212

CICS update

July 2003

In this issue

CICS 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 *CICS Update*, comprising twelve monthly issues, costs £175.00 in the UK; $270.00 in the USA and Canada; £181.00 in Europe; £187.00 in Australasia and Japan; and £185.50 elsewhere. In all cases the price includes postage. Individual issues, starting with the December 1999 issue, are available separately to subscribers for £16.00 ($24.00) each including postage.

CICS Update on-line

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

Editor

Trevor Eddolls

Contributions

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

A new CEMT program in COBOL to start and stop CICS

CICS is more than 30 years old, but it now supports applications written in C++ and Java, and it allows a single application image to be spread over several computer systems. To achieve such youthful longevity, CICS has undergone many transformations. IBM mainframe products get three levels of version number within a distinct named product. CICS is on its third distinct product name, and its fifth high-level version number. It has been maintained at two different IBM facilities since first being marketed.

An OLTP manages transactions that exhibit the following four ACID properties, as explained in Gray & Reuter's book (1993): *Atomicity, Consistency, Isolation, and Durability*. Although CICS did not have functions like journalling to support all of these properties at first, it did have them very early in its life. Like any OLTP, CICS has to interact with telecommunications networks, database managers, different programming languages, and the features and constraints of operating systems. Another way to look at CICS is that it allows a large number of users to share a relatively small number of resources with data integrity. The first versions of CICS were developed in a former IBM facility near Chicago. In 1973, IBM moved CICS development to Hursley, a village near Winchester, UK, where it has remained ever since.

Today, no on-line system could survive without some way to access the World Wide Web. IBM introduced the CICS Web Interface product in 1996. Since then, there have been various solutions and enhancements to CICS to allow applications to interact with Web browsers. The latest version, CICS Transaction Server for OS/390 Version 2.2, allows legacy application programs to interact with Web browsers as if they were 3270 terminals, and it allows new programs to present a modern interface on Web browsers. To help customers give legacy applications a new look on the World Wide Web, the CICS team created the Front End Programming Interface (FEPI) in CICS/ESA Version 3.3 in 1992.

FEPI allows a CICS transaction to emulate a 3270 terminal. The idea is to write a new FEPI transaction to sit between a legacy CICS application and a Web server. That way, the legacy application, which was written for 3270 terminals, can continue to run unchanged.

I wrote a simple COBOL program to start and to stop CICS.

HOW TO INSTALL XCEM CEMT

Submit the following job to compile the source program:

```
//SRSTXCEM JOB SRS1ØØ44,SYSTEMES,CLASS=G,MSGCLASS=T,NOTIFY=&SYSUID
//         EXEC  DFHC3LCL,
//*        INDTG=ESS,
//*        DEBUG=OUI,
//         SP=',SP',
//         OUTC=T
//LIBR.SYSIN DD *
-OPT LIST
-DLM SRSTXCEM
-ADD SRSTXCEM,SEQ=/81,6,1,1/
       IDENTIFICATION DIVISION.
       PROGRAM-ID.   SRSTXCEM.
      ********************** F U N C T I O N *************************
      *                                                              *
      *                         stop CICS                            *
      *                                                              *
      ****************************************************************
      *****************************     ******************************
      *      *cics stop command
      ****************************************************************
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION.
       77  W-DEBUT              PIC X(8) VALUE 'SRSTXCEM'.
       77  W-RETOUR-LIB         PIC X(9) VALUE 'EIBRESP->'.
       77  W-RETOUR             PIC S9(8) COMP VALUE +Ø.
       77  W-EIBFN-LIB          PIC X(9) VALUE ' EIBFN ->'.
       77  W-EIBFN              PIC X(2) VALUE SPACES.
       Ø1  W-ANO-CN.
           Ø5 W-APPLID          PIC X(8).
           Ø5 FILLER            PIC X.
           Ø5  FILLER           PIC X(5Ø) VALUE 'UTILIZATION OF XCEM FOR
      -        ' NON AUTHORIZED USERS ('.
           Ø5  W-USERID         PIC X(8).
           Ø5  FILLER           PIC X(7) VALUE ') TERM('.
           Ø5  W-TERM           PIC X(4).
           Ø5  FILLER           PIC X VALUE ')'.
       Ø1  W-PARAM.
           Ø5 W-TRANS           PIC X(4).
           Ø5 FILLER            PIC X(76) VALUE SPACES.
      *
       PROCEDURE DIVISION.

      *
      *1- ANALYSE  COMMAND .......
      *
       1ØØ-INIT.
           IF EIBTRMID(1:2) NOT = 'CN'
              MOVE EIBTRMID TO W-TERM
              EXEC CICS ASSIGN APPLID(W-APPLID)
              END-EXEC
              EXEC CICS ASSIGN USERID(W-USERID)
              END-EXEC
              EXEC CICS WRITE OPERATOR
                   TEXT(W-ANO-CN)
                   CRITICAL
              END-EXEC
              EXEC CICS RETURN
              END-EXEC
           END-IF
           EXEC CICS RECEIVE INTO(W-PARAM)
                RESP(W-RETOUR)
           END-EXEC
           MOVE 'CEMT' TO W-TRANS
           EXEC CICS XCTL PROGRAM('DFHEMTP')
                INPUTMSG(W-PARAM)
                INPUTMSGLEN(LENGTH OF W-PARAM)
           END-EXEC.

-END
/*
```

XCEM CEMT will run on any CICS from Version 4.1 upwards.

Copy the member XCEM to a loadlib in the DFHRPL list of your desired CICS region.

Enter the following commands, into either CEDA or the batch program DFHCSDUP:

```
DEFINE TRANSACTION(XCEM) GROUP(XCEM) DESCRIPTION(XCEM CEMT)
    PROGRAM(XCEM) TASKDATALOC(ANY) PRIORITY(255)
DEFINE PROGRAM(XCEM) GROUP(XCEM) DESCRIPTION(XCEM CEMT)
    LANGUAGE(ASSEMBLER) DATALOCATION(ANY)
```

The transaction name can be changed, but should start with 'C' to allow it to run at Maxtasks.

Use CEDA to install the group XCEM.

Enter the transaction XCEM and enjoy!

Claude Dunand
Systems Programmer (France)

Exclusive control conflicts and VSAM deadlocks – hints and tips

INTRODUCTION

There are a number of very useful CICS trace entries, produced during CICS file control processing, that can be used to provide a great deal of helpful information, provided that you know what to look for within the (potentially) vast amount of trace data that can be generated by CICS.

This article describes how useful diagnostic information may be gleaned from the problem documentation provided with a deadlock situation (or deadly embrace) within CICS file control. In particular, the CICS trace table provides a considerable amount of useful information, that can be analysed for clues as to the nature and cause of the deadlock. This article also discusses exclusive control conflicts within CICS and gives advice on them.

BACKGROUND TO CICS TRACING

CICS trace data is primarily used for application and system debugging. It records a series of events (in chronological order), showing the state of various system resources and activities. It also reveals the multitasking of user and system tasks within the CICS system, being dispatched and running quasi-reentrantly under the main CICS TCB (the QR TCB). Other TCB task activity (running truly in parallel to the QR TCB) is also recorded.

CICS trace may be written to the internal CICS trace table, which is a wraparound area of storage above the 16MB line within the CICS address space. This can then be examined, from both CICS transaction and system dumps, to reveal the sequence of events that preceded the cause of the dump (typically an abend of some kind). The size of this internal CICS trace table is controlled by the TRTABSZ system initialization parameter, and by the Internal Trace Table Size option on the CETR transaction's main panel.

In addition to the internal CICS trace table, trace entries can be sent to the CICS auxiliary trace destination, which comprises either one or two CICS-managed BSAM datasets. These provide a far larger repository for holding trace information than the internal CICS trace table, and can therefore be used to record a longer time period of CICS system activity. The I/O operations required to implement auxiliary tracing means this option has a greater impact on CICS system performance than when just using internal CICS tracing.

CICS provides a batch utility program to format the auxiliary trace data. The name of the program is CICS-release specific – its suffix is the CICS release number. Therefore, in CICS Transaction Server 1.3 (containing the CICS component CICS/ESA 5.3.0) it is DFHTU530. In CICS Transaction Server 2.2 (containing the CICS component CICS/ESA 620) it is DFHTU620. The batch utility program has a range of options that may be used to selectively filter specific trace entries when formatting the auxiliary trace data, such as filtering by task number, transaction identifier, trace entry number, etc.

The most commonly used options when using the batch utility program are to control the amount of information to be returned in the formatted trace. The options are ABBREV (for abbreviated), SHORT, and FULL. Abbreviated trace shows the minimum data for every trace entry. It is useful for revealing the flow of events through CICS and the commands issued by applications. Short trace is similar to abbreviated trace, but provides additional data such as the time of each trace entry. Finally, specifying full trace gives up to seven data items associated with the trace points. These include parameter lists, names of resources, control blocks, and response and reason codes. Full trace is very useful when a particular series of events has already been investigated, and a problem then needs to be analysed at a trace entry by trace entry level, showing what parameters (and their values) were being utilized at the time.

CICS FILE CONTROL AND EXCLUSIVE CONTROL CONFLICTS

CICS Transaction Server for z/OS V2.2 has enhanced its support for commands against VSAM files. CICS file control processing can now release the exclusive control of VSAM Control Intervals (CIs) after having performed an EXEC CICS READ UPDATE request. In turn, this can avoid the potential for VSAM exclusive control conflicts occurring in the interval between the EXEC CICS READ UPDATE and its subsequent EXEC CICS REWRITE command. This behaviour is the default for EXEC CICS READ UPDATE / EXEC CICS REWRITE commands in CICS TS 2.2.

For further details on this CICS improvement, please refer to my earlier article entitled *CICS file control enhancements*, which appeared in Issue 204 of *CICS Update* (November 2002).

This file control enhancement helps avoid the potential for exclusive control conflicts against the CI in use for the EXEC CICS READ UPDATE and EXEC CICS REWRITE commands. It does not avoid the possibility of such conflicts for other types of API request, however. For example, EXEC CICS WRITE requests can still receive an exclusive control conflict when they attempt to acquire ownership of the CI that is to hold the record being added to the file. As such, exclusive control conflicts will still be seen while CICS is processing application requests against VSAM files. These exclusive control conflicts still result in CICS issuing unsolicited exception trace entries (trace point AP 04BA) for diagnostic purposes. These trace points can be useful when investigating persistent exclusive control conflict situations. One example where they can be seen is during a deadlock between two or more tasks within the CICS system, where each task has a resource that the other wants. Such a deadlock situation can be analysed and understood by analysis of the CICS trace entries recorded by the tasks in question. An example of such a deadlock analysis for a CICS TS 1.3 region is given below.

PROBLEM DETERMINATION DEADLOCK EXAMPLE

The following set-up was used for this example. Two user tasks (FC01 and FC02) executed two programs (ANDYPGM1 and ANDYPMG2). These two programs issued EXEC CICS READ UPDATE commands against two VSAM KSDS files (PRODANDY and TESTANDY). Both were defined as non-RLS files.

The FC01 and FC02 transactions were both initiated at similar times, and so their activity overlapped within the system. The first evidence of a problem was when FC02 abended with an AFCF abend code. The following messages were issued by CICS:

```
DFHDUØ2Ø3I A transaction dump was taken for dumpcode:
                                                AFCF, Dumpid: 1/ØØØ2.

DFHAC2236   Transaction FCØ2 abend AFCF in program ANDYPGM2 term V11D.
```

The CICS *Messages and Codes* manual (or CMAC transaction) explanation for an AFCF abend is as follows: a deadlock has been detected between two or more tasks issuing file control requests. It also states that when transactions update several files within the same unit of work, all transactions should update these files in the same order. This is a good clue that the file access pattern for the two programs is probably not being carried out in the same order.

The associated transaction dump for task FC02 contained trace entries pertaining to this task, and its activity prior to the AFCF abend. The most interesting trace entry is the unsolicited exception trace point AP 04B8. If this is examined using FULL trace, so that its associated data items are returned, then useful information regarding the deadlock can be determined. In particular, data item 3 contains details of the task that caused FC02 to be abended AFCF. In this case, the details were:

```
AP Ø4B8 FCVS *EXC* - DEADLOCK_DETECTED
                     TASK-ØØØ42 KE_NUM-ØØ34 TCB-QR             =ØØ1398=

TASKNUM :   ØØØ41C

TRANSID :   FCØ1

FCTNAME :   PRODANDY
```

Data item 4 of the trace entry shows the VSAM Work Area (VSWA) for the conflicting task. This contains the VSAM Request Parameter List (RPL) imbedded at offset 8 into the VSWA, and may be used to determine further information about this task's request to VSAM. Data item 6 shows the File Request Thread Element (FRTE) for the abending task's request. These DFHVSWA and DFHFRTE control blocks are described in the CICS *Data Areas* manual and may be examined for additional information, such as the FRT_FUNCTION byte, which defines the type of request.

This exception trace entry has confirmed that the AFCF abend was caused by a deadlock situation with transaction FC01, task 00041. It also confirms that the abended transaction (FC02) was running as task 00042 within the system. (It is possible that no transaction dump is obtained for an abend. It is also a possibility that CICS trace is set off at the time of the abend. In these circumstances, the AP 04B8 exception trace may be the only useful piece of diagnostic information to accompany the abended task. As can be seen, however, with just this one exception trace entry, useful information pertaining to the deadlock can be identified.)

Assuming trace was active in the system, the events leading up to the AFCF abend can be examined. Since deadlock events typically relate to several tasks coexisting within a CICS system, it may be the case that a system dump is requested to accompany a recurrence of such an abend. This could be achieved by setting an entry in the dump table, by the command:

```
CEMT S TRD(AFCF) SYS
```

By taking a system dump, task storage and trace entries for all the tasks in the system at the time of the abend may be examined.

Now that the two tasks involved in this abend have been identified, the trace may be examined for just these particular task numbers (so avoiding a great many unrelated task entries). In the case of a system dump, the appropriate VERBX command to achieve this for the example AFCF abend being investigated would be:

```
VERBX DFHPD53Ø 'TR=1,TRS=<TASKID=(ØØØ41,ØØØ42)>'
```

and in the case of an auxiliary trace being formatted by the batch utility DFHTU530, the command would be:

```
ABBREV,TASKID=(ØØØ41,ØØØ42)
```

Using such commands, the appropriate trace entries can be returned, and are shown below (note that these have been edited for ease of reference):

```
ØØØ41 QR PG Ø9Ø1 PGPG  ENTRY INITIAL_LINK     ANDYPGM1        =ØØØ1Ø4=
ØØØ41 QR AP ØØE1 EIP   ENTRY READ                             =ØØØ11Ø=
ØØØ41 QR AP Ø4EØ FCFR  ENTRY READ_UPDATE_INTO PRODANDY        =ØØØ111=
ØØØ41 QR AP Ø4E1 FCFR  EXIT  READ_UPDATE_INTO/OK              =ØØØ341=
ØØØ41 QR AP ØØE1 EIP   EXIT  READ             OK              =ØØØ342=

ØØØ42 QR PG Ø9Ø1 PGPG  ENTRY INITIAL_LINK     ANDYPGM2        =ØØØ411=
ØØØ42 QR AP ØØE1 EIP   ENTRY READ                             =ØØØ417=
ØØØ42 QR AP Ø4EØ FCFR  ENTRY READ_UPDATE_INTO TESTANDY        =ØØØ418=
ØØØ42 QR AP Ø4E1 FCFR  EXIT  READ_UPDATE_INTO/OK              =ØØØ529=
ØØØ42 QR AP ØØE1 EIP   EXIT  READ             OK              =ØØØ53Ø=

ØØØ41 QR AP ØØE1 EIP   ENTRY READ                             =ØØ1162=
ØØØ41 QR AP Ø4EØ FCFR  ENTRY READ_UPDATE_INTO TESTANDY        =ØØ1163=
ØØØ41 QR AP Ø4B7 FCVS  *EXC* VSAM          EXCEPTION VSAM RPL =ØØ1168=
ØØØ41 QR AP Ø4BA FCVS  *EXC* WAIT_FOR_EXCLUSIVE_CONTROL       =ØØ1171=
ØØØ41 QR DS ØØØ4 DSSR  ENTRY WAIT_OLDC FCXCWAIT,TESTANDY      =ØØ1172=

ØØØ42 QR AP ØØE1 EIP   ENTRY READ                             =ØØ1389=
ØØØ42 QR AP Ø4EØ FCFR  ENTRY READ_UPDATE_INTO PRODANDY        =ØØ139Ø=
ØØØ42 QR AP Ø4B7 FCVS  *EXC* VSAM          EXCEPTION VSAM RPL =ØØ1395=
ØØØ42 QR AP Ø4B8 FCVS  *EXC* DEADLOCK_DETECTED                =ØØ1398=
ØØØ42 QR AP Ø4E1 FCFR  EXIT  READ_UPDATE_INTO                 =ØØ1399=
ØØØ42 QR AP 2ØØØ PCPG  ENTRY ABEND                            =ØØ14ØØ=
ØØØ42 QR DU Ø1Ø1 DUDU  ENTRY TRANSACTION_DUMP AFCF            =ØØ14Ø8=

ØØØ41 QR DS ØØØ5 DSSR  EXIT  WAIT_OLDC/OK                     =ØØ2541=
ØØØ41 QR AP Ø4E1 FCFR  EXIT  READ_UPDATE_INTO/OK              =ØØ2544=
ØØØ41 QR AP ØØE1 EIP   EXIT  READ                  OK         =ØØ2545=
```

It can be seen that task 00041 issued an EXEC CICS READ UPDATE command against file PRODANDY (see trace entry =000110=). This completed normally (the EIP EXIT trace entry =000342= has the qualifier OK). The task then lost control, and task 00042 was dispatched by CICS. This issued an EXEC CICS READ UPDATE command against file TESTANDY (trace entry =000417=). Again, this completed normally. Since this is not a CICS TS 2.2 region, CICS does not release ownership of the CI upon completion of the READ UPDATE command. Therefore, both tasks 00041 and 00042 retain ownership of the CI containing the particular records they have read for update purposes.

Task 00041 is next to be redispatched. It now issues another EXEC CICS READ UPDATE command, but this time against file TESTANDY (trace entry =001162=). This does not complete normally, however. VSAM returns an exception condition of Exclusive Control Conflict, since the CI containing the record to be updated by task 00041 is currently owned by the RPL for task 00042. CICS documents this fact by issuing two unsolicited exception trace entries – AP 04B7 and AP 04BA (trace entries =001168= and =001171=). The EXEC CICS READ UPDATE request cannot complete because of this, hence there is no corresponding EIP EXIT trace point issued at this stage. Instead, task 00041 is made to wait for the CI to become available. This results in a CICS dispatcher wait, for resource type FCXCWAIT (Exclusive Control Conflict wait) at trace entry =001172=.

Task 00042 is redispatched once more. This too issues another EXEC CICS READ UPDATE command, this time for file PRODANDY. Again, VSAM returns an exception condition of Exclusive Control Conflict, and CICS issues exception trace AP 04B7 (=001395=). However, if CICS were to suspend task 00042 in the same manner as task 00041, the resulting deadlock could not be satisfied since both tasks would be suspended, waiting for the other to release the appropriate CI for them to continue.

Therefore, CICS checks whether a deadlock situation would occur if it were to suspend task 00042. Given that it would, CICS documents this (by the exception trace AP 04B8 examined earlier) then takes steps to avoid the deadlock by abending the task with an AFCF abend. The abend code is shown on the call to CICS Dump Domain to request the associated transaction dump for the AFCF abend (trace entry =001408=).

As part of abend processing, CICS will ensure that all resources owned by task 00042 are released (and, if recoverable, their state restored to its previously committed value). This means that an Endreq is issued to release ownership of the CI within file TESTANDY. Once this has occurred, CICS is able to redispatch task 00041, since the CI is now available for use by other tasks. Task 00041 is able to acquire the CI, and its EXEC CICS READ UPDATE of file TESTANDY is able to complete successfully – the EIP EXIT trace entry for this shows OK at trace entry =002545=.

OBSERVATIONS

This particular example of a deadlock resulted from bad application implementation, since the same resources were being updated by concurrent tasks in different orders. Such an approach can lead to the possibility of a deadlock occurring.

In the worked example given above, there were two tasks that together conspired to produce the deadlock. However, in a more complex case, it may be that there are more than two participant tasks. The basic problem determination approach as outlined above can still be applied. In particular, the exception trace entries can be used to determine which tasks are waiting for Exclusive Control Conflicts to be resolved, and which tasks are causing AFCF abends because of CICS avoiding deadlock situations from taking place. Since CICS trace may be set off at the time of the abend, the exception trace entry data may be vital in determining the cause of the failure. Both the AP 04B8 and AP 04BA exception trace points contain the useful diagnostic information as their data item 3; with just this information, the task and its RPL address that has caused this conflict may be seen.

As mentioned above, the CICS TS 2.2 default behaviour means that CICS TS 2.2 will release ownership of a VSAM CI upon completion of an EXEC CICS READ UPDATE command, and reacquire CI ownership at the time of the corresponding EXEC CICS REWRITE. This means that Exclusive Control Conflicts are less likely to occur for such an environment at that release of CICS.

Had the files been defined to use RLS (VSAM Record Level Sharing services), the concept of Exclusive Control Conflicts would not apply. CICS does provide alternative diagnostic information to give details on potential deadlocking situations and lock delays with RLS files, however. An AFCV abend is issued if a request made against a file opened in RLS mode was unable to acquire a record lock. It waited for the lock, but the wait time exceeded the maximum wait time applicable to that request. An abend AFCW is issued if a file control request against a file opened in RLS mode is one deemed to lead to a deadlock situation if allowed to proceed. The transaction is abended in order to break the deadlock chain.

An abend AFCV is associated with messages DFHFC0164 and DFHFC0165, giving further details about the nature of the request timing out waiting for an RLS lock. Similarly, an abend AFCW is associated with messages DFHFC0166 and DFHFC0167, which describe the deadlocked environment that was detected prior to the abend.

A ‘LIVELOCK’ SITUATION

CICS TS 1.3 APAR PQ68163 (PTF UQ72904) and CICS TS 2.2 APAR PQ68847 (PTF UQ72640) recently modified CICS file control processing to avoid a problem in the area of exclusive control conflict management. In certain circumstances, multiple tasks within a CICS system may repeatedly fail with exclusive control conflicts if a VSAM Control Area (CA) split is required. For example, if a number of tasks attempt to add records into a VSAM KSDS file with record keys that reside within the same CA, then it may not be possible to complete a CA split if one is required. This situation was observed when adding records to a file with a large number of alternate indexes, and with timestamps comprising part of the record keys. The similar keys can result in a CA split, but this may fail because of exclusive control conflicts from the other tasks performing similar requests against the same CI(s). The repeated redispatch of the various tasks performing the EXEC CICS WRITE commands to the file resulted in a ‘moving deadlock’ (or ‘livelock’) within the CICS system, with task throughput being impeded and CICS being unable to process new work satisfactorily.

To avoid this situation, CICS has been modified to avoid posting other waiting tasks when a task fails with an exclusive control conflict. Instead, they will remain suspended until the RPL is able to complete without an Exclusive Control Conflict. Only then will the chain of tasks waiting for exclusive control be posted and allowed to retry their request once more. This avoids the problem situation described above.

FURTHER READING

For CICS/ESA 4.1.0, the product’s trace entries were documented in the CICS *Diagnosis Handbook*. For CICS Transaction Server Release 1.1 and Release 1.2, they were documented in the CICS *User’s Handbook*. For CICS Transaction Server Release 1.3 and Release 2.2, the number of trace entries (and their associated data entries) has grown sufficiently to warrant a complete manual entitled *CICS Trace Entries*.

SUMMARY AND CONCLUSIONS

I hope that this article has given some useful guidance on approaching a deadlock situation within CICS File Control processing, in particular by focusing upon the useful information contained within the associated CICS trace entries.

Andy Wright (andy_wright@uk.ibm.com)
CICS Change Team
IBM (UK)

QMF goes on-the-fly to Adobe Acrobat Reader, MS Word, MS Excel, … – part 2

This month we conclude the article looking at connecting CICS and QMF on a mainframe to Acrobat Reader, MS Word, or MS Excel.

```
* COMPUTE LENGTH FOR LAST COLUMN
                   MOVE NB-COLS TO I
                   SUBTRACT COL-START(I) FROM LONGUEUR
                      GIVING COL-LENGTH(I)
                END-IF
* IF DASHED LINE HAS BEEN FOUND AND REPORT LINE IS NOT BLANK ...
                IF COL-START(1) > ZEROES AND
                   TEMP = Ø AND
                   TEMP2 NOT EQUAL SPACES THEN
                   MOVE LOW-VALUES TO TEMP1
                   MOVE '"' TO TEMP1(1:1)
* CONCATENATE COLUMNS WITH ","
                   PERFORM VARYING I FROM 1 BY 1
                           UNTIL I > NB-COLS
                      IF I < NB-COLS THEN
                         STRING TEMP1 DELIMITED BY LOW-VALUES
                                TEMP2(COL-START(I):COL-LENGTH(I))
                                      DELIMITED BY '      '
                                '","' DELIMITED BY SIZE
                                INTO TEMP1
                         END-STRING
                      ELSE
* CONCATENATE COLUMNS WITH " FOR LAST FIELD
                                TEMP2(COL-START(I):COL-LENGTH(I))
                                      DELIMITED BY '      '
                                '"' DELIMITED BY SIZE
                                INTO TEMP1
                         END-STRING
                      END-IF
                   END-PERFORM
* CONCATENATE LINE IN COMMAREA
                   STRING DFHCOMMAREA DELIMITED BY LOW-VALUES
                          TEMP1       DELIMITED BY LOW-VALUES
                          X'15'       DELIMITED BY SIZE
                          INTO DFHCOMMAREA
                   END-STRING
                END-IF
             END-IF
     END-PERFORM.
```

```
* CLEAN COMMAREA
     INSPECT DFHCOMMAREA REPLACING ALL LOW-VALUES BY SPACES.
*
* BUILD THE HTML DOCUMENT USING QMF REPORT
*
 Ø1-BUILD-QMFHTML.
* BUILD HTML DOCUMENT
     MOVE LOW-VALUES TO TEMP1.
     PERFORM UNTIL EIBRESP > Ø OR EIBRESP2 > Ø
             MOVE 133 TO LONGUEUR
             MOVE SPACES TO TEMP2
* READ THE QMF REPORT FROM TS QUEUE
             EXEC CICS READQ QUEUE(USERID-EXPORT)
                             INTO(TEMP2)
                             LENGTH(LONGUEUR)
                             NOHANDLE
             END-EXEC
* CONCATENATE RECORD IN WORKING WITH X'15' (LINE FEED) AT THE END
             IF EIBRESP = ZEROES AND EIBRESP2 = ZEROES THEN
                STRING TEMP1 DELIMITED BY LOW-VALUES
                       TEMP2 DELIMITED BY '                     '
                       X'15'  DELIMITED BY SIZE
                       INTO TEMP1
                END-STRING
             END-IF
     END-PERFORM.
* CLEAN WORKING AND WRITE IT TO COMMAREA
     INSPECT TEMP1 REPLACING ALL LOW-VALUES BY SPACES.
     MOVE TEMP1 TO DFHCOMMAREA.
* PUT HTML DOCUMENT INTO COMMAREA
     MOVE TEMP1 TO COMMAREA.
```

GENERATING DYNAMIC PDF DOCUMENTS

If you are not a C/C++ expert, you may think that using PDFlib will be difficult. This is not the case. In fact, what we should do to build a PDF document is:

1 Open the document (even if the document is built in memory).

2 Set some properties (creator, date, …).

3 Begin a page.

4 Choose the font.

5 Write the text.

6 Repeat step 5 if necessary.

7 End the page.

8 Repeat steps 3 to 6 if needed.

9 Close the document.

Look at the following program:

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#pragma filetag("IBM-5ØØ")
#include "PDFlibdl.h"
/*-------------------------------------------------------------------*/
/* Program : QMFPDF                                                  */
/* Date        : Ø9/2ØØ2                                             */
/* Author    : Delaunoy P.                                           */
/*                                                                   */
/* Functions :                                                       */
/*  ========                                                         */
/* 1) Read the QMF report in the TSQ queue                           */
/* 2) Load the PDFLIB routines                                       */
/* 3) Build the PDF document                                         */
/* 4) Begin a new page if a char '1' is found in column 1 (ASA)      */
/* 5) Close the document and put it in commarea                      */
/* 6) Unload the PDFLIB routines                                     */
/*-------------------------------------------------------------------*/
/* COMMAREA declaration */
struct com_struct  {
   unsigned char PDF_fichier[325ØØ];
} *commarea ;
main(void)
{
    PDF *p;
    Int font;
    PDFlib_api *PDFlib = PDF_boot_dll();
    Char *PDF_buf;
    Int page_nbr;
    Char userid[8];
    Short int longueur;
    Signed long int PDF_longueur;
    Signed long int resp;
    Signed long int resp2;
    Char qmf_row[256];
    char asa="1";
 /* get addressability to the EIB */
    EXEC CICS ADDRESS EIB(dfheiptr);
 /* access common area sent from caller */
```

```
    EXEC CICS ADDRESS COMMAREA(commarea);
 /* get the userid */
    EXEC CICS ASSIGN USERID(userid) NOHANDLE;
/* check PDFLIB routines */
    if (PDFlib == NULL)
    {
       fprintf(stderr, "Error: couldn't load PDFlib DLL.\n");
       return Ø;
    }
   p = PDFlib->PDF_new();
/*--------------------------------*/
/* 1 Open the document */
/*--------------------------------*/
    if (PDFlib->PDF_open_file(p, "") == -1)
    {
       fprintf(stderr, "Error: cannot open PDF file.\n");
       exit(2);
    }
/*----------------------------------------*/
/* 2 set document properties */
/*----------------------------------------*/
    PDFlib->PDF_set_info(p, "Creator", "CICS");
    PDFlib->PDF_set_info(p, "Author", "QMF 3.3");
    PDFlib->PDF_set_info(p, "Title", "QMF Report");
    Resp=Ø;
    Resp2=Ø;
    Page_nbr=1;
/* read printed report in temporary storage queue */
    while (resp == Ø  && resp2 == Ø) {
       memcpy(&qmf_row," ",255);
       longueur=133;
       EXEC CICS READQ TS QUEUE(userid)
                          INTO(qmf_row)
                          LENGTH(longueur)
                          RESP(resp)
                          RESP2(resp2)
                          NOHANDLE
                          ;
/* new page requested ? */
       if (memcmp(&qmf_row,&asa,1) == Ø) {
/*-------------------*/
/* 7 End a page */
/*-------------------*/
          if (page_nbr > 1) PDFlib->PDF_end_page(p);
          page_nbr += 1;
/*-----------------------*/
/* 3 Begin a page */
/*---------------------*/
          PDFlib->PDF_begin_page(p, a4_width, a4_height);
/*------------------------*/
/* 4 Choose the font */
/*------------------------*/
          font = PDFlib->PDF_findfont(p, "Helvetica-Bold", "host", Ø);
          PDFlib->PDF_setfont(p, font, 1Ø);
/*---------------------*/
/*5 Write the text */
/*---------------------*/
          PDFlib->PDF_set_text_pos(p, 1ØØ, 5ØØ);
          PDFlib->PDF_show(p,(const char *)qmf_row);
       }
       else {
          PDFlib->PDF_continue_text(p,(const char *)qmf_row);
       }
    }
/*-------------------------------*/
/*9 Close the document */
/*-------------------------------*/
   PDFlib->PDF_end_page(p);   /* close page */
   PDFlib->PDF_close(p);   /* close PDF document */
/* Get PDF document from memory */
   PDF_longueur=Ø;
   PDF_buf=(char *)PDFlib->PDF_get_buffer(p,&PDF_longueur);
/* copy PDF document in commarea */
    memcpy(commarea,PDF_buf,(size_t)PDF_longueur);
/* delete the PDF object */
    PDFlib->PDF_delete(p);
/* unload the library */
PDF_shutdown_dll(PDFlib);
/*end */
return Ø;
}
```

Code page translation problems

PDFlib generates PDF documents directly in binary. The problems is that HTTP request handler performs, by default, an EBCDIC to ASCII code page translation, because all the data on a mainframe is in EBCDIC. But with a PDF document generated by PDFlib, you must tell the Web server not to perform this translation, since the document is binary.

Please read the Novation Enterprise Server V4.6 documentation or the CICS Web support guides for more details.

GENERATING A DYNAMIC WORD DOCUMENT

As far as I know, there is no way to generate a .doc file dynamically because the data in this file is mainly binary. The solution is to use an RTF file, which is a text file, as you can see:

```
{\rtf1\ansi\ansicpg1252\deffØ\deftab72Ø{\fonttbl{\fØ\fswiss MS Sans
Serif;}{\f1\froman\fcharset2 Symbol;}{\f2\froman Times New Roman;}}
{\colortbl\redØ\greenØ\blueØ;}
\deflang2Ø6Ø\horzdoc{\*\fchars }{\*\lchars }\pard\plain\f2\fs2Ø
Put your QMF report here. \par }
```

To put your QMF report in an MS Word document, you just have to:

- Encode the document above in the COMMAREA.
- Execute your QMF report and print it in a temporary storage queue.
- Read the temporary storage queue and insert it in the COMMAREA.

This solution is quite simple but has the disadvantage of mixing the presentation of the data within a program. So, if you want to modify the document, you have to recompile the program. Another solution is to use a template manager which merges a template file with the QMF report. The template file is created with MS Word and is a standard document with some special tags. The template manager uses those tags to substitute them with the QMF report. For more details, see template manager in the Novation documentation.

GENERATING A DYNAMIC EXCEL DOCUMENT

Like .doc files, as far as I know, there is no way to generate dynamically XLS documents. The solution is to use CSV files. CSV stands for comma separated value. To create the CSV files, you put commas between each field of the QMF report to delimit them. Unfortunately for people who don’t live in the USA or UK, the decimal separator is the comma. So, if the number 3,1415 appears in a CSV file, it will be split into two fields (3 and 1415). That’s why it’s highly recommended to surround the fields with quotes, like "3,1415".

To surround the fields with quotes and commas, you must know what are the fields' starting positions and lengths. One solution could be to export the form (not the report!) in temporary storage and to analyse it. This could be quite tedious. I prefer to search in the separator line, just below the column headings, the spaces between the dashes. The length of each field is given by the difference between the current position and the previous space position.

CHARTS AND IMAGES

PDFlib has a full set of graphics and image routines to create charts and pictures in a PDF document. You can even create animated images. See the PDFlib reference guide.

THE 32KB BARRIER

The CICS maximum DFHCOMMAREA length is 32,767 bytes and the generated document must still comply with this rule. But CICS Web Support and Novation Enterprise Server can run in pointer mode – instead of passing a COMMAREA, you may pass a pointer to it using the template manager. For more details, see the *CICS Internet Guide* or the Novation documentation.

CONCLUSIONS

Although this way to access DB2 data might not be the most efficient one, it enables you to build MS Word, MS Excel, and Adobe Acrobat reader files from existing reports, without any coding modification. It's even possible, if you have the proper algorithm, to create from QMF reports:

- XML files
- TXT files
- ZIP files
- JPEG, BMP files
- Other.

This technique is not limited to QMF reports. You can also create dynamic MS Word documents with VSAM data. But always keep in mind that HTML, PDF, RTF, XLS, etc file standards don't support binary, packed, and non-displayable data.

Pierre Delaunoy
Director
Ministère de la Communauté Française (Belgium)

CICSPlex/System Manager Report Writer

After working with CPSM it became very apparent that there was a wealth of information in the CPSM maintained dataspaces. All this is apparent from the CPSM Resource Tables and readily available using the CPSM API. The CPSM EUI (End User Interface) ISPF Dialog only goes so far in providing access to this information. It is useful, but in my opinion has many shortcomings. I found myself wanting quick access to some of the information that may or may not be included in the tabular dialog screens. This was compounded by the fact that there was no printing ability. Almost all the information is buried in the detail screens, but the purpose of having all this real-time information is defeated if you can't access it quickly and by your own rules. The new Web User Interface (WUI) deals with some of these issues, but View and Viewset changes are required to provide new views. In some shops this can be a change control nightmare and I was looking for a 'quick and dirty' way to get an answer.

I started playing with the CPSM REXX API and was very pleased with the flexibility in accessing the data. The API is reminiscent of SQL. You connect to CPSM, formulate a query against a CPSM object, establish your filter criteria, determine your sort sequence, execute your query, obtain the result set, and fetch through the results processing each returned record as needed, then disconnect from CPSM. Although there isn't an SQL SELECT statement and a WHERE clause, the concepts are very similar. The only major functionality that is missing is a ‘join’ between two or more tables. This must be handled within your program if needed.

All CPSM objects are defined in the *CPSM Resource Table Reference* manual. Each object is described as a table of data columns. Each column can be displayed and used as a selection criterion or as a sorting key. As with any application programming task, the hardest part is deciding what it is you really want to see. After that, it is simply a case of picking and choosing the columns from the appropriate table. There are object tables for just about every CICS resource you can think of (CICS regions, terminals, connections, programs, transactions, journals, Temp Storage queues, etc).

After getting my feet wet with a few small REXX EXECs, I found there were common activities I was performing in all programs. This led to the creation of several REXX functions to do things like Connect, Get, Order, Get_Meta_Data, etc. I later found that most of my programs were to inquire on things in a single Resource Table even though the CPSM REXX API is fully capable of using the full CEMT SPI to issue performs and sets. I also realized that most of my programs were basically clones with a few minor changes based on the Resource Table columns.

This started me working on a generic CPSM Report Writer, which would allow me and others to access any CPSM Resource Table and create a report against a single table. My design criteria were to allow a single REXX EXEC to be run in batch with parameters and DD statements used to influence the processing. The required Context, Scope, Object, and optional CMAS name would be provided as execute card parameters. All other features would be influenced by the usage of special DDnames in the JCL. The idea was that very simple JCL could be run for very basic requests. As the requests become more complicated, the additional features could be ‘turned on’ by adding the necessary DDs and creating the input datasets containing the appropriate input.

I was also trying to avoid any need to modify the CPSMRW REXX

EXEC. The only things in the EXEC that you might want to tailor are:

- The default page_size, currently set to 55 lines.
- The default CMAS name, which currently resolves to C&SYSCLONE.XCMAS in the EXEC.

To change the default page_size, just find 'page_size = 55' in the CPSMRW REXX EXEC and change it to the desired value. As you will see later, page_size is modifiable within each report heading.

The default CMAS name can be set in the REXX EXEC or in the RWPROC. If you have a good naming convention for your CMASs in a Sysplex, the CPSMCMAS subroutine (find 'cpsmcmas:' in the EXEC) can be changed to incorporate the logic necessary to identify the correct CMAS based on the LPAR the CPSMRW job is running on. A simple REXX SELECT clause can handle this.

```
select
   when mvsvar('sysname') = 'SYS1' then cmas = 'CMAS1'
   when mvsvar('sysname') = 'SYS2' then cmas = 'CMAS2'
   otherwise cmas = 'ERROR'
end
```

As you will see later, the RWPROC does provide the &CMAS symbolic that can be set in every CPSMRW job.

The program's 'advanced' features are:

- User-definable list of Resource Table variables for the print line on the report (report view).
- Multiple user definable report views.
- User-definable heading and/or preprocessing logic.
- User-definable filters for selective reporting.
- User-definable multi-level sort sequence.

CPSMRW was developed and tested with CICS/TS 1.3 and CPSM 1.4 under OS/390 V2R10.

PROC JCL

I created a PROC to execute the CPSMRW REXX EXEC called RWPROC. The PROC allows for easy customization without bothering the casual user with the details.

```
//CPSMRW   PROC CONTEXT=PRODCPLX,               CPSM CONTEXT
//*****************************************************************
//* PROC: CPSMRW                                                  *
//*                                                               *
//* PURPOSE: EXECUTE THE CPSM REXX API TO RUN BASIC REPORTS       *
//*                                                               *
//* SYMBOLICS:                                                    *
//*                                                               *
//* CONTEXT - THE CPSM CONTEXT                                    *
//* SCOPE   - THE CPSM SCOPE                                      *
//* OBJECT  - THE CPSM OBJECT                                     *
//* CMAS    - THE TARGET CMAS TO CONNECT TO                       *
//* CPSMLOAD- THE CPSM LOADLIB                                    *
//* OBJLIB  - THE PDS THAT HOLDS THE OBJECT VIEW MEMBERS          *
//* EXECLIB - THE PDS THAT HOLDS THE REXX EXECS                   *
//* REPORT  - SYSOUT CLASS FOR OPERATOR OUTPUT                    *
//* DEBUG   - SYSOUT CLASS FOR DEBUGGING OUTPUT                   *
//*                                                               *
//* NOTES:                                                        *
//*                     CHANGE LOG                                *
//*****************************************************************
//         SCOPE=PROD,                          CPSM SCOPE
//         OBJECT=CICSRGN,                      CPSM OBJECT
//         CMAS=,                               LOCAL CMAS
//         CPSMLOAD='CPSM.PROD.CTS13Ø.SEYUAUTH', CPSM LOADLIB
//         EXECLIB='RZENUK.CPSMRW',             REXX EXEC PDS
//         OBJLIB='RZENUK.CPSMRW',              CPSM OBJECT PDS
//         REPORT='T',                          REPORT OUTPUT
//         DEBUG='*'                            DIAGNOSTIC MESSAGES
//*****************************************************************
//* BATCH TSO STEP TO EXECUTE THE CPSMRW REXX EXEC                *
//*****************************************************************
//CPSMRW   EXEC PGM=IKJEFTØ1,
//         PARM='CPSMRW &CONTEXT &SCOPE &OBJECT &CMAS'
//STEPLIB  DD   DSN=&CPSMLOAD,DISP=SHR
//SYSEXEC  DD   DSN=&EXECLIB,DISP=SHR
//SYSTSPRT DD   SYSOUT=&REPORT
//SYSTSIN  DD   DUMMY
//*****************************************************************
//* PDS TO HOLD CPSMRW MEMBERS FOR VIEWS                          *
//*****************************************************************
//OBJECT   DD   DSN=&OBJLIB,DISP=SHR
//DIAGMSGS DD   SYSOUT=&DEBUG
```

PROC tailoring instructions

If all items are placed in the same PDS, change the EXECLIB and OBJLIB to point to this PDS. Set CPSMLOAD to your CPSM LOADLIB. Set CONTEXT and SCOPE to valid defaults. The CPSMRW EXEC will reason out a CMAS name, but this may not be what you want, so you may want to set a default for CMAS also.

All the code examples included can be placed in members in a single PDS.

EXECUTE JCL

To execute the PROC use the following JCL:

```
//MYCPSM   JOB  (ØØØØ),'R ZENUK',MSGCLASS=T,REGION=ØM,CLASS=O
//*****************************************************************
//* BASIC EXAMPLE OF RUNNING CPSMRW                               *
//*****************************************************************
/*JOBPARM SYSAFF=*
//*****************************************************************
//* PROCLIB WHERE CPSMRW PROC LIVES                               *
//*****************************************************************
//         JCLLIB ORDER=(MY.CPSMRW)
//*****************************************************************
//* EXECUTE CPSMRW                                                *
//*****************************************************************
//CPSMRW   EXEC RWPROC
```

Using the PROC from above, tailored ‘as is’, this will run a query against the CPSM CICSRGN object using Context PROD and Scope PROD.

Another example:

```
//MYCPSM   JOB  (ØØØØ),'R ZENUK',MSGCLASS=T,REGION=ØM,CLASS=O
//*****************************************************************
//* BASIC EXAMPLE OF RUNNING CPSMRW WITH SYMBOLIC OVERRIDES     *
//*****************************************************************
/*JOBPARM SYSAFF=*
//*****************************************************************
//* PROCLIB WHERE CPSMRW PROC LIVES                               *
//*****************************************************************
//         JCLLIB ORDER=(MY.CPSMRW)
//*****************************************************************
//* EXECUTE CPSMRW                                                *
```

```
//*****************************************************************
//CPSMRW   EXEC RWPROC,SCOPE=OTORS,OBJECT=CONNECT
```

Using the PROC from above, this will list all connections defined to the scope of OTORS (a sample Scope that contains only TORs).

Another example:

```
//MYCPSM   JOB  (ØØØØ),'R ZENUK',MSGCLASS=T,REGION=ØM,CLASS=O
//*****************************************************************
//* BASIC EXAMPLE OF RUNNING CPSMRW WITH SYMBOLIC OVERRIDES       *
//* ADDING A FILTER AND SORT ORDER                                *
//*****************************************************************
/*JOBPARM SYSAFF=*
//*****************************************************************
//* PROCLIB WHERE CPSMRW PROC LIVES                               *
//*****************************************************************
//         JCLLIB ORDER=(MY.CPSMRW)
//*****************************************************************
//* EXECUTE CPSMRW                                                *
//*****************************************************************
//CICSRGN  EXEC RWPROC,OBJECT=PROGRAM
//*****************************************************************
//* FILTER DD TO ADD FILTER CRITERIA                              *
//*****************************************************************
//FILTER   DD   DSN=MY.CPSM(PFILTER),DISP=SHR
//*****************************************************************
//* ORDER DD TO ADD SORT CRITERIA                                 *
//*****************************************************************
//ORDER    DD   DSN=MY.CPSM(PORDER),DISP=SHR
```

Using the PROC from above, this will list all selected programs using the filter criteria in member PFILTER and sort the results using the sort sequence specified in member PORDER.

And finally, here is an example with multiple comment reminders for all the features:

```
//MYCPSM   JOB  (ØØØØ),'R ZENUK',MSGCLASS=T,REGION=ØM,CLASS=O
//*****************************************************************
//* BASIC EXAMPLE OF RUNNING CPSMRW WITH COMMENTED REMINDERS      *
//*****************************************************************
/*JOBPARM SYSAFF=*
//*****************************************************************
//* PROCLIB WHERE CPSMRW PROC LIVES                               *
//*****************************************************************
//         JCLLIB ORDER=(MY.CPSMRW)
//*****************************************************************
```

```
//* EXECUTE CPSMRW                                                 *
//******************************************************************
//CICSRGN  EXEC RWPROC
//******************************************************************
//* ADD AN OVERRIDE DD TO USE DIFFERENT PRINT LINE VARIABLES       *
//******************************************************************
//*CICSRGN  DD   DSN=MY.CPSM(CICSRGN1),DISP=SHR
//******************************************************************
//* USE THE 'NODETAIL' OVERRIDE WHEN ONLY TOTALS ARE NEEDED        *
//******************************************************************
//*CICSRGN  DD   DSN=MY.CPSM(NODETAIL),DISP=SHR
//******************************************************************
//* UNCOMMENT THE FILTER DD TO ADD FILTER CRITERIA                 *
//******************************************************************
//*FILTER   DD   DSN=MY.CPSM(PFILTER),DISP=SHR
//******************************************************************
//* UNCOMMENT THE HEADING DD TO ADD A HEADING                      *
//******************************************************************
//*HEADING  DD   DSN=MY.CPSM(HEADING),DISP=SHR
//******************************************************************
//* UNCOMMENT THE ORDER DD TO ADD SORT CRITERIA                    *
//******************************************************************
//*ORDER    DD   DSN=MY.CPSM(PORDER),DISP=SHR
//******************************************************************
//* UNCOMMENT THE GROUP DD TO ADD GROUP CRITERIA                   *
//******************************************************************
//*GROUP    DD   DSN=MY.CPSM(PGROUP),DISP=SHR
```

CPSMRW PROGRAM DOCUMENTATION

CPSMRW is written in REXX and uses the CPSM EYUAPI REXX Function package. The REXX EXEC is completely self-contained and does not require any other modules to execute. It uses several internal subroutines, but has no external dependencies.

CPSMRW has four required parameters:

- Context – valid CPSM Context. Default value is PRODCPLX.
- Scope – valid CPSM Scope. Default value is PRODCPLX.
- Object – valid CPSM Object Name. Default value is CICSRGN.
- CMAS – valid CMAS on the local system. Default value is C*nn*XCMAS (where *nn*=&SYSCLONE).

CPSMRW has several DD statements used to influence its operation:

- DIAGMSGS (required) – CPSMRW utility messages like parameter values, input DD contents, etc.
- OBJECT (required) – the default location for the 'View' member for the CPSM Object print line (usually REXX SAY statements).
- FILTER (optional) – CPSM Filter criteria.
- ORDER (optional) – CPSM Order criteria.
- GROUP (optional) – CPSM Group criteria.
- HEADING (optional) – column headings and/or start-up logic (usually REXX SAY statements).
- object (optional) – DD used to override the contents of the OBJECT DD – the name will be the same as the CPSM Object identified in the Object parameter.

USAGE RULES AND GUIDELINES FOR EACH DD

DIAGMSGS

The DIAGMSGS DD is used to record all program messages and any inputs that may be useful for diagnosing problems. Always look here when diagnosing a problem and make sure the SYSOUT for DIAGMSGS is actually displaying the values you believe you are using. This helps identify simple JCL errors when member names or datasets are misspelled or mistakenly changed.

OBJECT

The OBJECT DD must point to a PDS that contains a member with the same name as the CPSM object identified in the PARM. This member is the default print line for reports against the selected CPSM object. This may be thought of as the default 'view'.

Contents of OBJECT DD members will be valid REXX statements that will be interpreted when needed. This ‘exit’ gets control at the end of the Fetch/Parse loop and was primarily envisioned for producing a print line. The minimum required is SAY statements for the CPSM Resource Table variables desired in the output. During CPSMRW FETCH and TPARSE processing all Resource Table variables are retrieved and prefixed with the characters ‘obj_’. Therefore, all Resource Table variable names in REXX SAY statements must be prefixed with ‘obj_’.

Example of a valid MAS member to produce a simple MAS report:

```
say obj_cicsname obj_pricmas obj_cmasname obj_cicsstate obj_desc
```

If a print line must span two lines, you must build a single print line variable. Normal REXX continuation will *not* work because the contents of the OBJECT member are interpreted one line at a time using the REXX INTERPRET statement. Here is an example of creating a ‘longer’ print line:

```
obj_line1 = obj_cicsname obj_pricmas obj_cmasname
obj_line2 = obj_cicsstate obj_desc
say obj_line1 obj_line2
```

Another example shows some additional processing of variables to format them in a more ‘report-friendly’ manner. It is also acceptable to include valid REXX comments in the OBJECT members:

```
/*********************************************************************/
/* Print line for CICSRGN CPSM Object                                */
/*********************************************************************/
tasks  = right(strip(obj_currtasks),3)
status = left(strip(obj_cicsstatus),1Ø)
obj_line1 = obj_jobname obj_applid status tasks obj_eyu_cicsrel
obj_line2 = obj_startup obj_strttime obj_mvssysid
say obj_line1 obj_line2
```

If no print line is desired, just the default total number of objects matching the criteria, you can provide an OBJECT member with a comment or NOP statement:

```
/* no print line */
nop
```

The OBJECT PDS is not limited to the traditional 80 byte LRECL; any size up to 32KB will work.

FILTER

The optional FILTER DD, if used, will point to a syntactically valid FILTER CRITERIA. The EXEC will add the required trailing period (fullstop) '.'. The syntax must be a REXX assignment statement and the variable 'filter' must be set to a valid CPSM FILTER. Filters can contains multiple AND, OR, and parenthetical expressions. See the CPSM *Application Programming Guide* for examples of valid FILTER CRITERIA.

Example of a FILTER:

```
filter = 'TRANID¬=C* AND RUNSTATUS=SUSPENDED'
```

The same rules apply regarding continuations, additional REXX code, and LRECL as described above with the OBJECT DD. This 'exit' gets control before the Fetch/Parse loop.

ORDER

The optional ORDER DD, if used, will point to a syntactically valid ORDER CRITERIA. The EXEC will add the required trailing period (full stop) '.'. The syntax must be a REXX assignment statement and the variable 'sort' must be set to a valid CPSM ORDER. Orders can be ascending or descending. See the *CPSM Application Programming Guide* for examples of valid ORDER CRITERIA.

Example of an ORDER:

```
sort = 'PROGRAM,USECOUNT/D'
```

The same rules apply regarding continuations, additional REXX code, and LRECL as described above with the OBJECT DD. This 'exit' gets control before the Fetch/Parse loop.

GROUP

The optional GROUP DD, if used, will cause a GROUP BY to occur based on the contents of the GROUP DD. The EXEC will add the required trailing period (fullstop) '.'. The syntax must be a REXX assignment statement. The variable 'group' must specify a valid resource table attribute for the grouping. The variable 'sumopt' is used to apply summary techniques to the result set (AVG, DIF, LIKE, MAX, MIN, SUM). The summary value will be placed in the table attribute identified. If GROUP is used and SUMOPT is not used, default summary processing will occur based on the CPSM Resource Table specifications.

Example of a GROUP:

```
group = 'TRANID'
sumopt = 'SUM(USECOUNT)'
```

The same rules apply regarding continuations, additional REXX code, and LRECL as described above with the OBJECT DD. This 'exit' gets control before the Fetch/Parse loop.

HEADING

The optional HEADING DD, if used, will contain syntactically valid REXX statements to produce a heading and/or any 'pre' processing desired before the detail print lines.

Example of a HEADING:

```
say '1Page:' page_num
say
say ' CICS Name     MVS ID       Tasks       APPLID'
```

It is also possible to set the page_size variable to change the number of lines per page. The program default is 55 lines and is driven only if a HEADING DD exists in the JCL. This causes the contents of the HEADING DD to be re-driven after 'page_size' lines. Carriage control is the user's responsibility. The best technique for this is placing a '1' in the first position of your REXX SAY statement for the heading line and leaving the first position blank in the print detail lines from the OBJECT or object DD. The page_num variable is also maintained and can optionally be use in your heading. If you use carriage control, don't forget to leave a blank in column 1 for the detail lines in the 'view' member. The SYSTSPRT DD will also need to interpret the ANSI Carriage

Control in column 1. This can be accomplished by coding your SYSTSPRT DD as follows:

```
//SYSTSPRT  DD  SYSOUT=*,RECFM=VBA,LRECL=125,BLKSIZE=129
```

The same rules apply regarding continuations, additional REXX code, and LRECL as described above with the OBJECT DD. This ‘exit’ gets control before the Fetch/Parse loop.

object

The optional object DD, if used, will point to an alternative member containing SAY statements. This allows you to maintain a PDS with multiple report formats for the same object. The name of the DD will be the same as the CPSM object to be printed. This is an override for the OBJECT DD to allow multiple members in the same PDS to contain print lines for the same CPSM object.

The syntax is the same as the OBJECT DD. This can also be used with a ‘nop’ member to eliminate details lines and simply print totals.

Examples of an object DD:

```
//CICSRGN  DD   DSN=your.view.pds(CICSRGN2),DISP=SHR
//CICSRGN  DD   DSN=your.view.pds(NODETAIL),DISP=SHR
```

Member NODETAIL can contain a single line with ‘nop’ or a comment

The same rules apply regarding continuations, additional REXX code, and LRECL as described above with the OBJECT DD. This ‘exit’ gets control before the Fetch/Parse loop.

RECOMMENDATIONS

Here are some helpful guidelines to follow when developing new CPSMRW reports (or any reports for any report writer for that matter):

- Always start with a working model if you have one.

- Avoid ‘eating the elephant’ – cut it up into small bite-size pieces then eat it slowly.
- When working through any report writer project, use a small working set of data, don’t try to process millions of rows for every interim run. Not only does this waste resources, but also makes each of your runs very long and will extend your project time. In CPSM terms, limit your SCOPE.
- Ensure that your subset of data provides the conditions you are trying to report on. Also, try to make sure your subset contains a wide array of conditions so your small subset exercises your report.
- When developing from scratch, start simple by first displaying the target object and desired columns. This approach allows you to debug one problem at a time.
- After seeing the rough report, arrange and space the columns as needed and add a heading as well as page breaks.
- After seeing your desired object displayed with the columns in the desired sequence, add your filter(s).
- After seeing your desired subset, add your sort criteria.
- If needed, add any required grouping criteria.
- Have fun, amaze your friends, and go home with a sense of accomplishment!

SAMPLE REPORTS

```
REPORT OUTPUT from SYSTSPRT

-------------------- CPSMRW started 29 May 2ØØ2 23:29:52 -------------

62 CICSRGN objects found

CICSName APPLID   Status   Tasks Rel SYSID

PICSP1AA XIP1AA   ACTIVE      12 E53Ø TS11
PICSP1AB XIP1AB   ACTIVE      12 E53Ø TS11
PICSP1AC XIP1AC   ACTIVE      12 E53Ø TS11
PICSP1AD XIP1AD   ACTIVE      12 E53Ø TS11
```

```
PICSP1AE XIP1AE   ACTIVE      12 E53Ø TS11
PICSP1AF XIP1AF   ACTIVE      12 E53Ø TS11
```

UTILITY MESSAGES FROM DIAGMSGS

```
CPSM modules executed from CICS.PROD.CTS130.SEYUAUTH
CPSMRW executed from RZENUK.CPSMRW
Parms: PRODCPLX OTORS CICSRGN
Connected to C01XCMAS on TS01 29 May 2002 23:29:52
Unconnected from C01XCMAS on TS01 29 May 2002 23:29:53
```

OBJECT PRINT LINE USED FROM CICSRGN DD MY.CPSMRW

```
/*******************************************************************/
/* Print line for CICSRGN CPSM Object                             */
/*******************************************************************/
 tasks  = right(strip(obj_currtasks),3)
 status = left(strip(obj_cicsstatus),1Ø)
 say obj_jobname obj_applid status tasks obj_eyu_cicsrel obj_mvssysid
```

HEADING USED FROM HEADING DD RZENUK.CPSMRW

```
/*****************************************************************/
/* sample heading line for a CICSRGN view                       */
/*****************************************************************/
 page_size = 5Ø
 say 'CICSName APPLID   Status   Tasks Rel  SYSID'
```

CPSMRW

```
/********************************************************************/
/*                                REXX                             */
/********************************************************************/
/* Purpose: Produce a batch report from CPSM                       */
/*-----------------------------------------------------------------*/
/* Syntax:  CPSMRW context scope object                            */
/*-----------------------------------------------------------------*/
/* Parms: context    - CPSM Context (defaults to PROD)             */
/*        scope      - CPSM Scope (defaults to PROD)               */
/*        object     - CPSM Object (defaults to CICSRGN)           */
/*        cmas       - CPSM CMAS (if default not desired)          */
/*-----------------------------------------------------------------*/
/* Sample JCL                                                      */
/*                                                                 */
/* //CPSMRW   EXEC PGM=IKJEFTØ1,DYNAMNBR=99,                       */
/* //         PARM='CPSMRW context scope object cmas'              */
/* //STEPLIB  DD   DSN=cpsm.seyuauth,DISP=SHR                      */
/* //SYSEXEC  DD   DSN=exec.pds,DISP=SHR                             */
/* //SYSTSPRT DD   SYSOUT=*                                          */
/* //DIAGMSGS DD   SYSOUT=*                                          */
/* //SYSTSIN  DD   DUMMY                                             */
/* //OBJECT   DD   DSN=your.view.pds,DISP=SHR                        */
/* //FILTER   DD   DSN=your.filter.pds(member),DISP=SHR  (optional)  */
/* //ORDER    DD   DSN=your.order.pds(member),DISP=SHR   (optional)  */
/* //GROUP    DD   DSN=your.group.pds(member),DISP=SHR   (optional)  */
/* //HEADING  DD   DSN=your.heading.pds(member),DISP=SHR (optional)  */
/* //object   DD   DSN=your.view.pds(member),DISP=SHR    (optional)  */
/*                                                                   */
/* If FRC is set to 9999 (FRC = 9999), all FETCH processing will be  */
/* bypassed.  This is useful for benchmarking CPSM GET processing    */
/* against various CPSM objects.                                     */
/*********************************************************************/
/*                      Change Log                                   */
/*********** @REFRESH BEGIN START    2ØØ2/Ø9/11 2Ø:26:53 *************/
/* Standard startup activities                                       */
/*********************************************************************/
 call time 'r'
 parse arg parms
 signal on syntax name trap
 signal on failure name trap
 signal on novalue name trap
 probe = 'NONE'
 modtrace = 'NO'
 modspace = ''
 call stdentry 'DIAGMSGS'
 module = 'MAINLINE'
 push trace() time('L') module 'From:' Ø 'Parms:' parms
 if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
 call modtrace 'START' Ø
/*********** @REFRESH END   START    2ØØ2/Ø9/11 2Ø:26:53 *************/
/* Set default page_size                                             */
/*********************************************************************/
 page_size = 55
 page_num = 1
/*********************************************************************/
/* Accept input parms                                                */
/*********************************************************************/
 arg parms
/*********************************************************************/
/* Parse parms                                                       */
/*********************************************************************/
 parse var parms context scope object cmas .
 if context = '' then context = 'PROD'
 if scope = '' then scope = 'PROD'
 if object = '' then object = 'CICSRGN'
 if cmas = '' then cmas = cpsmcmas()
/*********************************************************************/
```

```
/* Determine if an override DD was provided, if so use it            */
/*********************************************************************/
 if listdsi(object "FILE") = Ø then
    do
     objdd = object
    end
 else
/*********************************************************************/
/* No override, use the default OBJECT DD                            */
/*********************************************************************/
    do
     objdd = 'OBJECT'
     call ddcheck objdd
/*********************************************************************/
/* Confirm the OBJECT member is present                              */
/*********************************************************************/
     objmem = "'"strip(sysdsname,'B',"'")'('object')'"'"
     if sysdsn(objmem) <> 'OK' then
        call rcexit 12 'Required OBJECT member' objmem 'is missing'
/*********************************************************************/
/* Allocate the object member                                        */
/*********************************************************************/
     call tsotrap "ALLOC F(OBJECT) DA("objmem") SHR REUSE"
    end
/*********************************************************************/
/* Load the OBJECT member into a stem                                */
/*********************************************************************/
 call tsotrap "EXECIO * DISKR" objdd "(STEM OBJLINE. FINIS"
/*********************************************************************/
/* Check to see if FILTER CRITERIA was provided                      */
/*********************************************************************/
 filter = ''
 if listdsi("FILTER" "FILE") = Ø then
    do
     call tsotrap "EXECIO * DISKR FILTER (STEM FILTLINE. FINIS"
     do f=1 to filtline.Ø
        interpret filtline.f
     end
     filter_dsn = sysdsname
    end
/*********************************************************************/
/* Call CPSMINIT to setup a valid CPSM environment                   */
/*********************************************************************/
 cpsm_thread = cpsminit(cmas)
/*********************************************************************/
/* Call CPSMOLEN to get the length of the object                     */
/*********************************************************************/
 object_len = cpsmolen(cpsm_thread object)
/*********************************************************************/
/* Call CPSMGET to get the result set, address and count             */
```

```
/*********************************************************************/
 obj_result = cpsmget(cpsm_thread context scope object filter)
 parse var obj_result obj_result obj_count
 call msg obj_count object 'objects found' filter
 if tsoenv = 'BACK' then say
/*********************************************************************/
/* Check to see if GROUP BY CRITERIA was provided                    */
/*********************************************************************/
 group = ''
 sumopt = ''
 if listdsi("GROUP" "FILE") = Ø then
    do
     call tsotrap "EXECIO * DISKR GROUP (STEM GRPLINE. FINIS"
     do g=1 to grpline.Ø
        interpret grpline.g
     end
     group_dsn = sysdsname
/*********************************************************************/
/* Call CPSMGRP to group the result set, address and count           */
/*********************************************************************/
     obj_result = cpsmgrp(cpsm_thread group obj_result sumopt)
     parse var obj_result obj_result obj_count
     call msg obj_count object 'objects grouped by' group sumopt
     if tsoenv = 'BACK' then say
    end
/*********************************************************************/
/* Check to see if a HEADING was provided                            */
/*********************************************************************/
 heading_dsn = ''
 if listdsi("HEADING" "FILE") = Ø then
    do
     heading_dsn = sysdsname
     call tsotrap "EXECIO * DISKR HEADING (STEM HEADLINE. FINIS"
     do h=1 to headline.Ø
        interpret headline.h
     end
     say
    end
/*********************************************************************/
/* Check to see if SORT CRITERIA was provided                        */
/*********************************************************************/
 sort = ''
 if listdsi("ORDER" "FILE") = Ø then
    do
     call tsotrap "EXECIO * DISKR ORDER (STEM ORDLINE. FINIS"
     do o=1 to ordline.Ø
        interpret ordline.o
     end
     order_dsn = sysdsname
/*********************************************************************/
```

```
/* Suffix the SORT CRITERIA with a period '.'                        */
/*********************************************************************/
     sort = sort'.'
     sortlen = length(sort)
/*********************************************************************/
/* Sort the results                                                  */
/*********************************************************************/
     SRC = eyuapi("ORDER BY(SORT)",
                  "LENGTH("sortlen")",
                  "RESULT(OBJ_RESULT)",
                  "THREAD(CPSM_THREAD)",
                  "RESPONSE(RESPONSE)",
                  "REASON(REASON)")
/*********************************************************************/
/* Error and message processing                                      */
/*********************************************************************/
     call rcexit SRC 'ORDER failed for' object
     call cpsmerr 2Ø 'MAINLINE ORDER' reason object response
     call saydd msgdd Ø 'ORDER BY' sort 'completed'
    end
/*********************************************************************/
/* Loop through the results table                                    */
/*********************************************************************/
 do i=1 to obj_count
/*********************************************************************/
/* Check for a page break and increment page_num                     */
/*********************************************************************/
    if heading_dsn <> '' & (i // page_size) = Ø then
       do
        page_num = page_num + 1
        say
        do h=1 to headline.Ø
           interpret headline.h
        end
        say
       end
/*********************************************************************/
/* Fetch the results                                                 */
/*********************************************************************/
    if group = '' then
       do
        FRC = eyuapi("FETCH INTO(ROW)",
                     "LENGTH(OBJECT_LEN)",
                     "RESULT(OBJ_RESULT)",
                     "THREAD(CPSM_THREAD)",
                     "RESPONSE(RESPONSE)",
                     "REASON(REASON)")
       end
    else
       do
```

```
        FRC = eyuapi("FETCH INTO(ROW)",
                     "LENGTH(OBJECT_LEN)",
                     "RESULT(OBJ_RESULT)",
                     "THREAD(CPSM_THREAD)",
                     "RESPONSE(RESPONSE)",
                     "REASON(REASON)")
/*                   "COUNT(SUM_COUNT) BOTH DETAIL", */
       end
/*********************************************************************/
/* Error processing                                                  */
/*********************************************************************/
    call rcexit FRC 'FETCH failed for' object
    if response <> 1Ø27 then
       call cpsmerr 21 'MAINLINE FETCH' reason object response
/*********************************************************************/
/* TPARSE the results table                                          */
/*********************************************************************/
    TRC = eyuapi("TPARSE",
                 "OBJECT("object")",
                 "PREFIX(OBJ)",
                 "THREAD(CPSM_THREAD)",
                 "STATUS(RESPONSE)",
                 "VAR(ROW.1)")
/*********************************************************************/
/* Error processing                                                  */
/*********************************************************************/
    call rcexit TRC 'TPARSE failed for' object response
/*********************************************************************/
/* Format the 'print' line                                           */
/*********************************************************************/
    do j=1 to objline.Ø
       interpret objline.j
    end
/*********************************************************************/
/* If the 'print' line sets FRC = 9999 then leave the loop           */
/*********************************************************************/
    if FRC = 9999 then
       do
        call saydd msgdd Ø 'FETCH, TPARSE and print processing bypassed'
        leave
       end
 end
/*********************************************************************/
/* Terminate the connection                                          */
/*********************************************************************/
 EXITRC = cpsmterm(cmas)
/*********************************************************************/
/* Print line details                                                */
/*********************************************************************/
 call saydd msgdd 1 'OBJECT print line used from' objdd 'DD' sysdsname
 call tsotrap "EXECIO * DISKW" msgdd "(STEM OBJLINE."
/*********************************************************************/
/* Filter details                                                    */
/*********************************************************************/
 if filter <> '' then
    do
     call saydd msgdd 1 'Filter criteria used from FILTER DD' filter_dsn
     call tsotrap "EXECIO * DISKW" msgdd "(STEM FILTLINE."
    end
/*********************************************************************/
/* Group details                                                     */
/*********************************************************************/
 if group <> '' then
    do
     call saydd msgdd 1 'Group criteria used from GROUP DD' group_dsn
     call tsotrap "EXECIO * DISKW" msgdd "(STEM GRPLINE."
    end
/*********************************************************************/
/* Heading Details                                                   */
/*********************************************************************/
 if heading_dsn <> '' then
    do
     call saydd msgdd 1 'Heading used from HEADING DD' heading_dsn
     call tsotrap "EXECIO * DISKW" msgdd "(STEM HEADLINE."
    end
/*********************************************************************/
/* Sort details                                                      */
/*********************************************************************/
 if sort <> '' then
    do
     call saydd msgdd 1 'Sort criteria used from ORDER DD' order_dsn
     call tsotrap "EXECIO * DISKW" msgdd "(STEM ORDLINE."
    end
/*********************************************************************/
/* Shutdown                                                          */
/*********************************************************************/
/* End of unique program code                                        */
/*********************************************************************/
 shutdown: nop
           if cpsm_thread <> 0 & TRC = 9999 then
              EXITRC = cpsmterm(cmas)
/*********** @REFRESH BEGIN STOP     2002/08/03 08:42:33 *************/
/* Shutdown message and terminate                                    */
/*********************************************************************/
           call stdexit time('e')
/*********** @REFRESH END   STOP     2002/08/03 08:42:33 *************/
/*********** @REFRESH BEGIN SUBBOX   2002/08/15 12:46:24 *************/
/* Internal Subroutines provided in CPSMRW                           */
/*                                                                   */
/* RCEXIT   - Exit on non-zero return codes                          */
/* TRAP     - Issue a common trap error message using rcexit         */
/* ERRMSG   - Build common error message with failing line number    */
/* STDENTRY - Standard Entry logic                                    */
/* STDEXIT  - Standard Exit logic                                     */
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* DDCHECK  - Determine if a required DD is allocated                 */
/* DDLIST   - Returns number of DD's and populates DDLIST variable   */
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/* SAYDD    - Print messages to the requested DD                      */
/* JOBINFO  - Get job related data from control blocks               */
/* PTR      - Pointer to a storage location                           */
/* STG      - Return the data from a storage location                 */
/* MODTRACE - Module Trace                                            */
/* CPSMCMAS - Get CMAS name                                           */
/* CPSMERR  - Format a CPSM error message for RCEXIT                  */
/* CPSMINIT - Initialize a CPSM session                               */
/* CPSMOLEN - Get a CPSM Objects Length                               */
/* CPSMGET  - Get a CPSM Result Set                                   */
/* CPSMGRP  - Group a CPSM Results Set                                */
/* CPSMTERM - Terminate a CPSM session                                */
/*********** @REFRESH END   SUBBOX   2ØØ2/Ø8/15 12:46:24 *************/
/*********** @REFRESH BEGIN RCEXIT   2ØØ2/Ø8/15 15:28:39 *************/
/* RCEXIT   - Exit on non-zero return codes                           */
/* EXITRC   - Return code to exit with (if non zero)                  */
/* ZEDLMSG  - Message text for it with for non zero EXITRC's         */
/**********************************************************************/
 rcexit: parse arg EXITRC zedlmsg
         if EXITRC <> Ø then
            do
             trace 'o'
/**********************************************************************/
/* If execution environment is ISPF then VPUT ZISPFRC                 */
/**********************************************************************/
             if execenv = 'TSO' | execenv = 'ISPF' then
                do
                 if ispfenv = 'YES' then
                    do
                     zispfrc = EXITRC
/**********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/**********************************************************************/
                     address ISPEXEC "VPUT (ZISPFRC)"
                    end
                end
/**********************************************************************/
/* If a message is provided, wrap it in date, time and EXITRC        */
/**********************************************************************/
             if zedlmsg <> '' then
                do
```

```
                 zedlmsg = time('L') execname zedlmsg 'RC='EXITRC
                 call msg zedlmsg
                end
/**********************************************************************/
/* Write the contents of the Parentage Stack                          */
/**********************************************************************/
             stacktitle = 'Parentage Stack Trace ('queued()' entries):'
/**********************************************************************/
/* Write to MSGDD if background                                       */
/**********************************************************************/
             if tsoenv = 'BACK' then
                do
                 call saydd msgdd 1 zedlmsg
                 call saydd msgdd 1 stacktitle
                end
             else
/**********************************************************************/
/* Write to the ISPF Log if foreground                                */
/**********************************************************************/
                do
                 zerrlm = zedlmsg
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/**********************************************************************/
/* Unload the Parentage Stack                                         */
/**********************************************************************/
             do queued()
                pull stackinfo
                if tsoenv = 'BACK' then
                   call saydd msgdd Ø stackinfo
                else
                   do
                    zerrlm = stackinfo
                    address ISPEXEC "LOG MSG(ISRZØØ3)"
                   end
             end
/**********************************************************************/
/* Put a terminator in the ISPF Log for the Parentage Stack           */
/**********************************************************************/
             if tsoenv = 'FORE' then
                do
                 zerrlm = center(' 'stacktitle' ',78,'-')
                 address ISPEXEC "LOG MSG(ISRZØØ3)"
                end
/**********************************************************************/
/* Signal SHUTDOWN.  SHUTDOWN label MUST exist in the program        */
/**********************************************************************/
             signal shutdown
```

```
            end
         else
            return
/*********** @REFRESH END   RCEXIT   2002/08/15 15:28:39 *************/
/*********** @REFRESH BEGIN TRAP     2002/08/07 11:48:14 *************/
/* TRAP     - Issue a common trap error message using rcexit         */
/* PARM     - N/A                                                    */
/*********************************************************************/
 trap: traptype = condition('C')
       if traptype = 'SYNTAX' then
          msg = errortext(RC)
       else
          msg = condition('D')
       trapline = strip(sourceline(sigl))
       msg = traptype 'TRAP:' msg', Line:' sigl '"'trapline'"'
       call rcexit 666 msg
/*********** @REFRESH END   TRAP     2002/08/07 11:48:14 *************/
/*********** @REFRESH BEGIN ERRMSG   2002/08/10 16:53:04 *************/
/* ERRMSG   - Build common error message with failing line number    */
/* ERRLINE  - The failing line number passed by caller from SIGL     */
/* TEXT     - Error message text passed by caller                    */
/*********************************************************************/
 errmsg: nop
         parse arg errline text
         return 'Error on statement' errline',' text
/*********** @REFRESH END   ERRMSG   2002/08/10 16:53:04 *************/
/*********** @REFRESH BEGIN STDENTRY 2002/09/11 01:48:55 *************/
/* STDENTRY - Standard Entry logic                                   */
/* MSGDD    - Optional MSGDD used only in background                 */
/*********************************************************************/
 stdentry: module = 'STDENTRY'
           if wordpos(module,probe) <> 0 then trace 'r'; else trace 'n'
           parse arg sparms
           push trace() time('L') module 'From:' sigl 'Parms:' sparms
           arg msgdd
           parse upper source . . execname . execdsn . . execenv .
/*********************************************************************/
/* Startup values                                                    */
/*********************************************************************/
           EXITRC = 0
           MAXRC = 0
           ispfenv = 'NO'
           popup = 'NO'
           lockpop = 'NO'
           keepstack = 'NO'
/*********************************************************************/
/* Determine environment                                             */
/*********************************************************************/
           if substr(execenv,1,3) <> 'TSO' & execenv <> 'ISPF' then
              tsoenv = 'NONE'
           else
              do
               tsoenv = sysvar('SYSENV')
              "ISPQRY"
               if RC = Ø then ispfenv = 'YES'
              end
/*********************************************************************/
/* MODTRACE must occur after the setting of ISPFENV                */
/*********************************************************************/
           call modtrace 'START' sigl
/*********************************************************************/
/* Startup message                                                 */
/*********************************************************************/
           lpar = mvsvar('SYSNAME')
           startmsg = execname 'started' date() time() 'on' lpar
           if tsoenv = 'BACK' then
              do
               jobname = mvsvar('SYMDEF','JOBNAME')
               jobinfo = jobinfo()
               parse var jobinfo jobtype jobnum .
               say jobname center(' 'startmsg' ',61,'-') jobtype jobnum
               say
/*********************************************************************/
/* If MSGDD is provided, write the STARTMSG and SYSEXEC DSN to MSGDD */
/*********************************************************************/
               if msgdd <> '' then
                  do
                   call saydd msgdd 1 startmsg
                   x = listdsi('SYSEXEC' 'FILE')
                   call saydd msgdd Ø execname 'loaded from' sysdsname
/*********************************************************************/
/* If there are PARMS, write them to the MSGDD                     */
/*********************************************************************/
                   if parms <> '' then
                      call saydd msgdd Ø 'Parms:' parms
/*********************************************************************/
/* If there is a STEPLIB, write the STEPLIB DSN MSGDD              */
/*********************************************************************/
                   if listdsi('STEPLIB' 'FILE') = Ø then
                      do
                       steplibs = dddsns('STEPLIB')
                       call saydd msgdd Ø 'STEPLIB executables loaded',
                           'from' word(dddsns,1)
                       if dddsns('STEPLIB') > 1 then
                          do
                           do stl=2 to steplibs
                              call saydd msgdd Ø copies(' ',31),
                                   word(dddsns,stl)
                           end
                          end
                      end
                  end
              end
           pull tracelvl . module . sigl . sparms
           call modtrace 'STOP' sigl
           interpret 'trace' tracelvl
           return
/*********** @REFRESH END   STDENTRY 2ØØ2/Ø9/11 Ø1:48:55 *************/
/*********** @REFRESH BEGIN STDEXIT  2ØØ2/Ø9/11 Ø1:ØØ:51 *************/
/* STDEXIT  - Standard Exit logic                                    */
/* ENDTIME  - Elapsed time                                           */
/* Note: Caller must set KEEPSTACK if the stack is valid             */
/*********************************************************************/
 stdexit: module = 'STDEXIT'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg endtime
          endmsg = execname 'ended' date() time() format(endtime,,1)
/*********************************************************************/
/* if MAXRC is greater then EXITRC then set EXITRC to MAXRC          */
/*********************************************************************/
          if MAXRC > EXITRC then EXITRC = MAXRC
          endmsg = endmsg 'on' lpar 'RC='EXITRC
          if tsoenv = 'BACK' then
             do
              say
              say jobname center(' 'endmsg' ',61,'-') jobtype jobnum
              if msgdd <> '' then
                 do
                  call saydd msgdd 1 execname 'ran in' endtime 'seconds'
                  call saydd msgdd Ø endmsg
                 end
             end
/*********************************************************************/
/* Remove STDEXIT and MAINLINE Parentage Stack entries, if there     */
/*********************************************************************/
          if queued() > Ø then pull . . module . sigl . sparms
          if queued() > Ø then pull . . module . sigl . sparms
          call modtrace 'STOP' sigl
/*********************************************************************/
/* if the Parentage Stack is not empty, display its contents         */
/*********************************************************************/
          if queued() > Ø & keepstack = 'NO' then
             do
              say 'Leftover Parentage Stack Entries:'
              say
              do queued()
                 pull stackundo
                 say stackundo
              end
              EXITRC = 1
             end
/*********************************************************************/
/* Exit                                                              */
/*********************************************************************/
         exit(EXITRC)
/*********** @REFRESH END   STDEXIT  2ØØ2/Ø9/11 Ø1:ØØ:51 *************/
/*********** @REFRESH BEGIN MSG      2ØØ2/Ø9/11 Ø1:35:53 *************/
/* MSG      - Determine whether to SAY or ISPEXEC SETMSG the message */
/* ZEDLMSG  - The long message variable                              */
/*********************************************************************/
 msg: module = 'MSG'
      parse arg zedlmsg
      if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
      parse arg sparms
      push trace() time('L') module 'From:' sigl 'Parms:' sparms
      call modtrace 'START' sigl
/*********************************************************************/
/* If this is background or OMVS use SAY                             */
/*********************************************************************/
      if tsoenv = 'BACK' | execenv = 'OMVS' then
         say zedlmsg
      else
/*********************************************************************/
/* If this is foreground and ISPF is available, use SETMSG           */
/*********************************************************************/
         do
          if ispfenv = 'YES' then
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
             address ISPEXEC "SETMSG MSG(ISRZØØØ)"
          else
             say zedlmsg
         end
      pull tracelvl . module . sigl . sparms
      call modtrace 'STOP' sigl
      interpret 'trace' tracelvl
      return
/*********** @REFRESH END   MSG      2ØØ2/Ø9/11 Ø1:35:53 *************/
/*********** @REFRESH BEGIN DDCHECK  2ØØ2/Ø9/11 Ø1:Ø8:3Ø *************/
/* DDCHECK  - Determine if a required DD is allocated                */
/* DD       - DDNAME to confirm                                      */
/*********************************************************************/
 ddcheck: module = 'DDCHECK'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          arg dd
          dderrmsg = 'OK'
          LRC = listdsi(dd "FILE")
/*********************************************************************/
/* Allow sysreason=3 to verify SYSOUT DD statements                  */
/*********************************************************************/
          if LRC <> Ø & strip(sysreason,'L',Ø) <> 3 then
             do
              dderrmsg = errmsg(sigl 'Required DD' dd 'is missing')
              call rcexit LRC dderrmsg sysmsglvl2
             end
          pull tracelvl . module . sigl . sparms
          call modtrace 'STOP' sigl
          interpret 'trace' tracelvl
          return
/*********** @REFRESH END   DDCHECK  2ØØ2/Ø9/11 Ø1:Ø8:3Ø *************/
/*********** @REFRESH BEGIN DDLIST   2ØØ2/12/15 Ø4:54:32 *************/
/* DDLIST   - Returns number of DD's and populates DDLIST variable   */
/* N/A      - None                                                   */
/*********************************************************************/
 ddlist: module = 'DDLIST'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
/*********************************************************************/
/* Trap the output from the LISTA STATUS command                     */
/*********************************************************************/
         call outtrap 'lines.'
         address TSO "LISTALC STATUS"
         call outtrap 'off'
         ddnum = Ø
/*********************************************************************/
/* Parse out the DDNAMEs and concatenate into a list                 */
/*********************************************************************/
         ddlist = ''
         do ddl=1 to lines.Ø
            if words(lines.ddl) = 2 then
               do
                parse upper var lines.ddl ddname .
                ddlist = ddlist ddname
                ddnum = ddnum + 1
               end
            else
               do
                iterate
               end
         end
/*********************************************************************/
```

```
/* Return the number of DD's                                        */
/********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return ddnum
/*********** @REFRESH END   DDLIST   2ØØ2/12/15 Ø4:54:32 *************/
/*********** @REFRESH BEGIN DDDSNS   2ØØ2/Ø9/11 ØØ:37:36 *************/
/* DDDSNS   - Returns number of DSNs in a DD and populates DDDSNS    */
/*------------------------------------------------------------------*/
/* TARGDD   - DD to return DSNs for                                 */
/********************************************************************/
 dddsns: module = 'DDDSNS'
         if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
         parse arg sparms
         push trace() time('L') module 'From:' sigl 'Parms:' sparms
         call modtrace 'START' sigl
         arg targdd
         if targdd = '' then call rcexit 77 'DD missing for DDDSNS'
/********************************************************************/
/* Trap the output from the LISTA STATUS command                    */
/********************************************************************/
         x = outtrap('lines.')
         address TSO "LISTALC STATUS"
         dsnnum = Ø
         ddname = '$DDNAME$'
/********************************************************************/
/* Parse out the DDNAMEs, locate the target DD and concatentate DSNs */
/********************************************************************/
         do ddd=1 to lines.Ø
            select
               when words(lines.ddd) = 1 & targdd = ddname &,
                    lines.ddd <> 'KEEP' then
                    dddsns = dddsns strip(lines.ddd)
               when words(lines.ddd) = 1 & strip(lines.ddd),
                    <> 'KEEP' then
                    dddsn.ddd = strip(lines.ddd)
               when words(lines.ddd) = 2 then
                    do
                     parse upper var lines.ddd ddname .
                     if targdd = ddname then
                        do
                         fdsn = ddd - 1
                         dddsns = lines.fdsn
                        end
                    end
               otherwise iterate
            end
         end
/********************************************************************/
```

```
/* Get the last DD                                                   */
/*********************************************************************/
         ddnum = ddlist()
         lastdd = word(ddlist,ddnum)
/*********************************************************************/
/* Remove the last DSN from the list if not the last DD or SYSEXEC   */
/*********************************************************************/
         if targdd <> 'SYSEXEC' & targdd <> lastdd then
            do
             dsnnum = words(dddsns) - 1
             dddsns = subword(dddsns,1,dsnnum)
            end
/*********************************************************************/
/* Return the number of DSNs in the DD                               */
/*********************************************************************/
         pull tracelvl . module . sigl . sparms
         call modtrace 'STOP' sigl
         interpret 'trace' tracelvl
         return dsnnum
/*********** @REFRESH END   DDDSNS   2ØØ2/Ø9/11 ØØ:37:36 *************/
/*********** @REFRESH BEGIN TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
/* TSOTRAP  - Capture the output from a TSO command in a stem        */
/*-------------------------------------------------------------------*/
/* VALIDRC  - Optional valid RC, defaults to zero                    */
/* TSOPARM  - Valid TSO command                                      */
/*********************************************************************/
 tsotrap: module = 'TSOTRAP'
          if wordpos(module,probe) <> Ø then trace 'r'; else trace 'n'
          parse arg sparms
          push trace() time('L') module 'From:' sigl 'Parms:' sparms
          call modtrace 'START' sigl
          parse arg tsoparm
/*********************************************************************/
/* If the optional valid_rc parm is present use it, if not assume Ø  */
/*********************************************************************/
          parse var tsoparm valid_rc tso_cmd
          if datatype(valid_rc,'W') = Ø then
             do
              valid_rc = Ø
              tso_cmd = tsoparm
             end
          call outtrap 'tsoout.'
          tsoline = sigl
          address TSO tso_cmd
          CRC = RC
          call outtrap 'off'
/*********************************************************************/
/* If RC = Ø then return                                             */
/*********************************************************************/
          if CRC <= valid_rc then
```

```
             do
              pull tracelvl . module . sigl . sparms
              call modtrace 'STOP' sigl
              interpret 'trace' tracelvl
              return CRC
             end
          else
             do
              trapmsg = center(' TSO Command Error Trap ',78,'-')
              terrmsg = errmsg(sigl 'TSO Command:')
/*********************************************************************/
/* If RC <> Ø then format output depending on environment           */
/*********************************************************************/
              if tsoenv = 'BACK' | execenv = 'OMVS' then
                 do
                  say trapmsg
                  do c=1 to tsoout.Ø
                     say tsoout.c
                  end
                  say trapmsg
                  call rcexit CRC terrmsg tso_cmd
                 end
              else
/*********************************************************************/
/* If this is foreground and ISPF is available, use the ISPF LOG     */
/*********************************************************************/
                 do
                  if ispfenv = 'YES' then
                     do
                      zedlmsg = trapmsg
/*********************************************************************/
/* Does not call ISPWRAP to avoid obscuring error message modules    */
/*********************************************************************/
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
                      do c=1 to tsoout.Ø
                         zedlmsg = tsoout.c
                         address ISPEXEC "LOG MSG(ISRZØØØ)"
                      end
                      zedlmsg = trapmsg
                      address ISPEXEC "LOG MSG(ISRZØØØ)"
                      call rcexit CRC terrmsg tso_cmd,
                          ' see the ISPF Log (Option 7.5) for details'
                     end
                  else
                     do
                      say trapmsg
                      do c=1 to tsoout.Ø
                         say tsoout.c
                      end
                      say trapmsg
                      call rcexit CRC terrmsg tso_cmd
                     end
                 end
             end
/*********** @REFRESH END   TSOTRAP  2ØØ2/12/15 Ø5:18:45 *************/
```

Editor's note: this article will conclude in next month's issue.

Robert Zenuk
Systems Programmer (USA)

CICS questions and answers

Q We are somewhat confused! We are designing a system that would make use of daisy-chained DPL. Some here believe that we need to make the CICS connections APPC if we want to do this. Is this true? We need to understand our options before we commit to our design.

A It all depends on whether you want to dynamically route the DPL or not. To help explain let's use three CICS regions – CICSA, CICSB, and CICSC. If you want to use dynamic routing then the link between the prior CICS regions needs to be APPC. So the link between CICSA and CICSB needs to be APPC, the link between CICSB and CICSC could be XM. The routing exit will not be driven if the DPL request arrived over a non-APPC connection. Static links are fine and can be routed no matter what the connection types.

If you have any CICS-related questions, please send them in and we will do our best to find answers. Alternatively, e-mail them directly to cicsq@xephon.net.

Phoenix Software International has announced CONDOR, it's online library management and program development system.

The product functions as a stand-alone VTAM application and/or under CICS and other TP monitors.

Its system maintenance utilities allow users to access all areas of their system without disrupting production.

For further information contact:
Phoenix Software International, 5200 West Century Blvd, Suite 800, Los Angeles, CA 90045, USA.
Tel: (310) 338 0400.
URL: http://www.phoenixsoftware.com/Condor/condor.htm.

* * *

BMC has announced Mainview AutoOPERATOR for OS/390, which helps increase availability through automation and rules-based operations, and support IMS, CICS, WebSphere MQ, and TapeSHARE. Included with AutoOPERATOR for OS/390 Version 6.3.01 is new Total Object Manager (TOM), enabling IT administrators to use object management as an umbrella for the management and automation of IT resources that support certain business functions.

It lets users control IT assets across the sysplex better and creates the foundation for managing divergent objects with complex inter-dependencies, such as Unix System Services (USS) processes, MQ queues, and SAP applications.

For further information contact:
BMC Software, 2101 City West Blvd, Houston, TX 77042, USA.
Tel: (713) 918 8800.
URL: http://www.bmc.com/products/proddocview/0,2832,19052_19429_28229_8571,00.html.

* * *

IBM has announced DB2 Query Management Facility Version 8, which exploits new capabilities of DB2 V8 and has new data visualization, solution building, Web-enablement, and solution sharing capabilities.

DB2 QMF Enterprise Edition includes DB2 QMF for TSO/CICS, DB2 QMF High Performance Option (HPO), DB2 QMF for Windows, and DB2 QMF for WebSphere.

DB2 QMF Classic Edition supports end users functioning entirely from traditional mainframe terminals and emulators (including IBM Host On Demand) to access DB2 UDB databases. This edition consists of DB2 QMF for TSO/CICS.

New in DB2 QMF for TSO/CICS is support for names up to 128 characters in length for Auth ID, Current SQLID, and table names. Table column names can be up to 30 characters long. Support is based on whatever length the database allows. Support includes larger data entry fields and the display of names in QMF dialog screens.

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