ibm.com/redbooks/pdfs/sg246541.pdf.
- *jPOS Programmer's Guide*, http://www.jpos.org/proguide/index.html.
- *Java for OS/390 and z/OS Tips and FAQs*, http://www.s390java.com.

We thank the following people who contributed to this work on TMS porting on z/OS: Ana Dobrašinovic and Dragan Vasovic from ARIUS ad (Belgrade, Serbia and Montenegro), Emina Spasic, Gordana Arsenijevic, and Dragan Nikolic from Postal Savings Bank j s (Belgrade, Serbia and Montenegro).

Mladen Mrkic
Application Developer
Dejan Simic
Associate Professor
ARIUS, ad University of Belgrade (Serbia and Montenegro)

zSeries CPC capping

CPC capping is limiting/restricting CPU usage in a CPC to a desired MSU value. One reason for capping a CPC could be for zSeries subcapacity WLC licensing when the systems are under utilized. zSeries CPC can be capped at the CP level or at the LPAR level.

CAPPING AT CP LEVEL

CPC can be capped by disabling one or more CPs to attain the desired MSU value. This way of capping a CP might require external support (from IBM) to disable a CP. Consider the case of a zSeries box rated at 78 MSU with 2 CPs, it is desired to cap the CPC to 39 MSU. If we assume that both the CPs can deliver the same processing power, disabling a CP would do the trick. Suppose, at some point in time it is required to roll back the capping, we would have to enable the CP. But, this once again could require external support. If the CPC were to be capped at 42 MSU, this way of capping might not work. Also, the same problem would occur when CPC capping was being changed frequently. Capping at the LPAR level could solve this problem.

LPAR CAPPING

LPAR capping limits MSU usage in an LPAR to a desired MSU/ % processing power of CPC capacity, but guarantees this processing capacity in case of contention. An MVS systems programmer at an installation can do this. There are two ways of LPAR capping – hard capping and soft capping.

Hard capping

Hard capping is based on LPAR weighting. The weight of a partition is used to identify the percentage of the processor to which a partition is guaranteed access. The LPAR hipervisor enforces this value when there is contention for CPU cycles. When the processor is not busy, and the LPARs are not capped, a partition can use more CPU resource than this guarantee. The percentage of the processor a partition is guaranteed is the weight of the partition divided by the total weight of all partitions. Consider the same zSeries box at 78 MSU with 2 CPs with 4 LPARs having the weighting shown in Figure 1.

For LPAR1 the minimum guaranteed amount of CPU is 8 MSU at contention, when there is no contention it could even use 78 MSU. If LPAR1 is capped, the maximum that MSU LPAR1 could use is just 8 MSU, even when there is no contention. Capping all the LPARs based on the weighting would give only capping of that LPAR, and the MSU consumed by the entire CPC could be 78 MSU. But, how to cap the CPC to 42 MSU or any MSU is not answered. For this, a dummy LPAR can be created with 46% weighting with capping and no OS would be run on this LPAR – see Figure 2.

LPAR	*Weights*	*% of CPC processing power*	*MSU guaranteed*
LPAR 1	10	10	8
LPAR 2	10	10	8
LPAR 3	50	50	39
LPAR 4	30	30	23

Figure 1: Weighting

LPAR	*Weights*	*% of CPC processing power*	*MSU guaranteed*
LPAR 1	4	4	3
LPAR 2	4	4	3
LPAR 3	21	21	26
LPAR 4	13	13	10
LPAR 5	46	46	36

Figure 2: Weighting

Capping all the LPARs based on weighting and operating all LPARs at their maximum except the dummy LPAR 5, CPU usage would be 42 MSU, as shown in Figure 3.

This method requires the creation of a dummy LPAR, which could be an overhead (it may not always be possible), and the capping is rigid, such that it never allows an overshoot at any point of time. All these problems are handled in soft capping.

Soft capping

In soft capping, MSU capacity is defined for the each LPAR, such that 4-hour rolling-average MSU usage is restricted to the

	MSU usage at full load
LPAR 1	3
LPAR 2	3
LPAR 3	26
LPAR 4	10
LPAR 5	0
Total MSU Usage of CPC	42

Figure 3: CPU usage

defined capacity. It also allows, in any RMF interval, the average MSU can be above the defined capacity provided the 4-hour rolling-average is below the defined capacity. Even in this case weighting are specified only to take care of contentions between LPARs. Even though the LPAR may have been assigned 50% of processor power, it can use only up to its defined capacity based

LPAR	Weights	% of CPC processing power	Defined capacity (MSU)	MSU guaranteed
LPAR 1	10	10	3	3
LPAR 2	10	10	3	3
LPAR 3	50	50	21	21
LPAR 4	30	30	12	12

Figure 4: Soft capping

on its rolling 4-hours average. Considering the same case of capping a zSeries box with 2 CPs at 78 MSU to 42 MSU or to any MSU – see Figure 4.

CPC MSU usage when the LPARs are operating at full would be 42 MSU as in Figure 5.

	MSU usage at full load
LPAR 1	3
LPAR 2	3
LPAR 3	21
LPAR 4	12
Total MSU Usage of CPC	42

Figure 5: MSU usage

CONCLUSION

CPC can be best capped by soft capping. It is less rigid than hard capping because it is flexible enough to allow increases in MSU usage in an LPAR.Also, it does not require any additional definitions/LPARs on the hardware side.

Arun Kumaar R
System Software Group
Tata Consultancy Services (India)

Keeping track of non-reusable ASIDs

PROBLEM ADDRESSED

Since the dawn of computers, storage has been a resource that a lot of users monitor very closely. This can be accomplished by system monitors such as RMF, MainView, or Omegamon, or by commands that are either issued manually or triggered at specific timed intervals. However, every now and then, situations arise where a systems programmer suddenly finds himself with a storage shortage condition. One needs to evaluate the cause: the system appears to be running, but nothing is actually going on or jobs don't start. It is now your task, as a systems programmer, to come up with a resolution and if you are not sure or have no idea what area of storage is experiencing the shortage the following should help you determine this.

It is well known that the system assigns an ASID (Address Space IDentifier) to an address space when the address space is created. It is also the fact that a limited number of ASIDs are available to the system. When all ASIDs are assigned to existing address spaces, the system is unable to start a new address space. This condition might be the result of too many LOST ASIDS (ie non-reusable) in the system. A lost ASID is one that is associated with an address space which has terminated, but, because of the address space's cross memory connections, the system does not reuse the ASID. In other words, when ending a job or started task, the initiator/terminator may find that this address space had been used to provide services to other address spaces through space-switching PC routines, or that this address space provided and did not remove cross memory access through ALESERV. In order to maintain system integrity, the address space is ended and the address space identifier (ASID) is marked as unavailable. This identifier might be temporarily or permanently unavailable. This does not necessarily indicate an error in the program or in the initiator.

When an address space becomes non-reusable either permanently or temporarily because of cross-memory binds, the ASCB (Address Space Control Block) and ASSB (Address Space Secondary Block) remain allocated and queued to the memory delete queue. When an address space becomes reusable, the ASID is added back to the ASVT (Address Space Vector Table – the supervisor table that maps ASIDs to ASCBs) and then the ASCB and ASSB are freed. Thus, to reduce storage impacts, both virtual and real, the ASCB and ASSB need to be freed as soon as an ASID becomes non-usable. Starting with z/OS V1R3 a new protocol is defined between cross-memory and memory delete to allow this. Freeing these blocks minimizes the SQA and ESQA impacts that a non-usable ASID has on the system, and this will allow a large number of replacement ASIDs to be defined with minimal storage impact. Memory delete now frees the ASCB and ASSB once cross-memory has accepted responsibility for the address space. If the address space is permanently non-reusable, because of a bind to a system LX (Linkage Indexes), then the virtual storage impact of the lost ASID is limited to the ASVT entry. If the address space is potentially reusable when binds have terminated, then cross-memory adds the XMSE for the address space to a new reuse queue. The reuse queue will be processed when an address space terminates. When all the binds have terminated, cross-memory will remove the XMSE from the reuse queue and add the ASID back to the ASVT. The ASID reuse code will now exist in both memory delete and cross-memory, because both may make an ASID reusable.

For a detailed description of methods that prevent running out of ASIDs, consult Chapter 3 (*Reusing ASIDs: Coding Cross Memory Services to Avoid the Loss of ASIDs from Reuse*) of *z/OS MVS Programming: Extended Addressability Guide* (SA22-7614-02).

An enquiring mind eager to learn how to identify cross memory connections to an address space may take a look at APAR II08563, which describes the control block chains to follow in order to map the cross memory connections established between address spaces. Based on the process explained therein a

program was written *(Mapping cross memory connections to an address space*) and is available at Xephon's Mainframe Week site (http: //www.mainframeweek.com/journals/articles/0031/)

Yet another useful feature that came with z/OS V1R3 is that monitoring of an ASID and linkage indexes (LXs) usage limits is now done on a timed basis every two minutes. When this limit is crossed, a message is issued. The limits for the resources that are monitored are fixed and cannot be changed. Indications of an LX shortage are set as follows: > 85% in-use is a shortage, < 70% is relieved, while indications of an ASID shortage are set as: > 95% in use is a shortage, < 90% is relieved. In general the messages are warnings, so that an action can be taken, but that probably means scheduling an IPL. Some resources may be able to be recovered by terminating the 'proper' address space or correcting a 'hang', but most of the time there is not much that can be done.

It is now obvious that a twofold problem may occur because of non-reusable ASIDs. On the one hand if started tasks or batch jobs that create unusable ASIDs end enough times, they will exhaust all available ASIDs and an IPL will be required. When this happens one of the following messages appears on the console:

```
IEA0̸59E ASID SHORTAGE HAS BEEN DETECTED
IEA6̸02I ADDRESS SPACE CREATE FAILED MAXUSERS WOULD HAVE BEEN EXCEEDED
IEF352I ADDRESS SPACE UNAVAILABLE
IEF353A INITIATOR TERMINATED DUE TO CROSS MEMORY BIND, RESTART INITIATOR
IEF355A INITIATOR TERMINATED, RESTART INITIATOR
IEF356I ADDRESS SPACE UNAVAILABLE DUE TO CROSS MEMORY BIND
```

These messages indicate that the system has run out of slots in the ASVT. The size of this table, and thus the total number of ASIDs that can be initialized until the system is re-IPLed, is determined during IPL by the values specified in the MAXUSER, RSVNONR, and RSVSTRT options of the IEASYS*xx* parmlib member. The total number of address spaces that can be initialized at one time, during the life of an IPL, is controlled by the total of IEASYS*xx* parameters MAXUSER and RSVSTRT. When any new ASID is to be started, the supervisor first determines

whether there are any slots available for the ASID on the ASVT MAXUSER queue. If the MAXUSER queue is filled, and if then the address space to be started is a started task, the ASVT slot may be satisfied from an unused slot on the RSVSTRT queue. If there are no slots available on either MAXUSER or RSVSTRT queues, the address space will not be initialized.

Unfortunately MVS is not very vocal in these cases since there is not any indication, count, or display of ASIDs hung, waiting for termination, or those that are non-reusable that would help us to quickly gain insight into ASVT free slot shortage.

It is now clear that a way to reduce the possibility of the system running out of ASIDs is through the use of RSVNONR and RSVSTRT parameters in the IEASYS*xx* member of SYS1.PARMLIB. These parameters reserve extra ASIDs to replace those lost because of cross memory activity. RSVSTRT is a number of reserved ASIDs for use when MAXUSER is exhausted and a system address space is needed, while RSVNONR is the number of reserved ASIDs used to replace ASIDs that are non-reusable because of cross-memory integrity reasons. As a matter of fact, the RSVNONR specification reserves a number of ASVT slots to replenish ASVT slots in the MAXUSER or RSVSTRT 'pools' that are marked non-reusable at address space termination. As we have explained earlier, when an address space goes away, its corresponding slot in the ASVT is marked non-reusable. When this occurs, the non-reusable slot is replaced by a slot from the RSVNONR pool. However, once the RSVNONR pool is exhausted, the slots in the MAXUSER or RSVSTRT pool cannot be replaced and are often lost for the remainder of the IPL. See *MVS Initialization and Tuning Guide* (SA22-7592-02) for more information about specifying those parameters as well as on methods for identifying problems in virtual storage. It is helpful to note that the longer one goes between IPLs and the more times one restarts cross-memory subsystems (such as MQ or DB2) the larger RSVNONR needs to be.

On the other hand, if a large number of address spaces become

non-reusable, this can cause an excessive amount of CSA/SQA and ECSA/ESQA to remain allocated for those spaces and force an IPL to occur. In effect, the ASID is 'lost from use' for the duration of the IPL, or until all connected address spaces have terminated. When IPLing is not an acceptable option, determine which programs caused the problem, and search problem reporting databases for a fix for the problem. It may turn out that a failure to complete address space termination has been caused by one of the following reasons: hanging in an address space resource manager running in the master address space, waiting for a 'free' address space termination task in master (which is blocked by other hung address spaces), control block corruption, or an overlay of ASCB/ASSB, etc.

ON-LINE MONITORING

When it comes to on-line monitoring of non-reusable ASIDs there are currently two options available within the standard IBM toolkit. The first one is the Interactive Problem Control System (IPCS) VSMDATA OWNCOMM report, while the second is RMF's Monitor III Common Storage Remaining report (STORCR).

In order to exploit these options common storage tracking must be activated, which is done by the MVS SET command:

```
SET DIAG=Ø1
```

The parmlib member DIAG01 should be set as follows:

```
VSM TRACK CSA(ON) SQA(ON)
```

The IBM CSA tracker is a facility that provides significant amounts of information about common storage ownership captured at the time the storage is acquired and cleared at the time the storage is released. Information collected by the IBM CSA tracker includes time, address, length of the allocation, owner jobname, jobnumber, ASID, and status. Information the IBM CSA tracker does not collect is subpool and storage protect key. It should be noted that CSA tracker maintains the data totally within memory, but the ESQA storage overhead and the small penalty in CPU consumption is neglible given the fact that when things go wrong the ability to examine this information is priceless.

IPCS

From TSO, get into IPCS and change the default for SCOPE to LOCAL and the SOURCE to ACTIVE. Then, from IPCS Option 6 enter the command to request formatting of storage data:

```
VERBEXIT VSMDATA 'SUMMARY OWNCOMM'
```

We would recommend starting with the simplest form of this command to see common storage usage. The VSMDATA OWNCOMM processing finds GQEs (Getmain Queue Elements) by sequentially scanning the GQEs in their cell pool extents, instead of processing the GQE queues anchored in the GQATs. So while it is possible that because of changing storage some address ranges could end up being reported in more than one GQE, this technique avoids having to attempt to deal with apparently broken or looping queues. So we tend to recommend this technique for monitoring programs that do unserialized processing of GQEs.

The output produced by the VSMDATA 'SUMMARY OWNCOMM' subcommand consists of two parts. The first one, called *Grand Totals*, is a summary of how much common storage is defined as well as how much is owned by active ASIDs and by *owner gone* ASIDs. *Owner gone* is IPSC's term for a non-reusable ASID. The second part consists of a summary that lists job names and how much common storage they own. It also has the status column that contains either AC for active or OG for *owner gone*. If you need to obtain the storage addresses, you can issue the command:

```
VERBX VSMDATA 'OWNCOMM DETAIL'
```

and then locate and note or stack the storage return address of the OG entry point and browse (BROWSE option of the IPCS Primary Option Menu) backwards from this address until you find the name of the module that issued the GETMAIN. By adding the CONTENTS(yes) option, the report will show the first few bytes of each area and, if the eyecatcher is present, subpool, key, contents, and length will make a good starting point to determine which product and component allocated the storage. This is

often a very good clue, even if it turns out that the allocater is no longer in storage. It is also a good idea to sort the report: sorting by time can be very useful for identifying the culprit if the storage usage is slowly growing over time.

For information on how to use IPCS to format common storage tracking data, see the description of the VERBEXIT VSMDATA subcommand in *z/OS MVS Interactive ProblemControl System (IPCS) Commands* (SA22-7594-02). For an example of the VSMDATA output, see Chapter 29, 'Virtual Storage Management (VSM)' of *z/OS MVS Diagnosis: Reference* (GA22-7588-02)

RMF

The RMF Monitor III Common Storage Remaining report is a snapshot of the system at the end of the specified report interval and it identifies jobs that have ended but did not release all their allocated common storage (CSA, ECSA, SQA, and ESQA) since IPL. The jobs on this report are sorted in descending order by storage percentage. That is, for each job with the maximum of the four common storage percentages, the job with the highest maximum percentage is reported first. The %REMAIN is summary information about common storage that was not released by ended jobs and is always the first reported line. There are no report options for this report.

Unfortunately, there are no cursor-sensitive fields on this report that would allow us to explore a bit further and thus find an eyecatcher, subpool and storage protection key of storage left allocated by ended job/task.

MXI

A quick and simple way to extract more information from the system is available from MXI (MVS eXtended Information). MXI is an ISPF-based application that enables the systems programmer to display important configuration information about the active MVS, OS/390, or z/OS system. Although primarily used on-line, MXI comes with a REXX interface and can also be

run in batch mode. Most of the displays can be filtered using ISPF-like masking characters and many display fields have ‘point-and-shoot’ functionality that drills down to a more detailed display. MXI is free and available from www.mximvs.com . It does not use any method of CPU serial number protection or encryption. No passwords or activation zaps are required.

MXI can display a wealth of information from your system including the following commands, which might help you when dealing with non-reusable ASIDs.

The Common Storage Remaining command (CSR) provides the same report as RMF III STORCR command. However, unlike RMF III, the jobname column is a cursor-sensitive field and if selected it will invoke the GQE command (new in the latest version of MXI) to display common storage Getmain Queue Elements, which, in turn, display details of each block of common storage allocated including length of allocated storage, type of storage, owner of the storage, return address from Getmain, and date and time that the storage was allocated.

The cursor-sensitive fields of the Getmain Queue Elements report are address (starting address of common storage block), SP-Key (subpool number-storage protect key), and GQE (GQE address). When pointed at an address or GQE field, MXI will invoke the MEM (Display Memory) command, which displays storage at the specified address or control block. If the SP-Key field is selected, the SP (Common Storage Subpool Usage) command will display the current common storage and LSQA subpool use.

In addition, the Memory Delete Queue (MDQ) command is very useful because it displays the ASIDs that have been marked non-reusable by the system and to which address spaces these ASIDs had latent cross memory links. Exploring cross memory connections is provided by the XM command, while the Linkage Indexes and PC Routines display can be obtained by the LX command.

It may happen that after locating ‘unowned’ (aka ‘orphaned’, ‘dead’, or ‘lost’) storage one may get tempted to find a way to free

up CSA or ECSA that wasn't released after job termination to alleviate the storage impact of non-reusable ASIDs or in order to 'save' the system until such a time that an IPL would not disrupt production. There are products that provide the ability to browse common storage and to release 'unused' storage but that is a high-risk situation. What is unused? That is the problem. As mentioned earlier, an ASID can be marked as non-reusable only temporarily and it will be used when all related connections terminate. For example, the ASID that owns a non-system or non-space switching system LX is marked as non-reusable, as is also the case with the ASID that is connected to another address space which owns a non-system or non-system LX.

The fact that the original owner has gone does not necessarily mean the storage is no longer in use. The side effects of releasing 'unowned' storage are thus unpredictable and may rapidly lead to an outage anyway. If one tries to free 'unowned' storage, one can cause a data integrity problem and corrupt a database, and IPLing won't fix that.

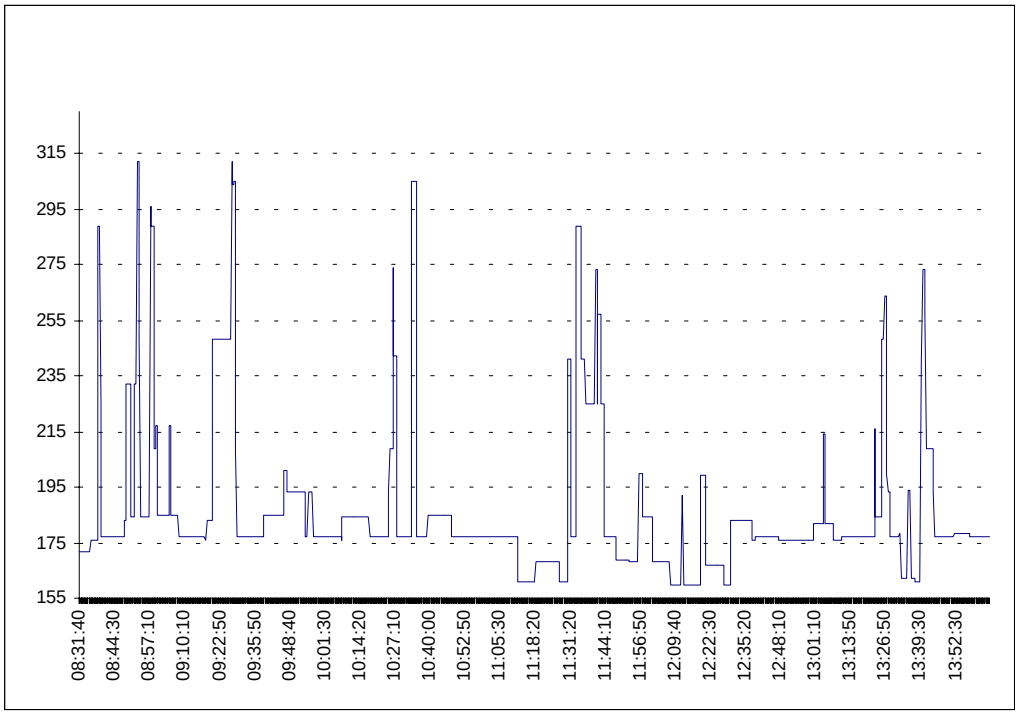


Figure 1: ECSA – number of frames not released

The bottom line is this: it is difficult to determine whether this kind of error is caused by bad coding practice by the vendors or bad design by IBM. For an example of misclassification of storage ownership see APAR OW47700. However, I do have suggestions for avoiding future problems: start tracking CSA/SQA/ECSA/ESQA usage, and use RMF or MXI to display the amount of SQA/CSA held by particular jobs or address spaces.

If one or more jobs or address spaces are using an excessive amount of CSA/SQA, tell the operator to cancel those jobs or address spaces.

Obtain a dump of common storage. Run IPCS verbx vsmdata 'owncomm' reporting. Determine whether there has been storage growth beyond the previous normal range or whether the allocation of SQA and/or CSA is inadequate (one may need to increase the sizes in IEASYS*xx* to allow for some breathing space).

COLLECTING DATA AND REPORTING

Up to now we have seen that on-line monitoring of non-reusable ASIDs could and should proactively be done. Things are, however, rather different when one tries to collect data pertaining to these ASIDs. One may try to run RMF post-processor's VSTOR report, which enables you to measure the use of virtual storage with minimal overhead. It contains the information you need to understand your current use of virtual storage. If you archive the data, you can use differences over time to predict a problem or constraint before it becomes critical. It also helps you to verify the size values set for CSA and SQA at IPL time, and determine whether you are using common storage effectively. It would be very useful also to have RMF Monitor III data tables recorded by SMF along with other RMF records, but for the time being that is not the case. There is a way, however, to force RMF to collect Monitor III data and to write it to a predefined VSAM dataset. This method has the advantage over direct RMF III analysis in that the data will not roll off – one does not have to look at screen after screen to try to determine a trend. If an action is taken to improve CSA/SQA usage, this allows comparisons of the before and after

conditions from one week to the next (for example). For information on this collecting method see Chapter 6, 'Using Monitor III VSAM Data Set Support' in *z/OS Resource Measurement Facility: Programmer's Guide* (SC33-7994-01).

At this point it should be noted that there is not an easy way to retrieve compressed RMF VSAM Monitor III data. Instead, an easy-to-use ready-made solution provided by RMF Performance Monitor was used.

RMF Performance Monitor (RMF PM) is IBM's free performance tool that provides mechanisms for collecting, displaying, monitoring, and logging real-time performance data from your system. You can combine nearly all data in DataViews as you like, without any restriction of predefined reports. You can plot a DataView for a selected period of your gathering interval, and you can save the data into a .WK1 file (up to 8,000 values) for further processing with any spreadsheet application. In order to track the use of ECSA (just an example) a DataView was created to collect the number of 4K frames not released at the end of the interval (Figure 1).

CODE

A neat batch interface that MXI provides was utilized to get the information pertaining to non-reusable ASIDs. The MXI commands are entered via the SYSIN DDname and the resultant output is written to the SYSPRINT DDname. Multiple commands can be specified and they will be processed in sequential order. The MXI program must be invoked with the parameter BATCH, but can also have the optional extra parameter NOTITLES to suppress the screen titles. A simple but effective batch procedure was constructed with the aim of producing a more precise common storage allocation report that would aid in monitoring and analysing non-reusable ASIDs. The procedure should be submitted or triggered at specific timed intervals in order to get a valid snapshots of the common storage allocation.

The code is a two-part stream: in the first part (MXIBATCH) MXI was invoked to issue the GQE command and its output was then

processed in step 2 (VSM) by invoking SAS statistical analysis of data. SAS was used since it is the most flexible tool for producing quick and accurate reports of what the data represents. There are five sets of reports produced by this report writer, each providing in-depth information on a specific area of common storage allocation being observed.

CSRPT is a set of four common storage allocation overall reports providing information on each type of CS in regard to ASIDs status, subpool usage, and storage protect key:

		STORAGE TYPE			
COMMON STORAGE	ASID	CSA		SQA	
LOCATION	STATUS	LENGTH	PCT	LENGTH	PCT
ABOVE	ACTIVE	43.6M	9Ø.78%	7.8M	97.18%
	GONE	1.4M	3.Ø5%	53.ØK	Ø.65%
ABOVE		45.1M	93.83%	7.9M	97.84%
BELOW	ACTIVE	2.9M	6.14%	172.3K	2.12%
	GONE	14.8K	Ø.Ø3%	3.3K	Ø.Ø4%
BELOW		2.9M	6.17%	175.7K	2.16%

HIST is a set of three reports each providing data sufficient to determine a trend).

OWNSTAT reports on CS allocation by owner (ASID) status:

	STORAGE TYPE					
	CSA			SQA		
	COMMON STORAGE LOCATION			COMMON STORAGE		
LOCATION						
STORAGE	ABOVE	BELOW	# CS	ABOVE	BELOW	# CS
OWNER			BLOCKS			BLOCKS
MASTER	2.5M	12Ø.4K	65	942.4K	64.5K	69
SYSTEM	12.9M	86.6K	4Ø	7.1M	2Ø2.7K	177
……..						
CATALOG	64		2	35.ØK	3.2K	26
CICS	66Ø.2K	5.1K	6	2.6K		14
DFSHSM	365.2K		777	25.6K	1.6K	6
JESXCF	321.8K		19	8.6K	64	1
JES2	55Ø.5K	31.2K	12	1.7K	64	1
JES2AUX				16Ø		1
LLA				2Ø8		5

OWNDET provides a detailed report on ASID's use of CS:

				COMMON STORAGE LOCATION	
STORAGE OWNER	ASID STATUS	STORAGE TYPE	SUBPOOL #	ABOVE	BELOW
CATALOG	ACTIVE	CSA	231	32	
		SQA	226		512
			239	2.5K	512
			245	13.3K	512
			248	480	
CATALOG	ACTIVE			16.3K	1.5K
	GONE	CSA	241	32	
		SQA	226		384
			239	5.3K	256
			245	12.3K	1.1K
			247	128	
			248	984	
CATALOG	GONE			18.7K	1.7K
CATALOG				35.1K	3.2K
CICS	ACTIVE	CSA	227	7.2K	136
			228	14.0K	
			231	167.9K	
			241	14.9K	
		SQA	239	32	
			245	1.3K	
CICS	ACTIVE			205.6K	136
	GONE	CSA	227	66.0K	2.0K
			228	12.0K	3.0K
			231	340.0K	
			241	38.0K	
		SQA	239	1.2K	
CICS	GONE			457.2K	5.0K
CICS				662.9K	5.1K
DFSHSM	ACTIVE	CSA	227	512	
			231	160	
			241	624	
		SQA	239	32	
			245	992	
DFHSM	ACTIVE			2.2K	
	GONE	CSA	227	67.8K	
			228	11.4K	
			231	148.0K	
			241	136.6K	

```
                    SQA       239              4.8K          872
                              245             19.7K          800
--------  --------                          --------      --------
DFHSM     GONE                                388.5K         1.6K
--------                                    --------      --------
DFHSM                                         390.8K         1.6K
```

OWNGONE set provides four reports on ‘owner gone’ ASIDs.

Below is a summary of CS ‘owner gone’ ASIDs by date:

			COMMON STORAGE	
LOCATION ALLOCATION DATE	CSA LENGTH	CSA/E LENGTH	SQA LENGTH	SQA/E LENGTH
2003/10/27	2.1K	397.1K	3.0K	27.0K
2003/10/29	224	8.7K		
2003/10/30	3.6K	23.4K		
2003/11/03	144	32.0K		4.8K
2003/11/04		15.5K		
2003/11/05	32	23.8K		
2003/11/06	256	23.1K		
2003/11/07	208	66.8K		4.8K
2003/11/08	16	3.8K		
2003/11/09		32	344	
2003/11/10	1.8K	19.7K		
2003/11/11	192	11.5K		
2003/11/12		23.0K		6.1K
2003/11/13	384	23.7K		
2003/11/14		3.8K		
2003/11/17	576	15.8K		
2003/11/20		23.1K		
2003/11/21	1.8K	11.6K		128
2003/11/22		38.2K		9.7K
2003/11/23		22.9K		
2003/11/24	48	31.7K		
2003/11/25	400	164.0K		128
2003/11/27	32	64		
2003/12/03		60.8K		
2003/12/04	256	340.0K		

Below is an example report of ‘owner gone’ ASIDs:

STORAGE OWNER	ALLOCATION DATE	TIME	ALLOC LOCATION	SPOOL-KEY	LENGTH
CATALOG	2003/10/27	08.43.20	SQA	226-0	128
		08.43.20	SQA/E	245-0	576
		08.43.20	SQA/E	248-0	784
		08.44.27	SQA	239-0	128
		08.44.27	SQA/E	239-0	464
		08.44.27	SQA/E	245-0	896

```
                    Ø8.44.36   SQA        226-Ø          128
                    Ø8.44.36   SQA        245-Ø          128
                    Ø8.44.36   SQA/E      239-Ø          224
                    Ø8.44.36   SQA/E      245-Ø         2.ØK
                    Ø8.44.36   SQA/E      248-Ø          2ØØ
                    11.47.32   SQA        245-Ø          256
                    11.47.32   SQA/E      239-Ø          568
                    11.47.32   SQA/E      245-Ø         1.8K
                    11.47.32   SQA/E      247-Ø          128
CICS      2003/10/27 Ø8.5Ø.Ø5   CSA/E      227-6         4.ØK
                    Ø8.5Ø.2Ø   CSA/E      227-6        16.ØK
                    Ø8.52.Ø2   CSA/E      231-7        2Ø.ØK
                    Ø8.55.3Ø   CSA/E      231-7        16.ØK
                    Ø8.57.53   CSA/E      228-6         4.ØK
                    Ø8.57.53   CSA/E      231-7        2Ø.ØK
                    Ø8.57.53   CSA/E      241-Ø         4.ØK
                    Ø8.59.36   CSA/E      227-6        16.ØK
                    Ø8.59.47   CSA/E      227-6         6.ØK
                    Ø9.ØØ.ØØ   CSA/E      228-Ø           48
                    Ø9.Ø7.4Ø   CSA/E      241-6         4.ØK
                    Ø9.18.29   CSA/E      231-7         4.ØK
                    1Ø.46.31   CSA/E      241-6         4.ØK
                    11.41.14   CSA/E      241-6         6.ØK
                    11.55.43   CSA/E      241-7         4.ØK
                    17.45.Ø3   CSA/E      227-6         4.ØK
                    17.45.Ø3   CSA/E      231-5        4Ø.ØK
                    17.45.Ø3   CSA/E      231-6        16.ØK
                    17.45.Ø3   CSA/E      231-7        2Ø.ØK
          2003/10/3Ø 11.43.18   CSA/E      228-6         4.ØK
                    15.Ø2.42   CSA        227-6         1.ØK
                    15.Ø2.44   CSA        227-6         1.ØK
                    15.Ø2.45   CSA        228-7         1.ØK
          2003/10/31 13.Ø9.32   CSA        228-7         1.ØK
                    13.19.54   CSA        228-7         1.ØK
          2003/11/Ø3 Ø7.11.29   CSA/E      241-6        16.ØK
          2003/11/Ø7 13.56.31   CSA/E      231-7        36.ØK
          2003/11/12 14.38.27   SQA/E      239-Ø         1.2K
          2003/11/2Ø 15.2Ø.28   CSA/E      227-6         4.ØK
          2003/11/28 11.17.48   CSA/E      231-Ø        16.ØK
                    11.17.48   CSA/E      231-5        16.ØK
                    11.17.48   CSA/E      231-7       132.ØK
                    12.23.19   CSA/E      231-5         4.ØK
          2003/12/Ø3 Ø7.Ø3.Ø4   CSA/E      228-Ø         4.ØK
                    Ø8.31.3Ø   CSA/E      227-6        16.ØK
```

JOB

```
//MXIBATCH  EXEC PGM=MXI,PARM='BATCH,NOTITLES'
//* Invoke MXI GQE command to get the snapshot
```

```
//* of all allocated Common Storage blocks
//SYSPRINT  DD DSN=&&OUTSMF,DISP=(NEW,PASS),
//          DCB=(RECFM=FB,LRECL=8Ø),
//          SPACE=(CYL,(1,1))
//SYSIN     DD   *
  GQE
/*
//VSM       EXEC SAS
//WORK      DD UNIT=SYSDA,SPACE=(CYL,(5Ø,5))
//GQELIST   DD DISP=(OLD,DELETE),DSN=&&OUTSMF
//SASLIST   DD SYSOUT=*
//CSRPT     DD SYSOUT=*
//HIST      DD SYSOUT=*
//OWNSTAT   DD SYSOUT=*
//OWNDET    DD SYSOUT=*
//OWNGONE   DD SYSOUT=*
//SYSUDUMP  DD DUMMY
//SYSIN     DD *
  /* Process the list of Getmain Queue Elements  */
  /* that describe all allocated (active & gone)  */
  /* blocks of Common Storage                      */
OPTIONS NOCENTER LINESIZE=255 PAGESIZE=999;
PROC FORMAT;
    PICTURE BKM Ø-1Ø23='ØØØØØØ'
    1Ø24-1Ø48576='ØØØ.ØK' (MULT=Ø.ØØ9765625)
           OTHER='ØØØ.ØM' (MULT=Ø.ØØØØØ9765625);
DATA CSALLOC;
 INFILE GQELIST;
   INPUT @Ø1 ADDRESS    HEX8.    /* Starting address of CS block */
         @13 LEN        8.       /* Length of allocated storage  */
         @21 TYP        $CHAR1.  /* Type of storage: CSA, SQA    */
         @22 STAT       $CHAR1.  /* ASID status: active, gone    */
         @24 OWNER      $CHAR8.  /* Owner of the storage         */
         @33 SUBPOOL    $CHAR3.  /* Subpool number               */
         @37 KEY        $CHAR1.  /* Storage protect key          */
         @41 DATE       $CHAR1Ø. /* Allocation date              */
         @52 TIME       $CHAR8.  /* Allocation time              */
         @61 GQEADDR    HEX8.    /* GQE address                  */
         @7Ø RETADDR    HEX8.;   /* Return address from Getmain  */
    IF ADDRESS > 16777215 THEN AREA ='ABOVE';
    ELSE AREA ='BELOW';
    IF TYP  = ,S' THEN TYPE = ,SQA';
    ELSE TYPE = ,CSA';
    IF STAT = ,-' THEN STATUS = ,GONE  ,;
    ELSE STATUS = ,ACTIVE';
 LABEL
  AREA    ='Common Storage*location'
  LEN     ='Length'
  TYPE    ='Storage*type'
  STATUS  ='ASID*status'
```

```
 OWNER   ='Storage*owner'
 SUBPOOL ='Subpool #'
 KEY     ='Storage*protect*key'
 DATE    ='Allocation*date'
 TIME    ='Alloc*time';
 /*-------------- History data ----------------- */
PROC SUMMARY DATA=CSALLOC NWAY;
   CLASS DATE TYPE STATUS SUBPOOL AREA;
   VAR LEN;
   OUTPUT OUT=CSAHIS SUM= N=NUMREC;
PROC PRINTTO PRINT=HIST; OPTIONS PAGENO=1;
TITLE ,Common Storage allocation by date';
PROC REPORT DATA=CSAHIS SPLIT='*';
COLUMN DATE STATUS AREA,LEN;
DEFINE DATE  /GROUP  WIDTH=11;
DEFINE STATUS/GROUP  WIDTH=8;
DEFINE AREA  /ACROSS;
DEFINE LEN    /SUM    FORMAT=BKM8.'--';
BREAK AFTER DATE/ SUMMARIZE OL SKIP;
PROC PRINTTO PRINT=HIST; OPTIONS PAGENO=1;
TITLE ,Common Storage type allocation by date';
PROC REPORT DATA=CSAHIS SPLIT='*';
COLUMN DATE STATUS TYPE AREA,LEN;
DEFINE DATE  /GROUP  WIDTH=11;
DEFINE STATUS/GROUP  WIDTH=8;
DEFINE TYPE  /GROUP  WIDTH=8;
DEFINE AREA  /ACROSS;
DEFINE LEN   /SUM    FORMAT=BKM8.'--';
BREAK AFTER STATUS/ SUMMARIZE OL SKIP;
BREAK AFTER DATE/ SUMMARIZE OL SKIP;
PROC PRINTTO PRINT=HIST; OPTIONS PAGENO=1;
TITLE ,common storage type & subpool allocation by date';
PROC REPORT DATA=CSAHIS SPLIT='*';
COLUMN DATE STATUS TYPE SUBPOOL AREA,LEN;
DEFINE DATE   /GROUP  WIDTH=11;
DEFINE STATUS /GROUP  WIDTH=8;
DEFINE TYPE   /GROUP  WIDTH=8;
DEFINE SUBPOOL/GROUP  WIDTH=8;
DEFINE AREA   /ACROSS;
DEFINE LEN    /SUM    FORMAT=BKM8.'--';
BREAK AFTER STATUS/SUMMARIZE OL SKIP;
BREAK AFTER DATE/SUMMARIZE OL SKIP;
PROC DELETE;
    DATA CSAHIS;
 /* --Common Storage & subpools report -------- */
PROC SUMMARY DATA=CSALLOC NWAY;
   CLASS TYPE STATUS SUBPOOL DATE AREA;
   VAR LEN;
   OUTPUT OUT=CSAOUT SUM= N=NUMREC;
 PROC PRINTTO PRINT=CSRPT;OPTIONS PAGENO=1;
```

```
 TITLE ,Common Storage allocation -  Summary';
 PROC REPORT DATA=CSAOUT split='*';
 COLUMN AREA STATUS TYPE,(LEN PCT);
 DEFINE AREA  /GROUP  WIDTH=15;
 DEFINE STATUS/GROUP  WIDTH=8;
 DEFINE TYPE  /ACROSS WIDTH=8;
 DEFINE PCT   /COMPUTED FORMAT=PERCENT8.2 ;
 DEFINE LEN   /SUM FORMAT=BKM8.;
   COMPUTE BEFORE;
    TOTCSA=_C3_;
    TOTSQA=_C5_;
   ENDCOMP;
  COMPUTE PCT;
   _C4_=_C3_/TOTCSA;
   _C6_=_C5_/TOTSQA;
  ENDCOMP;
    BREAK AFTER AREA/ SUMMARIZE OL SKIP;
 PROC PRINTTO PRINT=CSRPT;OPTIONS PAGENO=1;
 TITLE ,Common Storage subpools allocation - Part I';
 PROC REPORT DATA=CSAOUT SPLIT='*';
 COLUMN TYPE SUBPOOL AREA,LEN;
 DEFINE TYPE   /GROUP  WIDTH=8;
 DEFINE SUBPOOL/GROUP  WIDTH=8;
 DEFINE AREA   /ACROSS;
 DEFINE LEN    /SUM FORMAT=BKM8.'--';
   BREAK AFTER TYPE/SUMMARIZE OL SKIP;
 PROC PRINTTO PRINT=CSRPT;OPTIONS PAGENO=1;
 PROC REPORT DATA=CSAOUT SPLIT='*';
 TITLE ,Common Storage subpools allocation - Part II';
 COLUMN STATUS SUBPOOL AREA,LEN;
 DEFINE STATUS /GROUP  WIDTH=8;
 DEFINE SUBPOOL/GROUP  WIDTH=8;
 DEFINE AREA   /ACROSS;
 DEFINE LEN    /SUM FORMAT=BKM8.'--';
   BREAK AFTER STATUS/SUMMARIZE OL SKIP;
PROC DELETE;
     DATA CSAOUT;
PROC SUMMARY DATA=CSALLOC NWAY;
 CLASS STATUS TYPE SUBPOOL KEY AREA;
 VAR LEN;
 OUTPUT OUT=CSASP  SUM= N=NUMREC;
 PROC PRINTTO PRINT=CSRPT;OPTIONS PAGENO=1;
 TITLE ,Subpool allocation  - detail';
 PROC REPORT DATA=CSASP  SPLIT='*';
 COLUMN TYPE STATUS SUBPOOL KEY AREA,LEN;
 DEFINE TYPE    /GROUP  WIDTH=8;
 DEFINE STATUS  /GROUP  WIDTH=8;
 DEFINE SUBPOOL /GROUP  WIDTH=8;
 DEFINE KEY     /GROUP  WIDTH=8;
 DEFINE AREA    /ACROSS;
```

```
 DEFINE LEN     /SUM FORMAT=BKM8.'--';
   BREAK AFTER STATUS/SUMMARIZE OL SKIP;
   BREAK AFTER TYPE/SUMMARIZE OL SKIP;
  RUN;
PROC DELETE;
     DATA CSASP;
 /* ------ Owner reports: summary & datail------ */
PROC SUMMARY DATA=CSALLOC NWAY;
  CLASS OWNER TYPE  AREA;
  VAR LEN;
  OUTPUT OUT=OWNOUT SUM= N=NEC;
LABEL
 NEC     ='# CS*BLOCKS';
 PROC PRINTTO PRINT=OWNSTAT;OPTIONS PAGENO=1;
 TITLE ,Common Storage allocation by Owner (active & gone)';
 PROC REPORT DATA=OWNOUT SPLIT='*';
 COLUMN OWNER TYPE,(AREA,LEN NEC);
 DEFINE OWNER  /GROUP  WIDTH=8;
 DEFINE NEC    /FORMAT=4. WIDTH=6;
 DEFINE TYPE   /ACROSS;
 DEFINE AREA   /ACROSS WIDTH=6;
 DEFINE LEN    /SUM    FORMAT=BKM8.'--';
RUN;
PROC DELETE;
     DATA OWNOUT;
 PROC SUMMARY DATA=CSALLOC NWAY;
   CLASS OWNER STATUS TYPE SUBPOOL AREA;
   VAR LEN;
   OUTPUT OUT=OWNSTD SUM= N=NUMREC;
  PROC PRINTTO PRINT=OWNDET;OPTIONS PAGENO=1;
  TITLE ,Common Storage allocated by Owner - Detail';
  PROC REPORT DATA=OWNSTD SPLIT='*';
  COLUMN OWNER STATUS TYPE SUBPOOL AREA,LEN;
  DEFINE OWNER   /GROUP  WIDTH=8;
  DEFINE STATUS  /GROUP  WIDTH=8;
  DEFINE TYPE    /GROUP  WIDTH=8;
  DEFINE SUBPOOL /GROUP  WIDTH=8;
  DEFINE AREA    /ACROSS;
  DEFINE LEN     /SUM FORMAT=BKM8. ,--';
   BREAK AFTER STATUS/SUMMARIZE OL SKIP;
   BREAK AFTER OWNER/SUMMARIZE  OL SKIP;
PROC DELETE;
     DATA OWNSTD;
 /* ------ Owner gone reports: summary & datail------ */
DATA OWNERG (DROP = STAT STATUS
                    TYP  ADDRESS);
SET CSALLOC;
IF STATUS ='ACTIVE' THEN DELETE;
IF AREA ='ABOVE' THEN LOC = TYPE !! ,/E';
   ELSE LOC =TYPE;
```

```
SPK=SUBPOOL !! ,-'!!KEY;
LABEL
 LOC    ='COMMON STORAGE LOCATION';
PROC PRINTTO PRINT=OWNGONE; OPTIONS PAGENO=1;
PROC SUMMARY DATA=OWNERG NWAY;
    CLASS DATE LOC;
    VAR LEN;
    OUTPUT OUT=K SUM=  N=NBL;
 TITLE ,CS allocation by „owner gone" ASIDs';
 PROC REPORT DATA=K SPLIT='*';
 COLUMN DATE  LOC , LEN;
 DEFINE DATE  /GROUP;
 DEFINE LOC   /ACROSS WIDTH=6;
 DEFINE LEN   /SUM FORMAT=BKM8.;
PROC DELETE;
     DATA K;
  PROC SUMMARY DATA=OWNERG NWAY;
    CLASS OWNER TYPE  AREA;
    VAR LEN;
    OUTPUT OUT=OWNSUM SUM= N=NEC;
  LABEL
   NEC     ='# CS*BLOCKS';
 TITLE ,Common Storage allocation by Owner gone - Summary';
 PROC REPORT DATA=OWNSUM SPLIT='*';
 COLUMN OWNER TYPE,(AREA,LEN NEC);
 DEFINE OWNER  /GROUP  WIDTH=8;
 DEFINE NEC    /FORMAT=4. WIDTH=6;
 DEFINE TYPE   /ACROSS;
 DEFINE AREA   /ACROSS WIDTH=6;
 DEFINE LEN    /SUM    FORMAT=BKM8.'--';
PROC DELETE;
     DATA OWNSUM;
PROC SUMMARY DATA=OWNERG NWAY;
    CLASS DATE OWNER TIME TYPE SPK AREA;
    VAR LEN;
    OUTPUT OUT=K1 SUM=  N=NBL;
 TITLE ,„Owner gone" ASIDs: by date, time and subpools';
 PROC REPORT DATA=K1 SPLIT='*';
 COLUMN DATE OWNER TIME TYPE SPK (AREA, LEN);
 DEFINE DATE  /GROUP;
 DEFINE OWNER /GROUP;
 DEFINE TIME  /WIDTH=1Ø;
 DEFINE TYPE  /GROUP WIDTH=7;
 DEFINE SPK   /WIDTH=7;
 DEFINE AREA  /ACROSS WIDTH=6;
 DEFINE LEN   /SUM FORMAT=BKM8.;
 BREAK AFTER DATE/ SUMMARIZE OL SKIP;
PROC DELETE;
     DATA K1;
PROC SUMMARY DATA=OWNERG NWAY;
```

```
    CLASS OWNER DATE TIME LOC SPK;
    VAR LEN;
    OUTPUT OUT=K3 SUM=  N=NBL;
  TITLE ,„Owner gone" ASIDs: owner name, date & subpools';
  PROC REPORT DATA=K3 SPLIT='*';
  COLUMN OWNER DATE TIME LOC SPK LEN;
  DEFINE OWNER /GROUP;
  DEFINE DATE  /GROUP;
  DEFINE TIME  /WIDTH=9;
  DEFINE LOC   /WIDTH=1Ø ,LOCATION';
  DEFINE SPK   /WIDTH=1Ø ,SPOOL-KEY';
  DEFINE LEN   /FORMAT=BKM8.;
 PROC DELETE;
      DATA K3;
```

CONCLUSION

We have seen that a non-reusable address space is one where a job that ended had been running in a cross memory environment. When such a job ends, the system ends the address space and marks its associated ASVT entry non-reusable (unavailable) until all the address spaces with which the job had cross memory binds have ended. This makes it sound like everything will clean itself out eventually, but it just never seems to happen. Finding the culprit is not that easy and, unless you want to randomly check the IBM repository for reported ‘owner gone’ errors, a systematic approach like the one described in this article might help you narrow down the list of suspects.

Mile Pekic
Systems Programmer (Serbia and Montenegro)

REXX routine to count lines of COBOL code – part 2

This month we conclude the code for the routine that counts the lines of COBOL code and also demonstrates some useful techniques in REXX/ISPF.

```
/* ************************************************************** */
  val_rc = Ø
  if cpyexp = 'Y' & cbsuse ¬= 'I' then
    /* expand copybooks */
    do
      if cpydsn.Ø <= Ø then
        do
          say 'no copybook datasets defined: cpyexp changed to "N"'
          cpyexp = 'N'
        end
        do
          do i = cpydsn.Ø to 1 by -1
            cpydsn = cpydsn.i
            call ext_cpy
          end
          call eval_copy
          call eval_cpy2
          if unresolved then
            call eval_cpy3
          call write_memlist
        end
    end
  call ext_src
exit Ø
/* extract the copybook values per copybook */
ext_cpy:
  ADDRESS TSO
  x = OUTTRAP(lm.,"*")
  "LISTD" cpydsn "MEMBERS"
  x = OUTTRAP("off")
  do j = 1 to lm.Ø
    if lm.j = "--MEMBERS--" then
    do
      call countmem
      leave
    end
  end
return
/* extract the copybook values per member */
countmem:
  do qc = 1 to queued()
    pull .
  end
  do k = j+1 to lm.Ø
    t1 = t1 + 1
    cpydsnx = strip(cpydsn,"B","'")
    parse value lm.k with mem plus
    cpydsny = "'"cpydsnx"("mem")'"
  /* edit macro called per member */
  "ISPEXEC EDIT DATASET("cpydsny") MACRO(COBCMEM)"
```

```
  /* put the extracted values into an array */
  pull cobc_mstr
  table1.t1.mst = cobc_mstr
  pull cobc_post
  table1.t1.pos = cobc_post
  pull cobc_pre
  table1.t1.pre = cobc_pre
  end
return
/* evaluate all unnested copybooks and convert to REXX variables */
eval_copy:
  do t = 1 to t1
  mem1 = word(table1.t.mst,2)
  var1 = mem1 '='
    if pdo = 'Y' then
      do
      if words(table1.t.pos) = 4 then
        do
          var2 = word(table1.t.pos,2) - word(table1.t.pos,4)
          interpret var1 var2
        end
      end
    else
      do
      if words(table1.t.pos) = 4 & (words(table1.t.pre) = 4 ,
                                  | word(table1.t.pre,1) = 'NULL') then
        do
          var2 =  word(table1.t.mst,4) - word(table1.t.mst,6)
          interpret var1 var2
        end
      end
  end
return
/* evaluate second level of nesting for copybooks */
eval_cpy2:
  v1 = Ø
  unresolved = Ø
  do t = 1 to t1
  mem1 = word(table1.t.mst,2)
  var1 = mem1 '='
    if pdo = 'Y' then
      do
      if words(table1.t.pos) = 4 then
        do
          var2 =  word(table1.t.pos,2) - word(table1.t.pos,4)
          cp = cp + 1
          memlist.cp = mem1 var2
          interpret var1 var2
        end
      else
```

```
    do
      pos1 = word(table1.t.mst,2)
      restpos = ''
      poscnt = words(table1.t.pos)
      poschk = word(table1.t.pos,2) - word(table1.t.pos,4)
      allnumeric = 1
      do u = 5 to poscnt
        posnum = ''
        var1   = 'posval ='
        posnum = word(table1.t.pos,u)
        interpret var1 posnum
        if datatype(posval,N) then
          do
            poschk = poschk + posval - 1
          end
        else
          do
            allnumeric = Ø
            restpos = restpos word(table1.t.pos,u)
          end
      end
      if allnumeric then
        do
          mem1 = word(table1.t.mst,2)
          var1 = mem1 '='
          var2 = poschk
          cp = cp + 1
          memlist.cp = mem1 var2
          interpret var1 var2
        end
      else
        do
          unresolved = 1
          v1 = v1 + 1
          table2.v1.pos = pos1 poschk restpos
        end
    end
  end
else
  do
    if words(table1.t.pos) = 4 & (words(table1.t.pre) = 4 ,
                                | word(table1.t.pre,1) = 'NULL') then
      do
        var2 =  word(table1.t.mst,4) - word(table1.t.mst,6)
        cp = cp + 1
        memlist.cp = mem1 var2
        interpret var1 var2
      end
    else
      do
```

```
nam1 = word(table1.t.mst,2)
restmst = ''
allnumeric = 1
poscnt = words(table1.t.pos)
poschk = word(table1.t.pos,2) - word(table1.t.pos,4)
do u = 5 to poscnt
  posnum = ''
  var1   = 'posval ='
  posnum = word(table1.t.pos,u)
  interpret var1 posnum
  if datatype(posval,N) then
    do
      poschk = poschk + posval - 1
    end
  else
    do
      allnumeric = Ø
      restpos = restpos word(table1.t.pos,u)
    end
end
precnt = words(table1.t.pre)
if  word(table1.t.pre,1) = 'NULL' then
  do
    prechk = Ø
  end
else
  do
  prechk = word(table1.t.pre,2) - word(table1.t.pre,4)
  do u = 5 to precnt
    prenum = ''
    var1   = 'preval ='
    prenum = word(table1.t.pos,u)
    interpret var1 prenum
    if datatype(preval,N) then
      do
        prechk = prechk + preval - 1
      end
    else
      do
        allnumeric = Ø
        restpos = restpos word(table1.t.pre,u)
      end
  end
  end
if allnumeric then
  do
    mem1 = word(table1.t.mst,2)
    var1 = mem1 '='
    mstchk = poschk + prechk
    var2 = mstchk
```

```
              cp = cp + 1
              memlist.cp = mem1 var2
              interpret var1 var2
            end
          else
            do
              unresolved = 1
              v1 = v1 + 1
              table2.v1.mst = nam1 mstchk restpos
            end
        end
      end
  end
return
/* evaluate third level of nesting for copybooks                    */
/* unresolved copybooks or further nesting ignored                  */
eval_cpy3:
  do v = 1 to v1
  mem1 = word(table2.v.pos,1)
  var1 = mem1 '='
    if pdo = 'Y' then
        do
          poscnt = words(table2.v.pos)
          poschk = word(table2.v.pos,2)
          allnumeric = 1
          do u = 3 to poscnt
            posnum = ''
            var1   = 'posval ='
            posnum = word(table2.v.pos,u)
            interpret var1 posnum
            if datatype(posval,N) then
              do
                poschk = poschk + posval - 1
              end
          end
          mem1 = word(table2.v.pos,1)
          var1 = mem1 '='
          var2 = poschk
          cp = cp + 1
          memlist.cp = mem1 var2
          interpret var1 var2
        end
    else
        do
          mstcnt = words(table2.v.mst)
          mstchk = word(table2.v.mst,2)
          allnumeric = 1
          do u = 3 to mstcnt
            mstnum = ''
            var1   = 'mstval ='
```

```
            mstnum = word(table2.v.mst,u)
            interpret var1 mstnum
            if datatype(mstval,N) then
              do
                mstchk = mstchk + mstval - 1
              end
          end
          mem1 = word(table2.v.mst,1)
          var1 = mem1 '='
          var2 = mstchk
          cp = cp + 1
          memlist.cp = mem1 var2
          interpret var1 var2
        end
  end
return
/* write the copybook info. store dataset */
write_memlist:
  memlist.Ø = cp
  if cbsuse = 'U' | cbsuse = 'O' then
    call output_cbs
return
/* extract the source dataset values per datset */
ext_src:
  s1 = Ø
  sl = Ø
  ADDRESS TSO
  x = OUTTRAP(lm.,"*")
  "LISTD" srcdsn "MEMBERS"
  x = OUTTRAP("off")
  do j = 1 to lm.Ø
    if lm.j = "--MEMBERS--" then
      do
        call countsrc
        leave
      end
  end
return
/* extract/count the source dataset values per member */
countsrc:
  do qc = 1 to queued()
    pull .
  end
  do k = j+1 to lm.Ø
    s1 = s1 + 1
    srcdsnx = strip(srcdsn,"B","'")
    parse value lm.k with mem plus
    srcdsny = "'"srcdsnx"("mem")'"
    /* edit macro called per member */
    "ISPEXEC EDIT DATASET("srcdsny") MACRO(COBCMEM)"
```

```
    pull cobc_mstr
    pull cobc_post
    pull cobc_pre
    mem1 = word(cobc_mstr,2)
    var1 = mem1 '='
    poscnt = words(cobc_post)
    poschk = word(cobc_post,2) - word(cobc_post,4)
    If cpyexp = 'Y' then
      do u = 5 to poscnt
        posnum = ''
        var1   = 'posval ='
        posnum = word(cobc_post,u)
        interpret var1 posnum
        if datatype(posval,N) then
          do
            poschk = poschk + posval - 1
          end
      end
    prechk = ø
    if pdo ¬= 'Y' then
      do
        if cobc_pre ¬= 'NULL' then
          do
            precnt = words(cobc_pre)
            prechk = word(cobc_pre,2) - word(cobc_pre,4)
            If cpyexp = 'Y' then
              do u = 5 to precnt
                prenum = ''
                var1   = 'preval ='
                mstnum = word(cobc_pre,u)
                interpret var1 prenum
                if datatype(preval,N) then
                  do
                    prechk = prechk + preval - 1
                  end
              end
          end
      end
    mstchk = poschk + prechk
    sl = sl + 1
    /* store the source values in array srclist */
    srclist.sl = mem1 mstchk
  end
  /* allocate the ouput dataset */
  call def_output
  /* write out source dataset values to report */
  call write_output
  "FREE DSNAME("outdsn")"
return
/* read in the copybook info. store dataset */
```

```
input_cbs:
  'ALLOC DA('cbsdsn') F(CBSDD) SHR REUSE'
  'EXECIO * DISKR  CBSDD  (STEM memlist. FINIS'
  if rc = Ø then
    if memlist.Ø > Ø then
      do
        cp = memlist.Ø
        do ml = 1 to memlist.Ø
          var1 = word(memlist.ml,1)
          var1 = var1 "="
          var2 = word(memlist.ml,2)
          interpret var1 var2
        end
      end
  "FREE DSNAME("cbsdsn")"
return
/* write out the copybook info. store dataset */
output_cbs:
  'ALLOC DA('cbsdsn') F(CBSDD) OLD'
  'EXECIO ' cp ' DISKW  CBSDD   (STEM memlist. FINIS'
  "FREE DSNAME("cbsdsn")"
return
/* check the validity of the copybook info store dataset */
check_cbsdsn:
  rc = MSG('OFF')
  SYSDSN = SYSDSN(cbsdsn)
  select
    when ( SYSDSN = 'OK' ) then
      do
        nop
      end
    when ( SYSDSN = 'DATASET NOT FOUND'  ) then
      do
        ADDRESS TSO
        "FREE DSNAME("cbsdsn")"
        "ALLOC DD("cpybkddn") NEW CATALOG REUSE ",
        "DSN("cbsdsn") ",
        "LRECL(8Ø) BLKSIZE(3272Ø) RECFM(F,B) ",
        "DSORG(PS) ",
        "STORCLAS("stor1") MGMTCLAS("mgmt1")",
        "VOLUME("vol1") UNIT("unit1")",
        "SPACE("spc1") TRACKS"
      end
    otherwise
      do
        do qc = 1 to queued()
          pull .
        end
        say 'copybook store dataset error'
        say SYSDSN
```

```
        exit 8
      end
  end
  "FREE DSNAME("cbsdsn")"
  rc = MSG('ON')
return
/* check the validity of the copybook datasets */
check_cpydsn:
  rc = MSG('OFF')
  SYSDSN = SYSDSN(cpydsn)
  select
    when ( SYSDSN = 'OK' ) then
      do
        nop
      end
    otherwise
      do
        do qc = 1 to queued()
          pull .
        end
        say 'copybook dataset error for ' cpydsn
        say SYSDSN
        exit 8
      end
  end
  "FREE DSNAME("cpydsn")"
  rc = MSG('ON')
return
/* check the validity of the source dataset */
check_srcdsn:
  rc = MSG('OFF')
  SYSDSN = SYSDSN(srcdsn)
  select
    when ( SYSDSN = 'OK' ) then
      do
        nop
      end
    otherwise
      do
        do qc = 1 to queued()
          pull .
        end
        say 'source dataset error for ' srcdsn
        say SYSDSN
        exit 8
      end
  end
  "FREE DSNAME("srcdsn")"
  rc = MSG('ON')
return
```

```
def_output:
  rc = MSG('OFF')
  SYSDSN = SYSDSN(outdsn)
  select
    when ( SYSDSN = 'OK' ) then
      do
        ADDRESS TSO
        "DELETE "outdsn
        call def_output
      end
    when ( SYSDSN = 'DATASET NOT FOUND'  ) then
      do
        ADDRESS TSO
        "ALLOC DD("outddn") NEW CATALOG REUSE ",
        "DSN("outdsn") ",
        "LRECL(8Ø) BLKSIZE(3272Ø) RECFM(F,B) ",
        "DSORG(PS) ",
        "STORCLAS("stor2") MGMTCLAS("mgmt2")",
        "VOLUME("vol2") UNIT("unit2")",
        "SPACE("spc2") TRACKS"
      end
    otherwise
      do
        do qc = 1 to queued()
          pull .
        end
        say 'output datset error'
        say SYSDSN
        exit 8
      end
  end
  rc = MSG('ON')
return
/* write out the report to output dataset and spool */
write_output:
  totalcnt = Ø
  comline = "---------------------------"
  say comline
  queue comline
  say "|  MEMBER  | LINES OF CODE |"
  queue "|  MEMBER  | LINES OF CODE |"
  say comline
  queue comline
  do s = 1 to sl
    line = "| "left(word(srclist.s,1),9,' ')"|"
    line = line right(strip(word(srclist.s,2),'L','Ø'),6,' ') "      |"
    say line
    queue line
    totalcnt = totalcnt + word(srclist.s,2)
  end
```

```
say comline
queue comline
say sl "members with a total of" totalcnt "lines of code"
queue sl "members with a total of" totalcnt "lines of code"
say comline
queue comline
say "Options in effect"
queue "Options in effect"
say comline
queue comline
if pdo = 'Y' then
  do
    say 'Only Code from the PROCEDURE DIVISION counted'
    queue 'Only Code from the PROCEDURE DIVISION counted'
  end
else
  do
    say 'Code from both PROCEDURE and DATA Divisions counted'
    queue 'Code from both PROCEDURE and DATA Divisions counted'
  end
if cpyexp = 'Y' then
  if cbsuse ¬= 'I' then
  do
    say 'Copybooks expanded inline and included in count'
    queue 'Copybooks expanded inline and included in count'
    say 'Copybooks expanded when found in following libraries:'
    queue 'Copybooks expanded when found in following libraries:'
    say comline
    queue comline
    do cb = 1 to cpydsn.Ø
      say cpydsn.cb
      queue cpydsn.cb
    end
    say comline
    if cbsuse = 'O' then
      do
        say 'Copybook information stored in:'cbsdsn
        queue 'Copybook information stored in:'cbsdsn
      end
    if cbsuse = 'U' then
      do
        say 'Copybook information updated in:'cbsdsn
        queue 'Copybook information updated in:'cbsdsn
      end
  end
  else
    do
      say 'Copybook information retrived from:'cbsdsn
      queue 'Copybook information retrived from:'cbsdsn
    end
```

```
  else
    do
      say 'Copybooks not expanded'
      queue 'Copybooks not expanded'
    end
say 'Source library: '
queue 'Source library: '
say  srcdsn
queue  srcdsn
say comline
queue comline
'EXECIO  * DISKW  OUTDDN  (FINIS'
return
```

COBCMEM – REXX EDIT MACRO

```
/* REXX */                     /* required REXX identifier           */
/* Procedure extract count information from a COBOL source member    */
/* builds and returns a string of statistical information to be      */
/* encyphered by other REXX routine                                  */
'ISREDIT MACRO'                /* required EDIT MACRO identifier     */
/* ADDRESS ISREDIT  */         /* set MODE to ISREDIT                */
trace o                        /* trace switch                       */
procdivst= Ø
procdiven= Ø
/* ** MEMBER Name ** */
cobc_mstr = ''
cobc_post = ''
cobc_pre  = ''
'ISREDIT (memname) = MEMBER'
cobc_mstr = cobc_mstr 'MEMBER=' memname
'ISREDIT (bnd1,bnd2) = BOUNDS'
/* ** Total unexpanded lines in member ** */
'ISREDIT (totlns) = LINENUM .ZLAST'
cobc_mstr = cobc_mstr 'totlns=' totlns
/* ** Total blank lines in member ** */
bl8Ø = ' '
bl8Ø = substr(bl8Ø,1,8Ø,' ')    /* make bl8Ø contain 8Ø blanks       */
    "ISREDIT BOUNDS = 1 8Ø"
    "ISREDIT FIND '"bl8Ø"' ALL" /* search for all blank 8Ø's         */
    "ISREDIT (allstr,allline) = FIND_COUNTS"
    "ISREDIT (bnd1,bnd2) = BOUNDS"
    "ISREDIT BOUNDS = 7 7 "
    "ISREDIT FIND '*' ALL" /* search for all blank 8Ø's         */
    "ISREDIT (comstr,comline) = FIND_COUNTS"
    "ISREDIT BOUNDS = " bnd1 bnd2 ""
allline = allline + comline
cobc_mstr = cobc_mstr 'alllbk=' allline
procsplt = Ø
```

```
'ISREDIT SEEK "PROCEDURE DIVISION" FIRST'  /* establish split position*/
seekrc = rc
if seekrc = 4 then
  do
    cobc_pre = 'NULL'
  end
  else
  do
    'ISREDIT (procsplt) = LINENUM .ZCSR'
    /* set labels */
    'ISREDIT LABEL 'procsplt' = .END Ø'
    'ISREDIT LABEL .ZFIRST = .START Ø'
    lines = procsplt
    linecnt = 'LINECNT=' lines
    exp_str = ''
    call exp_zone
    cobc_pre = exp_str
  end
/* set labels */
procsplt = procsplt + 1
'ISREDIT LABEL 'procsplt' = .START Ø'
'ISREDIT LABEL .ZLAST = .END Ø'
lines = totlns  - ( procsplt - 1 )
linecnt = 'LINECNT=' lines
exp_str = ''
call exp_zone
cobc_post = exp_str
'ISREDIT CANCEL'
queue cobc_mstr
queue cobc_post
queue cobc_pre
exit
exp_zone:
  'ISREDIT CURSOR = .START'
  exp_str = exp_str linecnt
  bl8Ø = ' '
  bl8Ø = substr(bl8Ø,1,8Ø,' ')
  "ISREDIT BOUNDS = 1 8Ø"
  "ISREDIT FIND '"bl8Ø"' .START .END ALL"
  "ISREDIT (allstr,allline) = FIND_COUNTS"
  "ISREDIT (bnd1,bnd2) = BOUNDS"
  "ISREDIT BOUNDS = 7 7 "
  "ISREDIT FIND '*' .START .END ALL"
  "ISREDIT (comstr,comline) = FIND_COUNTS"
  "ISREDIT BOUNDS = " bnd1 bnd2 ""
  allline = allline + comline
  exp_str = exp_str 'alllbk=' allline
  do forever
    cpy_txt = " COPY "
```

```
    'ISREDIT FIND "'cpy_txt'" .START .END'
    findcp_rc = rc
    if findcp_rc = 4 then leave
    'ISREDIT (lcontent) = LINE .ZCSR '
    wp = wordpos('COPY',lcontent)
    wp = wp + 1
    copy_mem = word(lcontent,wp)
    copy_mem = strip(copy_mem,'T','.')
    copy_mem = strip(copy_mem,'B',' ')
    exp_str = exp_str    copy_mem
  end
return
```

BATCHPDF – REXX GENERATOR

This is written by Doug Nadel and is available from http://www.sillysot.com/mvs/.

Rolf Parker
Systems Programmer (Germany)

JCL tips – part 1

When you finish coding your programs you want to put them to work in production, but before that you want to fine tune them. For batch programs, you want to reduce the run-time because the batch window can be very short. What you need to know is how to write good JCL.

Over many years of working with JCL, I have been in situations where I did not know, or was not sure, how to solve some problems. This article is about some of the JCL commands and parameters that we all know or have heard of, but, when we need them, we find we've forgotten them.

JCL CHECKING FOR ERRORS

Before you deploy your jobs in the production environment you

at least want to know that your JCL is correctly written. During testing, you can use the TYPRUN parameter in the JOB statement. One of the available values for this parameter is SCAN, which requests that the system scan this job's JCL for syntax errors, without executing the job or allocating devices:

```
//TEST JOB (ACCT),'PRGMR',CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID,
//         TYPRUN=SCAN
//*
```

This parameter asks the system to check for:

- Spelling of parameter keywords and some subparameter keywords that are not valid.
- Characters that are not valid.
- Unbalanced parentheses.
- Misplaced positional parameters on some statements.
- In a JES3 system only, parameter value errors or excessive parameters.
- Invalid syntax on JCL statements in catalogued procedures invoked by any scanned EXEC statements.

TYPRUN works on the whole job, but, if you want to check just a particular step, you can do it by using a special value (JCLTEST or JSTTEST) on the PGM parameter of the EXEC statement, as in the following example:

```
//TEST     JOB (ACCT),'PRGMR',CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=JCLTEST
//INFILE1  DD DSN=APPLID.INPFILE.TST,DISP=SHR
//REPORT   DD SYSPRINT=A
//SYSPRINT DD SYSPRINT=*
//*
```

The system does not check for misplaced statements, for invalid syntax in JCL subparameters, or for parameters and/or subparameters that are inappropriate together. If you want to check for these possible errors too, you might need to have an additional utility installed. A good example of such a utility is

JCLPREP from Allen Systems Group (ASG). JCLPREP can find a great many errors you can make in your JCL, since it can:

- Perform JCL checking (the same as TYPRUN or JCLTEST above).
- Enable catalog look-up for file validation (file existence).
- Perform checking for member existence in dataset libraries (programs, CNTLs, …).
- Perform program lookup for JOBLIB/STEPLIB statements.
- Perform in-stream and catalogued PROC validation.
- Validate the presence of UNIT/SMS parameters for NEW files.
- Perform security checking.
- Perform your own rules for such statements as JOB, EXEC, DD, JCLLIB, PROC, and OUTPUT.

JCLPREP can run in the foreground as well as in the background as a separate batch job. Its report has a list of all error messages on the top and then each error message is repeated in the place where it occurs.

FILE DELETION

When you have batch processing, you find that you must deal with the deletion of files. The usual reason for deleting the file is that you do not want your job to fail, because the file was already created in a previous cycle and you want to keep only the last version of the file.

The deletion of the file must be done in a fast and easy way. The easiest way to do it is to use the IEFBR14 program, as in the following example:

```
//TEST     JOB (ACCT),'PRGMR',CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IEFBR14
//DELFILE1 DD DSN=APPLID.DELFILE1.TST,DISP=(MOD,DELETE,DELETE)
```

```
//DELFILE2 DD DSN=APPLID.DELFILE2.TST,DISP=(MOD,DELETE,DELETE)
//*
```

The problem you may have is if your file has been migrated, using DFHSM. If it is still on DASD then the retrieval will be quick and so will deletion. But if the file is migrated to tape, then your job is slowed down by the fact that it needs to wait for the file to be retrieved, plus time is spent waiting for a free tape unit and so on. The same thing occurs if your file is already on tape. Deletion by IEFBR14 asks for the tape to be mounted and then the file can be deleted. The solution is to use another way of deleting:

```
//TEST     JOB (ACCT),'PRGMR',CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//STEP1    EXEC PGM=IDCAMS
//SYSIN    DD *
DELETE APPLID.DELFILE1.TST
DELETE APPLID.DELFILE2.TST
IF MAXCC = 8 THEN SET MAXCC = Ø
     //*
```

What it is doing is attempting to delete a file directly from the catalog, without asking for the file to be available – it can be migrated or on tape already. A little detail like this can save some time.

DYNAMIC JCL – USING PROCEDURES AND SET AND INCLUDE STATEMENTS

There can be situations when you know only the structure of your job, but you do not know exactly the values of some of the parameters, or you do not know even the name of the input file, values for some DD statement parameters, or how many files there are and what they are for (eg with SORT), etc.

In these situations, if it is possible, developers usually try to make as many jobs as they need, or they make jobs by using JCL procedures (using the combination of PROC and PEND statements), or they use JCL symbols – as I will explain here.

JCL symbols are something like JCL variables. You can define them, use them, change their values, and use them again. Use of these symbols is simple substitution of the specified value

wherever the symbol is found in the job or procedure. There is also a group of system symbols and maybe everyone is familiar with the SYSUID symbol, which is usually used in a JOB statement's NOTIFY parameter to alert a user when a job is finished.

There are two ways you can define a JCL symbol. Since symbols are heavily used by JCL procedures, symbols are defined as input parameters to the JCL procedure, as in:

```
//PROC1 PROC MIDNAME='STREAM1',
//           CYCLE='Ø3Ø1Ø2'
//*
//STEP1   EXEC PGM=IEFBR14
//DELFILE DD DSN=APPLID.&MIDNAME..D&CYCLE,
//           DISP=(MOD,DELETE,DELETE)
//*
// PEND
```

In this simple procedure (defined in USER.APPLID.PROCLIB) we defined two JCL symbols, MIDNAME and CYCLE, that can further define the name of the file we want to delete in STEP1. Symbols are used in the body of the procedure by placing the character & in front of the symbol name. You need to add the character dot (.) after the symbol name if the following character is $, @, #, or dot itself (like in the example for MIDNAME).

Now you can have a job like this:

```
//RUNJCL   JOB (ACCOUNT),CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//SEARCH JCLLIB ORDER=USER.APPLID.PROCLIB
//*
//STEP1  EXEC PROC1
//*
```

By executing this job, we will delete file APPLID.STREAM1.D031101. However, if we want to delete some other files too, we can write something like:

```
//RUNJCL   JOB (ACCOUNT),CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//SEARCH JCLLIB ORDER=USER.APPLID.PROCLIB
//*
//STEP1  EXEC PROC1
//*
```

```
//STEP1  EXEC PROC1,MIDNAME='STREAM2'
//*
//STEP1  EXEC PROC1,MIDNAME='STREAM1',
//           CYCLE='030101'
//*
```

With this job we will delete three files: APPLID.STREAM1.D031102, APPLID.STREAM2.D031102, and APPLID.STREAM1.D031101. By noting a symbol's value in the EXEC statement, we actually override the values set in the procedure itself.

In the situation above, we got one level of dynamic JCL. By using SET and INCLUDE statements we can get even more. By using the INCLUDE statement we can keep a portion of our JCL in some library as a member. This gives us the opportunity (combined with the SET statement, or with the already-defined symbols in a procedure) to have different jobs every time we execute it.

Let's say we have an application that has an output file whose name we do not know until run-time; for instance, we have to create a file for a specific customer, but we only find out which one during run-time. In that case, our program, which determines the name of the file, can write out that name (or a part, as here) in some PDS library, as in the following example:

```
//* MEMBER CUSTNAME IS DEFINED IN USER.APPLID.INCLUDE
//FMN SET MIDNAME='CUST3'
```

Now we can have the following job, which further processes this file:

```
//RUNJCL JOB (ACCOUNT),CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//SEARCH JCLLIB ORDER=USER.APPLID.INCLUDE
//*
//INCL1    INCLUDE MEMBER=CUSTNAME
//*
//STEP1    EXEC PGM=PGM1
//INPF     DD DSN=APPLID.&MIDNAME..TST,DISP=SHR
//REPORT   DD SYSOUT=*
//SYSPRINT DD SYSOUT=*
//*
```

The only request here is that member CUSTNAME must be defined (created, updated) in a separate job, not in a step in the

same job, because INCLUDE and SET statements are evaluated at the beginning of the job.

It is not unusual for some programs to fail because a file they use is not defined as expected. Differences in record length or record format can cause a problem. We like to be sure that all jobs use the same description – record length, destination, record format, etc – so we can avoid any unpleasant surprises in production. In that case, we can have something like this:

```
//* MEMBER OUTFILE IS DEFINED IN USER.APPLID.INCLUDE
//OUTF DD DSN=APPLID.&MIDNAME..D&CYCLE,DISP=(NEW,CATLG,DELETE),
//        DCB=(LRECL=430,RECFM=VB),SPACE=(CYL,(100,25),RLSE)
```

In the JCL procedure we can have:

```
//* THIS PROC IS DEFINED IN USER.APPLID.PROCLIB
//PROC1  PROC,MIDNAME='STREAM',
//       CYCLE='030101'
//*
//STEP1    EXEC PGM=IDCAMS
//INPF     DD DSN=APPLID.INPFILE.D&CYCLE,DISP=SHR
//INCL1    INCLUDE MEMBER=OUTFILE
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
     REPRO INFILE(INPF) OUTFILE(OUTF)
//*
```

In the job we can have:

```
//RUNJCL   JOB (ACCOUNT),CLASS=B,MSGCLASS=Z,NOTIFY=&SYSUID
//*
//SEARCH JCLLIB ORDER=(USER.APPLID.PROCLIB,USER.APPLID.INCLUDE)
//*
//STEP10  EXEC PROC1,MIDNAME='STREAM1',
//             LASTNAME='STEP10'
//*
//STEP20  EXEC PROC1,MIDNAME='STREAM1',
//             LASTNAME='STEP20'
//*
```

There are two output files, APPLID.STREAM1.STEP10 and APPLID.STREAM1.STEP20, but we are sure that they are defined with the same parameters so we cannot have any problems.

EXECUTE OTHER JOBS

The most common way that applications process data is by using batch job streams. They consist of as many jobs as an application needs. Usually there are three major parts; we need to:

- Prepare the data for processing.
- Process the data.
- Create some reports about processed data.

Of course, each of these steps can have more than one job. The problem is to make them organized as a whole and to make life easier for operators. It is even more important when we know that these job streams are running through the night when people are not so good at concentrating.

All the reasons above (plus many others) were enough to start us thinking about tools that we could use as automated job schedulers/managers, and today there are many of them (eg CA7, ASAP, BETA 42, etc). But these tools are not free, plus there is the cost of maintenance, time to educate staff, etc.

You can easily replace them by using IF/THEN/ELSE/ENDIF statements as well as by calling JOB procedures that come with MVS. It is possible that newer versions of MVS do not have this very useful procedure so you need to check it first in SYS1.PROC library (or some similar system library).

So whenever you have a job stream and one job triggers another, you can have that 'trigger' embedded in the job that precedes it – like in the following example:

```
//BINDPGM JOB ACCOUNT,PEDJA,MSGCLASS=Z,CLASS=B,NOTIFY=&SYSUID
//*
//*****************************************
//*   STEP1 IN JOB6
//*****************************************
//STEP1    EXEC PGM=PROG1
//INPFILE1 DD DSN=APPLID.INPFILE1.TST,DISP=SHR
//OUTFILE1 DD DSN=APPLID.OUTFILE1.TST,DISP=(NEW,CATLG),
//            SPACE=(CYL,(10,5),RLSE),
//            DCB=(LRECL=80,RECFM=FB,BLKSIZE=0)
//SYSPRINT DD SYSPRINT=*
```

```
//*
//*************************************
//*   IF STEP1 ENDS WITH RC=Ø START JOB7
//*************************************
//IF1      IF  ( STEP1.RC=Ø ) THEN
//CALL1    EXEC JOB,D='APPLID.JCLLIB',N=JOB7
//IF1      ENDIF
//*
//*************************************
//* STEP2 IN JOB6
//*************************************
//STEP2    EXEC PGM=PROG2
//INPFILE2 DD DSN=APPLID.INPFILE2.TST,DISP=SHR
//OUTFILE2 DD DSN=APPLID.OUTFILE2.TST,DISP=(NEW,CATLG),
//            SPACE=(CYL,(1Ø,5),RLSE),
//            DCB=(LRECL=8Ø,RECFM=FB,BLKSIZE=Ø)
//SYSPRINT DD SYSPRINT=*
//*
//*************************************
//*   IF STEP1 ENDS WITH RC=Ø START JOB8
//*************************************
//IF2      IF  ( STEP1.RC=Ø ) THEN
//CALL2    EXEC JOB,D='APPLID.JCLLIB',N=JOB8
//IF2      ENDIF
//*
```

We can see that the CALL to another job can be positioned anywhere in the job, not just after the last step, so you can have even more functionality. By positioning the call at the end of the job you actually continue executing the job stream where the new job starts. By positioning the call in the middle of the job, you start a new job stream that can proceed independently. A similar effect can be achieved by putting both calls at the end of the job, but this way you can save some valuable time.

Importantly, if you need to restart your job you must be careful if that job has IF/THEN/ELSE/ENDIF statements, since the job you want to restart does not need to be the same as the original. If the expression you use in the IF statement uses a step RC that is not part of the restart job any more, you need to change the job to suit the new situation. Usually it is done by keeping either the THEN or ELSE branch – whichever is true at that moment.

TIME

Usually, when system programmers set job classes they also set a default processor time that they think a job in that class will need to run. So we have different job classes – short, night, system, etc. Of course, not everyone is allowed to submit jobs in every class, but usually there are similar classes for everyone. When we run our jobs, we like to find a class that will allow our job to run as soon as possible, especially if we work in a busy environment. Sometimes we choose the wrong job class, so we end up with our job failed – even sometimes without knowing why. Well, it can be because the job exceeded the specified time limit for that job class, and, as a result, we get S322 as the abend code.

However, we can work around this if we use the TIME parameter, in either the JOB or the EXEC statement. In the JOB statement we can set a maximum processor time we need for the whole job, while on the EXEC statement we can specify the time needed for that step only. Here are some simple rules for using the TIME parameter:

- Do not code TIME=0 on a JOB statement. The results are unpredictable.
- If you want to give unlimited time, code TIME=NOLIMIT or TIME=1440.
- Every job step can have its own time limit. The sum of all the values for the TIME parameter on the EXEC statement cannot exceed the default value set by the system or the value for the TIME parameter coded on the JOB statement.
- The step limit is set as the EXEC TIME parameter value, the default time limit, or the job time remaining after the execution of previous steps, whichever is the smaller.

Predrag Jovanovic
Project Developer
Pinkerton Computer Consultants Inc (USA)

Defining page datasets for a new partition without STEPCAT

When defining system datasets in order to create a new MVS partition, you have to create new page datasets.

The ‘traditional’ method uses a STEPCAT to catalog these VSAM datasets in the new master catalog.

```
//STEPØ1  EXEC PGM=IDCAMS
//*
//STEPCAT  DD DISP=SHR,DSN=CATALOG.MCAT.BPØ1
//*
//SYSPRINT DD SYSOUT=*
//*
//DD        DD   DISP=OLD,UNIT=SYSALLDA,VOL=SER=BDØØØ1
//*
//SYSIN    DD *
   DELETE PAGE.BKUP.PLPA     -
                    FILE(DD)
   SET MAXCC = Ø
   DEFINE  PAGESPACE( -
                 NAME(PAGE.BPØ1.PLPA) -
                 VOLUME(BDØØØ1) -
                 FILE(DD) -
                 CYLINDERS(8Ø) -
                     )
/*
```

But the usage of STEPCAT is not the best solution because it:

- Is not supported for SMS volumes.
- Doesn’t work if the UCB of the volume where the catalog is located is loaded > 16 MB line (LOCANY UCB).
- Adversely affects performance.
- Might not be supported at all in the future.

An alternative to STEPCAT is available: the use of the Multi-Level Alias (MLA) feature.

This article describes how to define page datasets for a new partion (BP01) without using STEPCAT but using MLA.

USING MULTI-LEVEL ALIAS (MLA)

Very often the naming convention uses the SYSNAME of the partition in the name of the page datasets, eg :

```
PAGE.&sysname.PLPA
```

So, for the new partition BP01, the names of the page datasets will be:

- PAGE.BP01.PLPA
- PAGE.BP01.COMMON
- PAGE.BP01.LOCAL01

The new master catalog of the partition BP01 (CATALOG.MCAT.BP01) is defined as a user catalog of the driving system.

Enabling the use of MLA

To enable dynamic use of MLA level 2, you should enter the following MVS command:

```
F CATALOG,ALIASLEVEL(2)

IEC351I CATALOG ADDRESS SPACE MODIFY COMMAND ACTIVE
IEC352I CATALOG ADDRESS SPACE MODIFY COMMAND COMPLETED
```

In order to check the result of this command, you should issue the F CALOG,REPORT command:

```
F CATALOG,REPORT

IEC351I CATALOG ADDRESS SPACE MODIFY COMMAND ACTIVE
IEC359I CATALOG REPORT OUTPUT 770
*CAS************************************
*  CATALOG COMPONENT LEVEL   = HDZ11G0   *
*  CATALOG ADDRESS SPACE ASN = 001D      *
*  SERVICE TASK UPPER LIMIT  =  180      *
*  SERVICE TASK LOWER LIMIT  =   60      *
*  HIGHEST # SERVICE TASKS   =   38      *
*  CURRENT # SERVICE TASKS   =   38      *
*  MAXIMUM # OPEN CATALOGS   = 1,024     *
*  ALIAS TABLE AVAILABLE     = YES       *
*  ALIAS LEVELS SPECIFIED    = 2         *          - MLA level
```

```
*  SYS% TO SYS1 CONVERSION   = OFF       *
*  CAS MOTHER TASK           = 009A3920  *
*  CAS MODIFY TASK           = 009A3700  *
*  CAS ANALYSIS TASK         = 009A0E88  *
*  CAS ALLOCATION TASK       = 009A33D8  *
*  VOLCAT HI-LEVEL QUALIFIER = SYS1      *
*  NOTIFY EXTENT             =   10%     *
*  DELETE UCAT/VVDS WARNING  = ON        *
*  DATA SET SYNTAX CHECKING  = ENABLED   *
*CAS************************************
IEC352I CATALOG ADDRESS SPACE MODIFY COMMAND COMPLETED
```

In order to make the change static, you should modify the SYSCAT statement of the LOAD*xx* member of SYS1.IPLPARM:

```
IODF     ** SYS4     OSCONF   00 Y
NUCLEUS  1
NUCLST   00
SYSCAT   PD0001123CCATALOG.MCAT.PROD01
IEASYM   00
*               I - 2 = MLA level = 2
```

Defining alias PAGE.BP01

At this point, on the driving system, you should define an alias for PAGE.BP01:

```
//STEP1    EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DEFINE ALIAS(NAME('PAGE.BP01')   RELATE('CATALOG.MCAT.BP01'))

/*
```

Defining new page datasets

The new page datasets will be allocated, formatted, and cataloged in the correct master catalog:

```
//STEP01  EXEC PGM=IDCAMS
//SYSPRINT DD SYSOUT=*
//SYSIN    DD *
 DEFINE PAGESPACE (NAME(PAGE.BP01.LOCAL) -
                   VOLUME(BD0001)        -
                   CYLINDERS(500))
/*
```

MVS news

IBM has announced new security technology with the latest release of its mainframe operating system, z/OS 1.5, providing the first single point of control for managing a multi-level security environment.

Combined with DB2 UDB for z/OS Version 8, the IBM solution provides multi-level security on the eServer zSeries mainframe to help meet the security requirements of government agencies and financial institutions.

z/OS 1.5 and DB2 V8 enable a single repository of data to be managed at the row level and accessed by individuals based on their need to know. Multi-level security on z/OS can take advantage of eServer zSeries functionality such as robust cryptography, high availability, scalability, and flexibility to provide a highly secure environment.

For further information contact your local IBM representative.
URL: http://www-1.ibm.com/servers/eserver/zseries/announce/zos_r5/.

* * *

Software AG has announced that its Adabas database is able to take advantage of IBM's new 64-bit Shared Virtual Storage technology for the eServer zSeries mainframe.

Software AG has enhanced Adabas to use the 64-bit Shared Virtual Storage area with Adabas Parallel Services (Version 7.5). Adabas Parallel Services runs in SMP environments under a single multi-engine CPU. Parallel Services enables multiple SMP engines to process commands against an Adabas database, providing greater throughput. The multiple SMP engines will now be able to cache data in the 64-bit Shared Virtual Storage, providing improved capacity for cacheing frequently used or recently updated data – thus processing Adabas information more quickly.

For further information contact:
Software AG, 11190 Sunrise Valley Drive, Reston, VA 20191, USA.
Tel: 703.860.5050.
URL: http://www.softwareag.com/corporat/news/feb2004/adabas_64Bit.htm.

* * *

Innovation Data Processing has announced an addition to the FDR /UPSTREAM storage management tool. It's the UPSTREAM Rescuer, which is a stand-alone system recovery facility for Intel Linux, SuSE zLINUX, and Solaris systems.

FDR/UPSTREAM provides enterprise-wide storage management for a variety of platforms using the z/OS or OS/390 MVS mainframe as the back-up server.

UPSTREAM Rescuer allows administrators to completely restore a system from data that is saved in FDR/UPSTREAM without separate system back-ups. FDR/UPSTREAM's Rescuer provides this system recovery supplement because it is completely integrated into FDR/UPSTREAM.

For further information contact:
Innovation Data Processing, 275 Paterson Avenue, Little Falls, NJ 07424, USA.
Tel: (973) 890 7300.
URL: http://www.innovationdp.fdr.com/ups.cfm.