# 194 CICS update

*January 2002*

## 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 1998 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.



*Printed in England.*

# Recognizing optimized temporary storage usage

## BACKGROUND

Analysis of the statistics from temporary storage usage within CICS Transaction Server can reveal useful information about how well the system is handling resources within the temporary storage domain. This article highlights several areas that are worth analysing when reviewing temporary storage use and performance in a CICS system.

## TEMPORARY STORAGE STATISTICS

CICS records statistics information at various points during its execution. Statistics can be generated at periodic intervals, at end-of-day processing, and during system shutdown, and can also be recorded by means of the supplied sample transaction STAT, which executes sample program DFH0STAT. Utility DFHSTUP is provided to format and report CICS statistics information offline; using DFH0STAT, a statistics report can be generated and spooled to a specific destination for analysis.

Temporary storage statistics are also provided for interpretation when running the CICS-supplied system dump formatting option for the temporary storage domain. If the CICS TS 1.3 system dump formatting option ‘VERBX DFHPD530 'TS'’ is specified within IPCS, the formatted output will contain statistics information at the start of the TS dump summary section.

Details of the various pieces of data recorded by temporary storage statistics processing, and the lifetime of the data (ie when/if the values are reset) can be found in the *CICS Performance Guide*.

## INTERPRETING AN EXAMPLE SET OF STATISTICS

The following example data (from some of the temporary storage statistics) will be used to explain a number of different aspects of temporary storage processing, and to highlight why this example

 

CICS system is making efficient use of temporary storage resources:

```
Put/Putq auxiliary storage requests : 165131
Get/Getq auxiliary storage requests : 191472
Peak temporary storage names in use : 24
Number of entries in longest queue : 1
Times queues created : 7491
Control interval size : 8192
Available bytes per control interval : 8128
Longest auxiliary temp storage record : 7950
Number of control intervals available : 1019
Peak control intervals in use : 9
Number of temp storage compressions : 82457
Temporary storage buffers : 24
Buffer waits : 0
Peak users waiting on buffer : 0
Buffer writes : 0
Forced writes for recovery : 0
Buffer reads : 0
Format writes : 0
```

OBSERVATIONS

From this particular statistical example, it can be seen that of the order of several hundred thousand read and write requests were made against auxiliary temporary storage queues during the statistics interval. Note that the formatter (and documentation in the *Performance Guide*) refers to Put/Putq and Get/Getq requests. These are the old-style macro-level interfaces for application programming calls to temporary storage services. In fact, these statistics also relate to the command-level Writeq and Readq API calls. A Writeq command relates to a Putq macro call; similarly a Readq command relates to a Getq macro call. There is no command-level equivalent to the Put and Get macro calls. Although macro-level programming is no longer supported for CICS applications, it is still used internally within the CICS product, where appropriate.

The example figures show that the peak number of queues present on the system in this statistics interval was 24. However, 7491 queues have been created. This demonstrates that a good queue management policy is being adhered to, since nearly all the queues have been deleted after being used. The 'Number of entries on longest queue' statistic also demonstrates that the queues are being used as single objects – they contain no more than one record (item).

The Control Interval (CI) size is 8192 bytes, and the longest record written to a queue is shown to have been 7950 bytes. This means that no data has spanned several CIs. Such spanning of CIs is handled automatically by CICS if required. Note that CICS reserves a small part of each CI for its own use – this means that, at best, a CI can never have all its space available for temporary storage data. This is why the 'Available bytes per control interval' value is less than the 'Control interval size'.

Out of 1091 CIs on the DFHTEMP dataset, the peak number that was in use was 9. There were 24 temporary storage buffers available in-core, however. Since a buffer can hold one CI, this means that the number of buffers was in fact over-generous. However, there are advantages to this. Although it results in additional virtual storage requirements for temporary storage management, the fact that there are sufficient buffers for all CIs that were required at any point in time means that I/O was never required by temporary storage to flush one CI out to disk in order to free up buffer space for a required CI to be read in to. This means that temporary storage processing was very efficient – I/Os were avoided for such events (the 'Buffer writes' and 'Buffer reads' statistics both show 0). This shows no write I/O events were required (either for recovery reasons or forced by the need to place a different CI into a buffer); it also shows that no CIs had to be read in from disk.

Since only 9 of the 1000-odd CIs were actually used, this means that the creation and deletion of queues by the application workload was taking place at a fairly constant pace. In other words, queues were being deleted (and their space in buffers being made available for reuse by other requests) at approximately the same rate as new queues were being created. If this were not the case, and queues were being created faster than being deleted, the number of CIs in use would have increased during the statistics interval, until eventually more than 24 CIs would have been utilized. After this point, the temporary storage buffers would not have been able to accommodate all the temporary storage data in-core any more, and subsequent Writeqs would have necessitated I/O events to flush old CIs from buffers to disk, to allow new CIs to be selected and placed in the buffers.

The statistics demonstrate that efficient application design has therefore improved system performance, by alleviating the need for I/O operations. In fact, since I/O has been avoided for this temporary storage environment, the use of auxiliary temporary storage support may well be reviewed for this system. Main temporary storage usage could be considered instead (note that the default destination is auxiliary). However, by providing a number of pre-GETMAINed auxiliary temporary storage buffers, the throughput of an auxiliary temporary storage request can be better than that of an equivalent Writeq to a main storage destination (assuming I/O operations can be avoided, as in this example). This is because a record written to a main temporary storage queue requires a GETMAIN operation to accommodate it, and a FREEMAIN operation when its queue is deleted, unlike for an auxiliary destination.

Since, in this example, CICS avoided I/O operations to the DFHTEMP VSAM dataset, there were no task waits on VSAM strings. Therefore, the number of allocated strings could be reduced to a low value for such a system. Conversely, the number of buffers can be set to a large value (at least to one that exceeds the peak number of CIs in use, as in this case). The system initialization parameter ‘TS=’ controls the number of buffers and strings, with default values of 3 and 3. The theoretical maximum number of auxiliary storage buffers is 32,767.

There have been no requests forced to wait for buffer availability, and no formatting writes required. Formatting writes occur when all CIs on DFHTEMP have data in them, and no space is available for a record to be placed into. CICS then tries to extend the dataset by formatting further CIs from secondary storage allocation, subject to the dataset definition. Similarly, there have been no forced writes for recovery purposes, as would be the case when CICS commits updates to recoverable temporary storage queues.

## COMPRESSIONS AND THE 75% RULE

Compressions of temporary storage CIs occur when a CI is selected to hold a record, and that CI has sufficient space for the new record, but the space is not available in one contiguous piece of storage within the CI. In such a case, CICS ‘squeezes out the gas’ by moving all the (still required) records to the start of the CI, thus leaving a contiguous section of reusable storage at the end of the CI. This is then used to hold the new record. Each such operation is a compression, and results in an increment of the count shown in the field ‘Number of temp storage compressions’. In the example statistics given above, this is 82,547. Since we have demonstrated than CI space is being freed up (by queue deletions) at a similar rate to new Writeq requests being processed, it is to be expected that compressions of those CIs in the buffers would be fairly high. This is because we are reusing existing CIs within the buffers, instead of selecting other CIs from the auxiliary storage dataset. CICS does this to optimize the use of CIs by choosing those already in-core in a buffer rather than ones held on the DFHTEMP dataset, thus avoiding I/O overheads.

The frequency of CI compression is not therefore directly related to the number of CIs in use by the system. It is directly related to a given CI’s ability to hold a record being written to temporary storage. However, there is a relationship between the number of CIs and the number of buffer compressions. Clearly, if you have a system with either ‘many’ or ‘few’ CIs in its DFHTEMP dataset, the likelihood that a CI will have already been used and have fragmented space is ‘low’ or ‘high’ respectively. On a system with many CIs, CICS will be able to store records within empty or almost empty CIs more often than on a system with few CIs. As such, the need to compress CIs is reduced in such a system with a large number of CIs. As in the example given above, however, when the rate of Deleteq requests matches or exceeds that of Writeqs, many of the CIs will not need to be selected for use by CICS; the result of this optimization is a high compression rate on those CIs within the buffers.

There is another related factor, however – the 75% rule. In CICS/ESA 4.1.0, an empty (new) CI was selected for a Writeq request (if space was not available within a CI currently buffered in-core) up until 75% of the primary allocation of CIs in DFHTEMP was reached. At this point, CICS switched to using a first-fit algorithm and went back to the start of DFHTEMP, reusing old CIs. These would either be logically empty by now if their queue data had been deleted by

Deleteq commands) or else have free space within them that may (or may not) require a compression to make the free space contiguous to hold a new record.

In CICS TS, temporary storage processing now continues allocating empty CIs up until the end of primary storage allocation in DFHTEMP. Once that is reached, CICS then switches back to the first-fit algorithm. This change was made because the ability to retain a 25% pool of empty CIs was no longer required by temporary storage processing in CICS TS. The CICS/ESA 4.1.0 mechanism of long record support for Writeqs that exceeded the length of an entire CI had required the ability to write special header records the size of an entire CI, and hence for CICS to maintain a free pool of CIs for such records. This is no longer the case when supporting such long records within the restructured temporary storage domain in CICS TS; the 25% pool no longer exists.

By selecting empty CIs rather than reusing existing ones, the design reduced the likelihood of an I/O event being required to read in a CI to be used to satisfy a Writeq request. This means that more CIs are selected from the range of primary allocation CIs in DFHTEMP than in CICS/ESA 4.1.0. As such, the likelihood of a compression is reduced. By utilizing the last 25% of CIs within the dataset, instead of reverting to the start once 75% had been used, the likelihood is increased that old records within the system will have been deleted by the time CICS has to revert to the start of the dataset and look for free space within old CIs. The expectation is that, by the time this point is reached, applications will have freed up queues and led to empty CIs once more. Selecting an empty CI avoids the need for I/O to read it into a buffer; it also avoids the possible need to compress a CI to move any remaining records to its start before reusing it.

A comparison of temporary storage statistics between a CICS/ESA 4.1.0 and a CICS TS system may well therefore show an increase in the peak number of CIs in use, and a decrease in the number of buffer compressions, when comparing the latter with the former. This assumes a comparable workload and similar temporary storage usage and access patterns between the two versions of CICS.

CONTACT INFORMATION

I hope that this article has helped explain the background to interpretation of temporary storage statistics. Readers wishing to discuss the material in this article further are welcome to contact me via e-mail, at andy_wright@uk.ibm.com.

*Andy Wright*
*CICS Change Team*
*IBM (UK)*



## Contributing to *CICS Update*

In addition to *CICS Update*, the Xephon family of *Update* publications now includes *AIX Update*, *DB2 Update*, *RACF Update*, *MQ Update*, *MVS Update*, *Oracle Update*, *RACF Update*, *TCP/SNA Update*, *TSO/ISPF Update*, and *VSAM Update*. Although the articles are of a very high standard, the vast majority are not written by professional writers, and we rely heavily on our readers themselves taking the time and trouble to share their experiences with others

Why not share your expertise and earn some financial reward at the same time? *CICS Update* is looking to swell the number of contributors who send in technical articles, hints and tips, and utility programs, etc. We would also be interested in articles about performance and tuning. If you have an idea for an article contact the editor, Trevor Eddolls, at any of the addresses shown on page 2. A copy of our *Notes for Contributors* is available from our Web site at www.xephon.com/nfc.

# TCP/IP programming with CICS PL/I server and VB6 client

## TCP/IP BASICS

With CICS TCP/IP, remote client systems can invoke CICS transactions. This is the usual mode of operation. The opposite way is also possible, where a CICS transaction is the client and a remote system is the server.

TCP/IP provides a reliable connection between different applications, and a connection is made before sending and receiving any data. Data is sent without errors and is received in the same order that it was sent. For TCP the data is a stream of bytes. A TCP/IP host can communicate with any remote CICS or non-CICS system that has TCP/IP installed.

## CICS TCP/IP SERVER PROCESS CONCEPTS

When you have CICS on the server side you can choose either a concurrent or an iterative server. In this article we will mostly discuss using a concurrent server.

### Iterative server

With an iterative server we can process only one socket at a time. It handles the request for connection and the transaction that should be executed. Iterative servers are simpler and are appropriate for transactions that don't last long.

If the transaction takes more time, a concurrent server would be a better solution because, when one client starts a transaction with the iterative server, another client can't make a call until the first client has finished.

### Listener

The Listener transaction, CSKL, is provided as part of CICS TCP/IP.

The Listener performs several operations.

It will ‘listen’ on the port specified in the configuration file and wait for incoming connection requests from clients. When a connection request arrives, the Listener accepts it and obtains a new socket to pass to the CICS child server application program. It starts the CICS child server transaction and waits for the child server to take the socket and then issues the close call. When this is done, the child server program owns the socket and the Listener has nothing to do with that socket any more. The Listener can process 49 child servers simultaneously.

### Security link module for Listener

The Listener provides the way for security checking to be performed before a CICS transaction is invoked. If a security module is not provided, all transactions can be executed.

If you want to write your own security module, you can call it anything you like, but you have to define it in the configuration dataset. You can write the program in PL/I, COBOL, or Assembler language, but you must define that program in the CICS Program Processing Table (PPT).

Just before the task creation process, the Listener invokes the security module by a CICS LINK, passing a COMMAREA. The Listener passes a data area to the security module that contains information for the module to use for security checking and a 1-byte switch. Your security module should perform a security check and set the switch properly.

When the security module returns, the Listener checks the value of the switch and initiates the transaction if the switch has a value of 1. In the module you can use any CICS statement and function because this is a real CICS LINK module. Remember, excessive programming could cause performance degradation.

An example of a security module:

```
TCPSEC:PROC(POINT) OPTIONS(MAIN);
 /*==================================================================*/
 /* FUNCTION    : SECURITY MODULE                                    */
```

```
/*=====================================================================*/

%INCLUDE DFHAID;
%INCLUDE DFHBMSCA;

DCL (VERIFY,TIME,DATE,ADDR,CSTG,STG,SUBSTR) BUILTIN;

DCL POINT       PTR;
DCL 1 COMAREA                  BASED(POINT),
    2 TRAN_ID   CHAR(4),   /*CICS transaction requested by the client*/
    2 USERDATA  CHAR(4Ø),  /* Data received from the client          */
    2 ACTION    CHAR(2),   /* Method of starting the task:           */
                           /* IC Interval control                    */
                           /* KC Task control                        */
                           /* TD Transient data                      */
    2 INTERVAL  CHAR(6),
       /* Interval requested for IC start control time format hhmmss */
    2 ADR_FAMILY BIN FIXED(15),
                /* Network address family. A value of 2 must be set. */
    2 PORT      BIN FIXED(15),
                          /* The port number of the requester's port */
    2 ADDRESS   BIN FIXED(31),
                          /* The IP address of the requester's host  */
    2 SWITCH1   CHAR(1),   /*1 Pass the socket Not 1 Close connection*/
    2 SWITCH2   CHAR(1),   /* 1 Listener sends message to client     */
         /* Not 1 Security Exit program sends message to client     */
    2 TERMINAL  CHAR(4),   /* Terminal ID                            */
    2 SOCK_ID   BIN FIXED(15),    /* Current socket descriptor       */
    2 USERID    CHAR(8);   /* User ID                                */

DCL 1 DATA                    BASED(ADDR(USERDATA)),
    2 USER_NAME  CHAR(8),
    2 ACCOUNT    PIC'(11)9',
    2 PASSWORD   PIC'(5)9',
    2 FILLER     CHAR(16);

SWITCH1='Ø';
SWITCH2='1';
SELECT (TRAN_ID);
  WHEN('XXØ1')
  DO;
    IF USER_NAME='XXXXXXØ1 ' & ADDRESS = REQUIRED_ADDRESS1  THEN
SWITCH1='1';
  END;
  WHEN('XXØ2')
  DO;
    IF USER_NAME='XXXXXXØ2 ' & ADDRESS = REQUIRED_ADDRESS2  THEN
SWITCH1='1';
```

```
   END;
   OTHERWISE;
 END;
 EXEC CICS RETURN;

END TCPSEC;
```

**Conversion routines**

CICS uses the EBCDIC data format, and TCP/IP networks use ASCII. When exchanging data between CICS and the TCP/IP network, your application programs must use the necessary data conversion modules. CICS TCP/IP provides several conversion routines:

1 An EBCDIC-to-ASCII conversion routine used to convert EBCDIC data within CICS to the ASCII format used in TCP/IP networks and workstations. This is module EZACIC04:

```
CALL EZACICØ4('TCPIPTOASCIIXLAT',TCP_BUF,RETCODE);
```

2 A corresponding ASCII-to-EBCDIC conversion routine, EZACIC05:

```
CALL EZACICØ5('TCPIPTOEBCDICXLT',TCP_BUF,RETCODE);
```

*Child server*

```
CSØ1: PROC OPTIONS(MAIN);
 /*********************************************************************/
 /* FUNCTION : EXAMPLE OF CHILD CICS/PL1 CHILD SERVER                */
/*********************************************************************/

 %INCLUDE DFHAID;
 %INCLUDE DFHBMSCA;
DCL (TIME,DATE,ANY,ADDR,CSTG,VERIFY,STG,SUBSTR,LENGTH) BUILTIN;

 DCL EZASOKET ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
 DCL EZACICØ4 ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;
 DCL EZACICØ5 ENTRY   OPTIONS(ASSEMBLER,RETCODE) EXTERNAL;

 DCL RES     BIN FIXED(15);

 DCL 1  SOKET_FUNCTIONS,
     2 SOKET_ACCEPT       CHAR(16) INIT('ACCEPT          '),
     2 SOKET_BIND         CHAR(16) INIT('BIND            '),
     2 SOKET_CLOSE        CHAR(16) INIT('CLOSE           '),
```

 

```
    2 SOKET_CONNECT      CHAR(16) INIT('CONNECT         '),
    2 SOKET_FCNTL        CHAR(16) INIT('FCNTL           '),
    2 SOKET_GETCLIENTID  CHAR(16) INIT('GETCLIENTID     '),
    2 SOKET_GETHOSTBYADDR   CHAR(16) INIT('GETHOSTBYADDR   '),
    2 SOKET_GETHOSTBYNAME   CHAR(16) INIT('GETHOSTBYNAME   '),
    2 SOKET_GETHOSTID       CHAR(16) INIT('GETHOSTID       '),
    2 SOKET_GETHOSTNAME     CHAR(16) INIT('GETHOSTNAME     '),
    2 SOKET_GETPEERNAME     CHAR(16) INIT('GETPEERNAME     '),
    2 SOKET_GETSOCKNAME     CHAR(16) INIT('GETSOCKNAME     '),
    2 SOKET_GETSOCKOPT      CHAR(16) INIT('GETSOCKOPT      '),
    2 SOKET_GIVESOCKET      CHAR(16) INIT('GIVESOCKET      '),
    2 SOKET_INITAPI         CHAR(16) INIT('INITAPI         '),
    2 SOKET_IOCTL           CHAR(16) INIT('IOCTL           '),
    2 SOKET_LISTEN          CHAR(16) INIT('LISTEN          '),
    2 SOKET_READ            CHAR(16) INIT('READ            '),
    2 SOKET_RECV            CHAR(16) INIT('RECV            '),
    2 SOKET_RECVFROM        CHAR(16) INIT('RECVFROM        '),
    2 SOKET_SELECT          CHAR(16) INIT('SELECT          '),
    2 SOKET_SEND            CHAR(16) INIT('SEND            '),
    2 SOKET_SENDTO          CHAR(16) INIT('SENDTO          '),
    2 SOKET_SETSOCKOPT      CHAR(16) INIT('SETSOCKOPT      '),
    2 SOKET_SHUTDOWN        CHAR(16) INIT('SHUTDOWN        '),
    2 SOKET_SOCKET          CHAR(16) INIT('SOCKET          '),
    2 SOKET_TAKESOCKET      CHAR(16) INIT('TAKESOCKET      '),
    2 SOKET_TERMAPI         CHAR(16) INIT('TERMAPI         '),
    2 SOKET_WRITE           CHAR(16) INIT('WRITE           ');

DCL RETR_LENG    BIN FIXED(15)       INIT(72);
DCL SOCK_ID      BIN FIXED(15);
DCL SOCK_ERR     BIN FIXED(31);
DCL SOCK_RET     BIN FIXED(31);
DCL SOCK_RECV    BIN FIXED(31);
DCL SOCK_LEN     BIN FIXED(31)       INIT(10000);
DCL SOCK_BUF     CHAR(10200);
DCL SOCK_ERRPIC PIC'(8)Z9';

DCL 1 SOCK_CLIENT,
    2 CLI_DOMAIN   BIN FIXED(31)   INIT(2),
    2 CLI_NAME     CHAR(8),
    2 CLI_TASK     CHAR(8),
    2 FILLER       CHAR(20);

DCL 1 SOCK_SERVER,
    2 SOCKET_ID           BIN FIXED(31),
    2 LSTN_NAME           CHAR(8),
    2 LSTN_SUBTASKNAME    CHAR(8),
    2 CLIENT_DATA         CHAR(35),
    2 FILLER              CHAR(1),
```

 

```
     2 SOCKADDR_IN,
       3 FAMILY            BIN FIXED(15),
       3 PORT              BIN FIXED(15),
       3 IP_ADDR           BIN FIXED(31),
       3 RESERVE           CHAR(8);

EXEC CICS IGNORE CONDITION LENGERR;
/* This retrieves the data passed by the START command    */
/* in the concurrent server (Listener) program. This      */
/* data includes the socket descriptor and the concurrent */
/* server client ID as well as optional additional data   */
/* from the client and we are now using that technology   */
/* because in most cases this is enough                   */

 EXEC CICS RETRIEVE INTO(SOCK_SERVER) LENGTH(RETR_LENG) RESP(RES);
 IF RES¬=DFHRESP(NORMAL)
    THEN CALL SEND_DATA('ERROR ON RETRIEVE STATEMENT_END',Ø);

/* This acquires the newly created socket from the             */
/* concurrent server. The TAKESOCKET parameters must           */
/* specify the socket descriptor to be acquired and            */
/* the client id of the concurrent server. This information    */
/* was obtained by the EXEC CICS RETRIEVE command.             */

 SOCK_ID=SOCKET_ID;
 CLI_NAME=LSTN_NAME;
 CLI_TASK=LSTN_SUBTASKNAME;
 CALL EZASOKET(SOKET_TAKESOCKET,SOCK_ID,SOCK_CLIENT,SOCK_ERR,SOCK_RET);
 IF SOCK_RET<Ø THEN CALL SEND_DATA(' ERROR ON TAKESOKET
STATEMENT_END',Ø);
 SOCK_ID=SOCK_RET;


 IF VERIFY(SUBSTR(CLIENT_DATA,5,5),'123456789Ø')¬=Ø
    THEN CALL SEND_DATA('ERROR ON CLIENT DATA_END',Ø);
 TRKEY = SUBSTR(CLIENT_DATA,5,5);

 EXEC CICS READ DATASET('FILE_NAME') RIDFLD(TRKEY) INTO(RECORD_VAR)
RESP(RES);
 IF RES=DFHRESP(NOTFND) THEN CALL SEND_DATA('ERROR READING FILE_END',1);
 IF RES¬=DFHRESP(NOTFND) & RES¬=DFHRESP(NORMAL)
    THEN CALL SEND_DATA('FILE CLOSED_END',1);

CALL SEND_DATA(SUBSTR(RECORD_VAR,56,15));

 SEND_DATA:PROC(MESSAGE,LOG);
   DCL MESSAGE CHAR(1ØØØØ);
   DCL LOG    DEC FIXED(1);
```

 

```
   SOCK_LEN = 15;
   CALL EZACICØ4(MESSAGE,SOCK_LEN);
   /* Conversation with the client                                    */
   SOCK_BUF=MESSAGE;
   CALL
EZASOKET(SOKET_WRITE,SOCK_ID,SOCK_LEN,SOCK_BUF,SOCK_ERR,SOCK_RET);
   IF SOCK_RET<Ø THEN EXEC CICS ABEND ABCODE('CSØ1');

   /* Terminates the connection and releases  */
   /* the socket resources when finished.     */
   DELAY(1ØØ);
   CALL EZASOKET(SOKET_CLOSE,SOCK_ID,SOCK_ERR,SOCK_RET);
   IF SOCK_RET<Ø THEN EXEC CICS ABEND ABCODE('CSØ1');

   EXEC CICS RETURN;

 END SEND_DATA;

 END CSØ1;
```

**Sockets**

The socket API is a collection of socket calls that enable you to:

- Perform the communication functions between application programs.
- Set up and establish connections to other users on the network.
- Send and receive data to and from other users.
- Close down connections.

A socket is an end point for communication that can be named and addressed in a network. From an application program perspective, a socket is a resource that is allocated by the TCP/IP address space. A socket is represented to the program by an integer called a socket descriptor.

MVS supports three socket types – stream, datagram, and raw. While CICS supports stream and datagram sockets, stream sockets provide the most reliable form of data transfer offered by TCP/IP. Stream sockets transmit data between TCP/IP hosts that are already connected to one another. Data is transmitted in a continuous stream. There is no record length or newline character between data. Communicating processes must agree on a scheme to ensure that both client and server have received all data. One way of doing this is for the sending process to send the length of the data followed by the data itself or we can send something like '_END' to notify the client or server that this is the end of the data.

An address family defines a specific addressing format. Applications that use the same addressing family have a common scheme for addressing sockets. TCP/IP for CICS supports the AF_INET address family:

```
DCL 1 SOCKADDR_IN,
    2 FAMILY       BIN FIXED(15),  /* Always 2 AF_INET family */
    2 PORT         BIN FIXED(15),  /* Application port number */
    2 IP_ADDR      BIN FIXED(31),
 /* Internet address of the network interface used by the application */
    2 RESERVE      CHAR(8);        /* All zeros               */
```

A port is a 16-bit integer that defines a specific application, within an IP address, in which several applications use the same network interface. The port number is a qualifier that TCP/IP uses to route incoming data to a specific application within an IP address.

Tasks use the GIVESOCKET and TAKESOCKET functions to pass sockets from parent to child. The task passing the socket uses GIVESOCKET, and the task receiving the socket uses TAKESOCKET.

Once a client has been connected to the server, and the socket has been transferred from the main task (parent listener) to the subtask (child server), the client and server exchange application data, using READ/WRITE calls.

*VB client*

```
Private Sub cmdsend_Click()

  DIM As WinsockKlijent AS mswinsock

  ALL_RECEIVED_DATA =  ""
  WinsockKlijent.Protocol = sckTCPProtocol
  WinsockKlijent.RemoteHost = "xxx.xx.x.x"
  WINSOCKKLIJENT.REMOTEPORT = YY  ' Integer
```

```
  WinsockKlijent.Connect

  WINSOCKKLIJENT.SENDDATA "CSØ1,XXXXXXØ1" & ACCOUNT ' String

  WinsockKlijent.Close

End Sub

Private Sub WinsockKlijent_DataArrival(ByVal bytesTotal As Long)
     WinsockKlijent.GetData RECEIVED_DATA, vbString
     ALL_RECEIVED_DATA = ALL_RECEIVED_DATA & RECEIVED_DATA
End Sub
```

*Nebojsa Cosic (Nebojsa.K.Cosic@verizon.com, nesasone@yahoo.com)*
*System Analyst*
*Ivan Bugarinovic*
*System Analyst*
*Pinkerton Computer Consultants Inc (USA)*

# CICSPlex SM API program written in REXX

In a previous article (see *Utilizing the power of the CICSPlex SM Web user Interface*, Issue 191, October 2001) we have looked at how we can use the CICSPlex SM Web User Interface to determine which library a given program in a CICS region was loaded from and the RPL concatenation list for that region, and to identify which libraries contain the given module. In the previous example, the last step was performed by using TSO base facilities.

In this article I provide a programmatic method which also performs the last step. (Steps 1 and 2 are again single system image in concept. It is the third step that requires locality of the resource.)

The example is a CICSPlex SM API program written in REXX.

## CICSPLEX SM APPLICATION PROGRAMMING INTERFACE

CICSplex SM provides an application programming interface to all its function. This API can be used to write applications of varying complexity from simple one-off scripts, through small pieces of automation, through to large applications such as a Web browser interface (hey, we already did that one!). This API provides the same Single System Image characteristics as are found via the other user interfaces.

The CICSPlex SM API can be invoked from CICS, an MVS batch program, TSO, and NetView. Two language bindings are provided, a REXX binding, and an EXEC CPSM binding (similar to EXEC CICS). Programs can be written in COBOL, Assembler, C, or PL/I.

## OBJECTS AND OBJECT INSTANCES

The CICSplex SM API can be used to manipulate the objects that exist in the CICSplex SM definitions and CICS run-time objects; and events propagated by RTA and CICSPlex SM are examples of such objects. The object classes, attribute names, types, and action commands are defined in the *Resource Tables* reference manual. Object instances are typically returned by requests for data. For EXEC CPSM programs, Dsects are provided to map onto the returned data, eg the usecount for a program may be referenced by PROGRAM_USECOUNT.

Resultsets are sets of objects of the same type with the same context. They reside in the management environment storage. Reference to a resultset is by its resultset token (returned by the command creating it). A resultset also has an implicit instance pointer that points to the current instance being referenced. Resultsets can be QUERY'd or DISCARDed.

Extensive facilities are provided for subsetting the data in a resultset, both at creation and for subsequent processing. This is achieved through FILTERs which specify sets of attribute, attribute value pairs, along with a comparator. An example would be 'tranid=P* AND PROGRAM=PAY* AND STATUS=ENABLED'. Elements in a resultset can also be MARKed/UNMARKed. Subsequent commands can be instructed to act only on MARKed instances. Specific entries can be LOCATed/DELETEd using pointer location manipulation, MARKed or FILTER properties.

In order to access the data in a resultset, instances must first be FETCHed into local storage.

Resultsets can be ORDERed according to specified criteria in ascending or descending sequence. They can also be COPYed to a new resultset under the control of a FILTER or MARKed entries. Finally, one can create a summarized resultset according to user specified summarization rules (eg USECOUNT and AVG, MIN, MAX, etc).

So far all the commands have interacted with the management environment, without touching the actual managed CICS systems. There are also commands that can manipulate such resources and create resultsets.

## MANAGED OBJECTS

Managed objects can be manipulated either synchronously or asynchronously:

- GET/REFRESH – commands get object instances according to FILTER criteria.
- REFRESH – performs a similar function, but refreshes data based on an existing resultset.
- SET – sets the attributes of objects identified in the resultset, eg 'status=disabled,openstatus=closed' would cause the target files to be closed and disabled.
- PERFORM SET – performs actions upon a resultset, eg 'SHUTDOWN IMMEDIATE'.
- PERFORM OBJECT – this is a combined GET followed by PERFORM SET.

## THE API PROGRAM

The input to the program is contained in the following variables:

- W_Context is the CICSplex we are interested in (PJPLX).
- W_Scope is the CICS system we are concerned with (a single region in this case).
- W_ProgramName is the program we want the information about.

They have been hard-coded in this example, but the EXEC could be trivially modified to take them as input parameters when running the EXEC.

The release of CICSPlex SM is coded in W_Version (0210). This is the release that the program declares it understands.

The EXEC runs under TSO. This is because it utilizes the OUTTRAP and LISTDS functions provided under TSO. It is only the CheckLibs routine that requires TSO and locality of the CICS system. The rest is purely Single System Image in nature, and could provide information about any CICS system in the CICSPlex (even if it were on the other side of the globe).

As you can see, the program is straightforward and takes little time to produce. If you'd like to see other examples of CICSPlex SM API programs, a serverpac (CS13) can be downloaded from http://www-4.ibm.com/software/ts/cics/txppacs/txpc2.html#cat2.

```
/* REXX */
/*-------------------------------------------------------------------*/
/* CICSPlex SM API program to identify                               */
/*                                                                   */
/* 1/ Library from which a load module was loaded                    */
/* 2/ Which libraries are in the system's concatenation list         */
/* 3/ Which libraries in the list contain the named program          */
/*                                                                   */
/* Requires TSO environment to execute                               */
/*-------------------------------------------------------------------*/
Address 'TSO'
Parse Value Ø Ø With W_Response W_Reason .
W_Context = 'PJPLX   '             /* The plex containing the system */
W_Scope   = 'IYCWZCGF'             /* the specific system's name     */
W_Version = 'Ø22Ø'                 /* The CPSM version               */
W_ProgramName= 'EYU9XLOP'          /* The program's name             */
failed = -1                        /* failure return code            */

Say 'Initializing API...'
rc = EYUINIT()
if rc = Ø then do
  Call ConnectToCPSM               /* get a connection thread        */
  if rc = Ø then do
    Call ObtainRPLNo               /* Get RPLNo prog was loaded from */
    if rc = Ø then do
      Say W_ProgramName ' in system ' W_Scope 'loaded from ' W_RPLNo
      Call ObtainRPLList           /* Obtain RPLList for the system  */
      if rc = Ø then do
        Say 'RPL list for CICS system ' W_Scope ' is '
        do i = 1 to W_RPLList.Ø
          n = i - 1                /* RPLs start numbering at Ø      */
          Say n ' ' W_RPLList.i
        end
        Call Terminate             /* Done with CPSM                 */
        Call CheckLibs             /* Check libraries for duplicates */
        end
      end
    end
  end
```

```
else
  Say 'Unexpected response from EYUINIT ' rc

Exit

/*-------------------------------------------------------------------*/
/* Connect to CICSPlex SM                                            */
/*-------------------------------------------------------------------*/
ConnectToCPSM:

Say 'Establishing connection...'
rc = EYUAPI('CONNECT' ,
            'CONTEXT('W_Context')',
            'SCOPE('W_Scope')',
            'VERSION('W_Version')',
            'THREAD(W_Thread)' ,
            'RESPONSE(W_Response)' ,
            'REASON(W_Reason)')
if rc = Ø then do
  if W_Response <> EYURESP(OK) then do
    Say 'Bad Response from CONNECT ' W_Response W_Reason
    rc = failed
    end
  end
else do
  Say 'Unexpected response from EYUAPI ' rc
  end

Return
/*-------------------------------------------------------------------*/
/* Obtain the RPL Number from the Program resource                   */
/*-------------------------------------------------------------------*/
ObtainRPLNo:

Say 'Get the PROGRAM resource table...'
W_Criteria    = 'PROGRAM=' W_ProgramName '.'
W_CriteriaLen = LENGTH(W_Criteria)
rc = EYUAPI('GET OBJECT(PROGRAM)' ,
                'THREAD(W_Thread)' ,
                'CRITERIA(W_Criteria)' ,
                'LENGTH('W_CriteriaLen')' ,
                'RESULT(W_Result)' ,
                'COUNT(W_Reccnt)' ,
                'RESPONSE(W_Response)' ,
                'REASON(W_Reason)')
if rc = Ø then do
  if W_Response = EYURESP(OK) then do
    select
```

```
when (W_Reccnt = 1) then do
  Say 'Querying resource ...'
  rc = EYUAPI('QUERY OBJECT(PROGRAM) THREAD(W_Thread)',
              'RESULT(W_Result) DATALENGTH(W_Into_ObjectLen)',
              'RESPONSE(W_Response) REASON(W_Reason)')
  if rc = Ø then do
    if W_Response = EYURESP(OK) then do
      Say 'Fetching PROGRAM entry...'
      rc = EYUAPI('FETCH INTO(W_Into_Object)' ,
                  'LENGTH(W_Into_ObjectLen)' ,
                  'THREAD(W_Thread)' ,
                  'RESULT(W_Result)' ,
                  'RESPONSE(W_Response)' ,
                  'REASON(W_Reason)')
      if rc = Ø then do
        if W_Response = EYURESP(OK) then do
          Say 'Parsing output...'
          rc = EYUAPI('TPARSE OBJECT(PROGRAM)' ,
                      'PREFIX(PGM)' ,
                      'STATUS(W_Response)' ,
                      'VAR(W_Into_Object.1)' ,
                      'THREAD(W_Thread)')
          if rc = Ø then do
            if W_Response = 'OK' then do
              W_RPLNo = PGM_RPLID
              end
            else do
              Say 'Bad TParse response ' W_Response
              rc = failed
              end
            end
          else do
            Say 'Unexpected response from EYUAPI ' rc
            end
          end
        else do
          Say 'Bad Fetch response ' W_Response W_Reason
          rc = failed
          end
        end
      else do
        Say 'Unexpected response from EYUAPI ' rc
        end
      end
    else do
      Say 'Bad Query Response ' W_Response W_Reason
      rc = failed
      end
```

```
        end
      else do
        Say 'Unexpected response from EYUAPI ' rc
        end
      end
    when (W_Reccnt > 1) then do
      Say 'To Many entries ' W_Reccnt
      rc = failed
      end
    otherwise do
      Say 'Entry not there'
      rc = failed
      end
    end                   /* end of select */
    end
  else do
    Say 'Bad Get response ' W_Response W_Reason
    rc = failed
    end
  end
else do
  Say 'Unexpected response from EYUAPI ' rc
  end

Return

/*------------------------------------------------------------------*/
/* Obtain the RPL list from the Program resource                    */
/*------------------------------------------------------------------*/
ObtainRPLList:

Say 'Get the RPLLIST resource table...'
rc = EYUAPI('GET OBJECT(RPLLIST)' ,
                'THREAD(W_Thread)' ,
                'RESULT(W_Result)' ,
                'COUNT(W_Reccnt)' ,
                'RESPONSE(W_Response)' ,
                'REASON(W_Reason)')
if rc = Ø then do
  if W_Response = EYURESP(OK) then do
    Say 'Querying Object ...'
    rc = EYUAPI('QUERY OBJECT(RPLLIST) THREAD(W_Thread)',
                'RESULT(W_Result) DATALENGTH(W_Into_ObjectLen)',
                'RESPONSE(W_Response) REASON(W_Reason)')
    if rc = Ø then do
      if W_Response = EYURESP(OK) then do
        Say 'Fetching ' W_Reccnt ' RPLLIST entries ...'
        W_RPLList.Ø = W_Reccnt
```

```
do i = 1 to W_Reccnt
  Say 'Fetching entry...'
  rc = EYUAPI('FETCH INTO(W_Into_Object)' ,
              'LENGTH(W_Into_ObjectLen)' ,
              'THREAD(W_Thread)' ,
              'RESULT(W_Result)' ,
              'RESPONSE(W_Response)' ,
              'REASON(W_Reason)')
  if rc = Ø then do
    if W_Response = EYURESP(OK) then do
      Say 'Parsing entry ...'
      rc = EYUAPI('TPARSE OBJECT(RPLLIST)' ,
                  'PREFIX(RPL)' ,
                  'STATUS(W_Response)' ,
                  'VAR(W_Into_Object.1)' ,
                  'THREAD(W_Thread)')
      if W_Response = 'OK' then do
        W_RPLLIST.i = RPL_DSNAME
        end
      else do
        Say 'Bad TParse response ' W_Response
        rc = failed
        end
      end
    else do
      Say 'Bad Fetch response ' W_Response W_Reason
      rc = failed
      end
    end
  else do
    Say 'Bad response from EYUAPI ' rc
    end
end      /* end do i */
end
else do
  Say 'Bad Query Response ' W_Response W_Reason
  rc = failed
  end
end
else do
Say 'Bad response from EYUAPI ' rc
end
end
else do
Say 'Bad Get response ' W_Response W_Reason
rc = failed
end
end
```

 

```
else do
  Say 'Bad response from EYUAPI ' rc
  end

Return

/*-------------------------------------------------------------------*/
/* Terminate API Connection                                          */
/*-------------------------------------------------------------------*/
Terminate:

Say 'Terminating connection to CPSM ...'
rc = EYUAPI('TERMINATE RESPONSE(W_Response) REASON(W_Reason)')
rc = EYUTERM()

Return

/*-------------------------------------------------------------------*/
/* Check libraries for duplicates                                    */
/*-------------------------------------------------------------------*/
CheckLibs:

do i = 1 to RPLList.Ø
  rc = OUTTRAP('Result.')            /* divert TSO output            */
  dsn = RPLList.i
  'TSO LISTDS' '''dsn'''
  do j = 7 to Result.Ø               /* Skip header onfo             */
    if W_ProgramName = Result.j then do
       Say W_ProgramName ' exists in ' dsn
       end
  end
end

Return X */
```

## OUTPUT

The program produces the following output:

```
Initializing API...
Establishing connection...
Get the PROGRAM resource table...
Querying resource ...
Fetching PROGRAM entry...
Parsing output...
EYU9XLOP  in system  IYCWZCGF loaded from  8
Get the RPLLIST resource table...
```

```
Querying Object ...
Fetching  2Ø  RPLLIST entries ...
Fetching entry...
Parsing entry ...
Fetching entry...
Parsing entry ...
….
….
Fetching entry...
Parsing entry ...
RPL list for CICS system  IYCWZCGF  is
Ø   CPSMDEV.PJOHNSO.LOAD
1   CPSMDEV.TEST.LOAD
2   CPSMDEV.BSF.LOAD
3   CPSMDEV.DUMMY.LOAD
4   UTL.PJOHNSO.LOAD
5   CPSMDEV.TABLE62Ø.LOAD
6   PUBPLU.CPSM.LOAD
7   PUBPLU.CPSM.TABLES
8   BLDBSF.PLUXA.SEYULOAD
9   PP.ADLE37Ø.OS39Ø21Ø.SCEECICS
1Ø   PP.ADLE37Ø.OS39Ø21Ø.SCEERUN
11   PP.PLI.V23Ø.PLIBASE
12   PP.PLI.V23Ø.PLILINK
13   PP.PLI.V23Ø.SIBMBASE
14   BLDBSF.PLUXA.SDFHLOAD
15   BLDBSF.PLUXA.SDFHLOAD
15   BLDBSF.PLUXA.SDFHLOAD
16   BLDBSF.PLUXA.SDFHLOAD
17   BLDBSF.PLUXA.SDFHAUTH
18   BLDBSF.PLUXA.SDFHAUTH
19   BLDBSF.PLUXA.SDFHAUTH
Terminating connection to CPSM ...
EYU9XLOP exists in  BLDBSF.PLUXA.SEYULOAD
```

*Dr Paul Johnson*
*CICS Transaction Server Systems Management Planning/Development*
*IBM (UK)*


*Have you come across any undocumented features in CICS TS 1.3? Please share your discovery with others – send your finding to Trevor Eddolls at trevore@xephon.com.*

# Printing TSO files under CICS

The following utility was created to print TSO files in a more elegant and aesthetic way than allowed by the TSO printing system. When I started looking for an alternative way of printing files, the answer became obvious – CICS. I already had some experience of controlling CICS printers by sending them PCL commands, form feeds, line feeds, etc. So, to print a TSO file using CICS, I just needed to devise a method of having the files to print available to CICS and also to trigger a CICS transaction from TSO.

The process works as follows: I start a REXX EXEC against the file to print. That file can be a PDS member or a sequential with fixed LRECL. The EXEC creates and submits a job with two steps – firstly, program PRINTC1 reads the file to print and copies it to a VSAM RRDS that is known to CICS but which is closed by default; secondly, program PRINTC2 communicates with CICS via VTAM/APPC and triggers a transaction associated with program PRINTC3, passing a parameter line. Already under CICS, PRINTC3 acquires the target printer, opens the RRDS file, and starts a second transaction against the printer. That second transaction (program PRINTC4) reads the RRDS file into an array and closes it, to make it available to TSO without more delay. Then it goes through the lines to print, formats them according to certain rules (font type and size, page headers, etc) and sends it to the printer. I must say that, since I started using this method, I have never needed to use TSO printing again.

## HOW TO INSTALL PRINTC

To install this application, you will need to take the following steps.

Define a VSAM RRDS file with a name of your choice, with a fixed LRECL of 140, and with the attribute REUSE. This attribute is essential. Choose a CICS region and a PCL-compatible printer attached to it. Declare the RRDS file under CICS, and leave it always closed. Define transactions PTC1 and PTC2 (or any other names of your choice) associated with programs PRINTC3 and PRINTC4, respectively. Compile these two programs for CICS.

At the beginning of PRINTC, set the appropriate variables to the name of those two transactions. Also set the name of the RRDS file and its CICS DDname, and also the printer name.

Now go to program PRINTC2, the program that communicates with CICS via VTAM/APPC. There you must define the contents of three Assembler variables:

- &LUNAM is the name of the CICS region under VTAM.
- &LOGMOD is the VTAM logmode.

&ACBNAM is the VTAM ACB name.

Go to SYS1.VTAMLST. There you must define ACBNAM, by creating a member with the following definition, or by inserting it into an existing one and activating it.

Below is an example, which you may have to modify according to your needs:

```
TSCICSB1 APPL  ACBNAME=TSCICSB1,
               APPC=YES,AUTOSES=5,EAS=4,
               DDRAINL=NALLOW,DLOGMOD=SNASVCMG,
               DMINWNL=5,DMINWNR=5,DRESPL=NALLOW,
               DSESLIM=10,LMDENT=19,
               MODETAB=EDITAB,PARSESS=YES,
               SECACPT=CONV,SRBEXIT=NO,VPACING=1
               STATOPT='APPL APPC/MVS'
```

Once more under CICS, go to CEDA and define a session and a connection, like the following example:

```
CEDA  View Sessions( TSCICSB1 )
    Sessions       : TSCICSB1
    Group          : GRPRINTC
    DEscription    :
   SESSION IDENTIFIERS
    Connection     : TSB1
    SESSName       :
    NETnameq       :
    MOdename       : INTER
   SESSION PROPERTIES
    Protocol       : Appc              Appc | Lu61 | Exci
    MAximum        : 002 , 001         0-999
```

 

```
    RECEIVEPfx     :
    RECEIVECount   :                     1-999
    SENDPfx        :
    SENDCount      :                     1-999
    SENDSize       : Ø1Ø24               1-3Ø72Ø
 +  RECEIVESize    : Ø1Ø24               1-3Ø72Ø


CEDA  View Connection( TSB1 )
    Connection     : TSB1
    Group          : GRPRINTC
    DEscription    :
   CONNECTION IDENTIFIERS
    Netname        : TSCICSB1
    INDsys         :
   REMOTE ATTRIBUTES
    REMOTESYSTem   :
    REMOTEName     :
    REMOTESYSNet   :
   CONNECTION PROPERTIES
    ACcessmethod   : Vtam                Vtam | IRc | INdirect
    PRotocol       : Appc                Appc | Lu61 | Exci
    Conntype       :                     Generic | Specific
    SInglesess     : No                  No | Yes
    DAtastream     : User                User | 327Ø | SCs |
STrfield | Lms
 +  RECordformat   : U                   U | Vb
```

Note how the names are related to the VTAM definitions and to the variables in PRINTC2. For a person unused to VTAM, this part can be quite tricky to configure properly.

Now, go back to TSO and compile programs PRINTC1 and PRINTC2 to a LOADLIB of your choice, and set the name of that LOADLIB also at the beginning of PRINTC.

You must also think of RACF-related restrictions – transaction and dataset authorizations, etc. I am not a RACF person, and I don't recall exactly what are the needs in this field, but there are a few things to take care of.

If everything went OK, you can start printing. If nothing happens the first time, look at the output of the job: is everything OK or did the second step display an error message? If it did, it is a VTAM-related problem. Go to the PRINTC2 program and see which macro issued the error. Check your VTAM definitions, to see whether the ACB is active and both the session and the connection are OK under CICS. If VTAM communicates with CICS and succeeds in starting the transaction there but nothing is printed, then you can try to CEDF both programs. For the second, it is easy, since you know the printer name. For the first, you don't know beforehand the virtual terminal name (it will be something like -998). A useful trick in this case is to insert an EXEC CICS DELAY(20) at the beginning of the program. Then you have twenty seconds to catch the virtual terminal name with *CEMT I TAS* and CEDF it, so you can observe what happens.

## WHAT KIND OF PRINT YOU GET

PRINTC was designed to work with a laser printer and A4 sheets. If you use a different paper size, you might need to adjust the font sizes in PRINTC4.

PRINTC has four pre-defined print styles, each of them with or without a header. The print style is related to the LRECL of the file: for an 80-byte file, I use Courier pitch 11. For 132 bytes, I use a smaller letter, to fit the columns in the page vertically: LetterGothic bold pitch 18 (all printing is done in portrait). For intermediate LRECLs, like 120 column listings, I use intermediate values in order to fit things nicely in the A4 sheet. I also adjust the left margin slightly.

When you launch PRINTC against a file to print, you must answer whether the file contains control characters or not. If it does, answer 'C', otherwise just press *Enter*.

Control characters are a way of controlling page advance (character '1') and line advance (character '0', jump one line, character '-', jump two lines). These control characters are located in the left-most column of a file, and that column cannot contain anything else.

My program behaves differently. If the file has CCs, I print only from the second column onwards, and honour all the CC directives. I do not print any header, or create any page break, because I assume you are printing a fully-formatted listing.

If the file does not have CCs, then I print everything starting at column one, and I insert a header in each page with the full file name, the date and time of printing, and a page number. I also create a page break every 61 lines.

I have PRINTC working with a laser printer attached to a controller by means of an AX-COBRA unit. To send the printer PCL commands, I must enclose them in a sequence of '&&??%%' and '&&??000', as you can see if you look for PGHEADxx items in the working storage of PRINTC4. Eventually, a different attach method will require some changes in these strings. If you change them, be aware that the length of these items must be reflected correctly in the corresponding 'length' variables: they have the same name as the base items, but suffixed by an 'L', and they are declared at 'level 77'.

Another important thing to consider is how your printer behaves with a 132-byte file. My program inserts a 'carriage return, linefeed' sequence (X'0D15') after each line, when I build the printer buffer, except for 132-byte files, because the printer is set up in such a way that it linefeeds itself while it prints a line with 132 bytes. If I inserted the X'0D15', I would have an extra blank line between each line printed. I have a flag at PRINTC4 called NO-LINEFEED-132 (the last 77 item). If you want to enable the linefeed insertion for 132 byte files, change the value to something other than 1.

You can use this application with more than one printer. For that, you need to modify the EXEC to accept it as a parameter. You can also print from several TSO regions. For that, all you need to do is to '/XEQ' the job for execution to the region where CICS (and the RRDS file) is located.

## PRINTC SOURCE CODE

```
/* REXX MVS ********************************************************/
/*                                                                 */
/*  PRINTC Prints TSO files to a PCL printer under CICS            */
/*  PRINTC is made up of the following programs:                   */
/*                                                                 */
/*   PRINTC   - This REXX                                          */
/*   PRINTC1  - Asm to repro TSO file to VSAM RRDS file            */
/*   PRINTC2  - Asm to communicate with CICS via APPC              */
/*   PRINTC3  - Asm started under CICS by APPC                     */
/*   PRINTC4  - COBOL to print file under CICS                     */
/*                                                                 */
```

```
/*******************************************************************/

ficvsam = "ADRTD.CICS7.DSPRINT"     /* VSAM RRDS dataset name        */
ficcics = "DSPRINT  "               /* CICS VSAM DDname              */
trans1  = "PTC1"                    /* CICS trans. for prog. PRINTC3 */
trans2  = "PTC2"                    /* CICS trans. for prog. PRINTC4 */
printer = "LJP8"                    /* CICS printer name             */
jobtemp = userid()".PRINTC.JOB"     /* temporary file for job        */
loadlib = "TREDSA.TSO12.LOADLIB"    /* Load containing PRINTC1/C2    */

arg ficheiro ee .
if ficheiro = "" then do
   say "File to print?"
   pull ficheiro .
   if ficheiro = "" then exit
end
fic = strip(ficheiro,,"'")
parse var fic pds "(" lixo
zz = msg(off)
xx = listdsi(ficheiro)
if sysreason = 12 then do
   say "VSAM files are not supported"
   signal saida
end
if syslrecl < 90                   then lrecl = "080"
if syslrecl < 111 & syslrecl > 89  then lrecl = "110"
if syslrecl < 130 & syslrecl > 110 then lrecl = "120"
if syslrecl > 129                  then lrecl = "132"
if ee = "" then do
  say "If the file has Control Chars at column1, enter CC"
  say "otherwise just hit ENTER"
  pull ee .
end
if ee <> "CC" then ee = "  "
parm  = left(trans1,4) || left(trans2,4) || left(fic,44)
parm  = parm || ee || lrecl || left(ficcics,8) || left(printer,4)
upper parm

"free  dd(jobdd)"
"alloc dd(jobdd) da('"jobtemp"') new reuse
       lrecl(80) blksize(8000) recfm(f,b)
       dsorg(ps) space(1 1) tracks delete"
if rc<>0 then do
   say "Error "rc" allocating file" jobtemp
   signal saida
end

dropbuf
queue "//PRINTCC JOB MSGCLASS=X,CLASS=A,MSGLEVEL=(1,1)"
queue "//*"
```

```
queue "//STEPØ1   EXEC PGM=PRINTC1"
queue "//STEPLIB  DD DISP=SHR,DSN="loadlib
queue "//SYSPRINT DD SYSOUT=* "
queue "//ENTRADA  DD DISP=SHR,DSN="fic
queue "//SAIDA    DD DISP=SHR,DSN="ficvsam
queue "//*"
queue "//STEPØ2   EXEC PGM=PRINTC2"
queue "//STEPLIB  DD DISP=SHR,DSN="loadlib
queue "//SYSIN    DD * "
queue parm
queue "/*"
queue "//SYSPRINT DD SYSOUT=* "
queue ""
"execio * diskw jobdd (finis"
"submit '"jobtemp"'"

saida:
"free da('"jobtemp"')"
"free dd(jobdd)"
exit
```

## PRINTC1 SOURCE CODE

```
*===================================================================*
*                                                                   *
* PRINTC1 - Reads from ENTRADA (DCB) and writes to SAIDA (ACB-RRDS)  *
*           The RRDS file must be defined with REUSE and LRECL 14Ø.  *
*                                                                   *
*===================================================================*
&PROGRAM SETC  'PRINTC1'
&PROGRAM AMODE 31
&PROGRAM RMODE 24
&PROGRAM CSECT
         SAVE  (14,12)
         LR    R12,R15
         USING &PROGRAM,R12
         ST    R13,SAVEA+4
         LA    R11,SAVEA
         ST    R11,8(R13)
         LR    R13,R11
         B     OPENFILS
         DC    CL16' &PROGRAM 1.Ø'
         DC    CL8'&SYSDATE'
*
OPENFILS DS    ØH
         OPEN  (SYSPRINT,OUTPUT)
         OPEN  (ENTRADA,INPUT)
         LTR   R15,R15
         BNZ   ERRO
```

```
         OPEN  (SAIDAA,OUTPUT)
*
LEITURA  EQU   *
         GET   ENTRADA,MSG1
         PUT   RPL=SAIDAR
         B     LEITURA
*
EXITØ    EQU   *
         CLOSE ENTRADA
         CLOSE SAIDAA
         CLOSE SYSPRINT
         L     R13,SAVEA+4
         LM    R14,R12,12(R13)
         SR    R15,R15
         BR    R14
*
ERRO     EQU   *
         PUT   SYSPRINT,=C'>>> ERROR OPENING INPUT FILE '
         B     EXITØ
*
SAIDAA   ACB   DDNAME=SAIDA,                                           X
               MACRF=(KEY,SEQ,OUT,RST)
SAIDAR   RPL   ACB=SAIDAA,                                             X
               OPTCD=(SEQ,KEY,SYN,NUP,MVE),                            X
               RECLEN=14Ø,                                             X
               AREA=MSG1,                                              X
               ARG=CHAVE
*
ENTRADA  DCB   DSORG=PS,MACRF=(GM),EODAD=EXITØ,                        X
               DDNAME=ENTRADA
*
SYSPRINT DCB   DSORG=PS,MACRF=(PM),LRECL=8Ø,                           X
               DDNAME=SYSPRINT
*
         LTORG
SAVEA    DS    18F
CHAVE    DC    F'Ø'
MSG1     DS    CL14Ø
         YREGS
         END
```

## PRINTC2 SOURCE CODE

```
***********************************************************************
*                                                                     *
*  PRINTC2 - Communicates with CICS via APPC and starts transaction  *
*            "trans1", as defined in PRINTC REXX.                     *
*                                                                     *
***********************************************************************
```

```
*
&LOGMOD  SETC  'INTER'                    VTAM Logmode    <<<<<====
&LUNAM   SETC  'AVCICS7'                  VTAM CICS name  <<<<<====
&ACBNAM  SETC  'TSCICSB1'                 VTAM ACB name   <<<<<====
&PROGRAM SETC  'PRINTC2'
&PROGRAM AMODE 24
&PROGRAM RMODE 24
&PROGRAM CSECT
         SAVE  (14,12)
         LR    R12,R15
         LA    R11,2Ø48(R12)
         LA    R11,2Ø48(R11)
         USING &PROGRAM,R12,R11
         ST    R13,SAVEA+4
         LA    R1Ø,SAVEA
         ST    R1Ø,8(R13)
         LR    R13,R1Ø
         B     GETPARM
         DC    CL16' &PROGRAM 2.Ø'
         DC    CL8'&SYSDATE'
SAVEA    DS    18F
*
GETPARM  DS    ØH
         OPEN  (SYSPRINT,OUTPUT)
         OPEN  (SYSIN,INPUT)
         GET   SYSIN,PARMEXEC
         CLOSE SYSIN
         LA    R3,FICACB
         USING IFGACB,R3
         LA    R2,FICRPL
         USING IFGRPL,R2
*
         OPEN  FICACB
         LTR   R15,R15
         BNZ   RETCODØ1
         SETLOGON OPTCD=START,                                         X
               RPL=FICRPL,                                             X
               ACB=FICACB
         LTR   R15,R15
         BNZ   RETCODØ2
         LA    R4,FICCNOS1
         USING ISTSLCNS,R4
         MVC   SLCSESSL,=X'ØØØ2'        Max number of lu sessions
         MVC   SLCMCWL,=X'ØØØ1'         Local contention winnrs
         MVC   SLCMCWP,=X'ØØØØ'         Partner contention winners
         MVC   SLCPARMS,=X'ØØ'          Session deactivated this side
*
         LA    R8,FICRPL6
         USING ISTRPL6X,R8
         APPCCMD CONTROL=OPRCNTL,                                      X
```

```
               QUALIFY=CNOS,                                           X
               RPL=FICRPL,                                             X
               AAREA=(R8),                                             X
               ACB=FICACB,                                             X
               LOGMODE=&LOGMOD,                                        X
               LUNAME=&LUNAM,                                          X
               AREA=(R4),RECLEN=7
         LTR   R15,R15
         BNZ   RETCODØ3
         LTR   RØ,RØ
         BZ    CNOSEXIT
         C     RØ,=F'11'
         BNE   RETCODØ4
         CLC   RPL6RC,=XL4'ØØØØØØØ3'
         BNL   RETCODØ5
*
CNOSEXIT EQU   *
         DROP  R8                        Alloc session with CICS
         XC    FICFMHCB,FICFMHCB
         LA    R1Ø,FICFMHCB
         USING ISTFM5,R1Ø
         MVI   FM5LENTH,X'11'            FMH5 leng without pipd
         MVI   FM5FLAG1,X'Ø5'
         MVC   FM5TYPE(2),=X'Ø2FF'       Type attach
         OI    FM5FLAG2,FM5PIPPR         Pip present
         XC    FM5FXLEN,FM5FXLEN         fix len parms
         MVI   FM5LNFLP,X'Ø3'            Lparmlen
         MVI   FM5RSCTP,X'D1'            conversation mapped
         MVI   FM5LNTPN,X'Ø4'            CICS transaction name leng
         MVC   FM5TPNAM(4),TRANSNAM      CICS transaction
         MVC   FM5TPNAM+4(3),=X'ØØØØØØ' Clear subfields
         LA    R9,17(R1Ø)
         USING FM5PIPFM,R9
         LA    R8,21(,R1Ø)
         USING FM5PIPSM,R8
         MVC   FM5PIPLN,=X'ØØ6C'         1Ø8:pip fields and data len
         MVC   FM5PIPGD,=X'12F5'
         MVC   FM5PIPSL,=X'ØØ68'         1Ø4:pip data len
         MVC   FM5PIPSG,=X'12E2'
         MVC   FM5PIPSD(1ØØ),PARMS       Move parameters to pip
         DROP  R8,R9
*
ALLOCATE EQU   *
         APPCCMD CONTROL=ALLOC,                                        X
               QUALIFY=ALLOCD,                                         X
               RPL=FICRPL,                                             X
               AAREA=FICRPL6,                                          X
               ACB=FICACB,                                             X
               LOGMODE=&LOGMOD,                                        X
               OPTCD=SYN,                                              X
```

```
               CONMODE=CS,                                          X
               AREA=(R1Ø),RECLEN=125    1ØØ+4+4+17
         LA    R8,FICRPL6
         USING ISTRPL6X,R8
         LTR   R15,R15
         BNZ   RETCODØ6
         LTR   RØ,RØ
         BZ    SENDDAT
         C     RØ,=F'11'
         BNE   RETCODØ7
         CLC   RPL6RC,=F'Ø'
         BNE   RETCODØ8
*
SENDDAT  EQU   *
         MVC   CONVERID,RPL6CNVD
         APPCCMD CONTROL=SEND,                                      X
               QUALIFY=FLUSH,                                       X
               RPL=FICRPL,                                          X
               AAREA=FICRPL6,                                       X
               ACB=FICACB,                                          X
               CONVID=CONVERID,                                     X
               OPTCD=SYN
         LTR   R15,R15
         BNZ   RETCODØ9
         LTR   RØ,RØ
         BNZ   RETCOD1Ø
*
RECVDAT  EQU   *
         APPCCMD CONTROL=RECEIVE,                                   X
               QUALIFY=SPEC,                                        X
               RPL=FICRPL,                                          X
               AAREA=FICRPL6,                                       X
               ACB=FICACB,                                          X
               CONVID=CONVERID,                                     X
               AREA=CICSDATA,                                       X
               AREALEN=125,FILL=LL,                                 X
               CONMODE=CS,                                          X
               OPTCD=SYN
         LTR   R15,R15
         BNZ   RETCOD11
         LTR   RØ,RØ
         BNZ   RETCOD12
*
DEALLOC  EQU   *
         L     R9,CONVERID
         APPCCMD CONTROL=DEALLOC,                                   X
               QUALIFY=FLUSH,                                       X
               RPL=FICRPL,                                          X
               AAREA=FICRPL6,                                       X
               ACB=FICACB,                                          X
               CONVID=(R9),                                         X
```

```
               OPTCD=SYN
         LTR   R15,R15
         BNZ   RETCOD13
         LTR   RØ,RØ
         BNZ   RETCOD14
*
         LA    R4,FICCNOS1
         USING ISTSLCNS,R4
         MVC   SLCSESSL,=X'ØØØØ'
         MVC   SLCMCWL,=X'ØØØØ'
         MVC   SLCMCWP,=X'ØØØØ'
         MVC   SLCPARMS,=X'ØØ'
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=CNOS,                                           X
               RPL=FICRPL,                                             X
               AAREA=FICRPL6,                                          X
               ACB=FICACB,                                             X
               LOGMODE=&LOGMOD,                                        X
               AREA=(R4),RECLEN=7
         LA    R8,FICRPL6
         USING ISTRPL6X,R8
         LTR   R15,R15
         BNZ   RETCOD15
         LTR   RØ,RØ
         BZ    DEALLOC1
         C     RØ,=F'11'
         BNE   RETCOD16
         CLC   RPL6RC,=F'3'
         BNL   RETCOD17
*
DEALLOC1 EQU   *
         DROP  R4
         LA    R4,FICCNOS1
         USING ISTSLCNS,R4
         MVC   SLCSESSL,=X'ØØØØ'
         MVC   SLCMCWL,=X'ØØØØ'
         MVC   SLCMCWP,=X'ØØØØ'
         MVC   SLCPARMS,=X'ØØ'
*
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=CNOS,                                           X
               RPL=FICRPL,                                             X
               AAREA=FICRPL6,                                          X
               ACB=FICACB,                                             X
               LOGMODE=&LOGMOD,                                        X
               AREA=(R4),RECLEN=7
         LA    R8,FICRPL6
         USING ISTRPL6X,R8
         LTR   R15,R15
         BNZ   RETCOD18
         LTR   RØ,RØ
```

```
         BZ    DEALLOC2
         C     RØ,=F'11'
         BNE   RETCOD19
         CLC   RPL6RC,=F'3'
         BNL   RETCOD2Ø
*
DEALLOC2 EQU   *
         DROP  R4
         LA    R4,FICCNOS2
         USING ISTSLD,R4
*
DEALLOC3 EQU   *
         APPCCMD CONTROL=OPRCNTL,                                      X
               QUALIFY=DISPLAY,                                        X
               RPL=FICRPL,                                             X
               AAREA=FICRPL6,                                          X
               ACB=FICACB,                                             X
               LOGMODE=&LOGMOD,                                        X
               OPTCD=SYN,                                              X
               AREA=(R4),AREALEN=64
         LTR   R15,R15
         BNZ   RETCOD21
         LTR   RØ,RØ
         BNZ   RETCOD22
         CLC   SLDFREEC,=H'Ø'          Loop until session is freed
         BNE   DEALLOC3
         CLOSE FICACB
         CLI   RECVDATA,C' '           Answer received from CICS
         BE    EXITØ                   If space, leave, else write
         PUT   SYSPRINT,RECVDATA       to sysprint
*
EXITØ EQU   *
         CLOSE SYSPRINT
         L     R13,SAVEA+4
         LM    R14,R12,12(R13)
         SR    R15,R15
         BR    R14
*
*========  Return codes  ===========================================*
*
RETCODØ1 SHOWCB ACB=FICACB,AM=VTAM,     Get error field                X
               FIELDS=ERROR,                                           X
               LENGTH=4,                                               X
               AREA=SHWCBFLD
         MVC   PRINTLI1,=C'RETCODØ1'
         MVC   PRINTLI2,SHWCBFLD
         B     RETCOD
RETCODØ2 MVC   PRINTLI1,=C'RETCODØ2'
         MVC   PRINTLI2,RPL6RC
         B     RETCOD
RETCODØ3 MVC   PRINTLI1,=C'RETCODØ3'
```

```
         B      RETCOD
RETCODØ4 MVC    PRINTLI1,=C'RETCODØ4'
         B      RETCOD
RETCODØ5 MVC    PRINTLI1,=C'RETCODØ5'
         MVC    PRINTLI2,RPL6RC
         B      RETCOD
RETCODØ6 MVC    PRINTLI1,=C'RETCODØ6'
         B      RETCOD
RETCODØ7 MVC    PRINTLI1,=C'RETCODØ7'
         B      RETCOD
RETCODØ8 MVC    PRINTLI1,=C'RETCODØ8'
         MVC    PRINTLI2,RPL6RC
         B      RETCOD
RETCODØ9 MVC    PRINTLI1,=C'RETCODØ9'
         B      RETCOD
RETCOD1Ø MVC    PRINTLI1,=C'RETCOD1Ø'
         B      RETCOD
RETCOD11 MVC    PRINTLI1,=C'RETCOD11'
         B      RETCOD
RETCOD12 MVC    PRINTLI1,=C'RETCOD12'
         B      RETCOD
RETCOD13 MVC    PRINTLI1,=C'RETCOD13'
         B      RETCOD
RETCOD14 MVC    PRINTLI1,=C'RETCOD14'
         B      RETCOD
RETCOD15 MVC    PRINTLI1,=C'RETCOD15'
         B      RETCOD
RETCOD16 MVC    PRINTLI1,=C'RETCOD16'
         B      RETCOD
RETCOD17 MVC    PRINTLI1,=C'RETCOD17'
         MVC    PRINTLI2,RPL6RC
         B      RETCOD
RETCOD18 MVC    PRINTLI1,=C'RETCOD18'
         B      RETCOD
RETCOD19 MVC    PRINTLI1,=C'RETCOD19'
         B      RETCOD
RETCOD2Ø MVC    PRINTLI1,=C'RETCOD2Ø'
         MVC    PRINTLI2,RPL6RC
         B      RETCOD
RETCOD21 MVC    PRINTLI1,=C'RETCOD21'
         B      RETCOD
RETCOD22 MVC    PRINTLI1,=C'RETCOD22'
         B      RETCOD
RETCOD   EQU    *
         PUT    SYSPRINT,PRINTLIN
         B      EXITØ
*
*========  Work areas ==============================================*
*
         DS     ØF
SHWCBFLD DS     CL4                          VTAM error from Showcb
```

```
CONVERID DS    CL4                          Conversation ID (CICS terminal)
*
PRINTLIN DS    ØCL8Ø                        Error line
PRINTLI1 DS    CL8                          Return code text number
         DC    C' '
PRINTLI2 DC    CL4' '                       RPL6RC (Rcpri+Rcsec)
         DC    CL67' '
*
CICSDATA DS    ØCL52                        Response from CICS
         DS    CL4                          Length (4)
RECVDATA DS    CL48                         Data received (48)
*
PARMEXEC DS    ØCL8Ø                        Input parms for this program
TRANSNAM DS    CL4                          Cics transaction1 comes here
PARMS    DC    CL1ØØ' '                     Other parms go to CICS
*
SYSPRINT DCB   DSORG=PS,RECFM=F,MACRF=(PM),                            X
               LRECL=8Ø,                                               X
               DDNAME=SYSPRINT
*
SYSIN    DCB   DSORG=PS,RECFM=F,MACRF=(GM),                            X
               LRECL=8Ø,                                               X
               DDNAME=SYSIN
*
*========  VTAM appc areas  ========================================*
*
         DS    ØF
FICAPPL  DC    AL1(L'ACBNAME)
ACBNAME  DC    CL8'&ACBNAM'
FICFMHCB DS    XL255
         DS    XL1
FICCNOS1 DS    XL17
         DS    XL1
FICCNOS2 DS    XL64
FICRPL   RPL   AM=VTAM,ACB=FICACB
FICACB   ACB   AM=VTAM,MACRF=LOGON,APPLID=FICAPPL
*
FICRPL6  ISTRPL6
         IFGRPL AM=VTAM
         IFGACB AM=VTAM
         ISTFM5
         ISTSLCNS
         ISTSLD
         YREGS
         END
```

*Editor's note: this article will be concluded in the next issue.*

*(Portugal)* © Xephon 2002

# CICS news

IBM has announced Version 4.1 of its DataInterchange, which translates XML data into record-oriented files, EDI, or another form of XML, and vice versa. The software can translate directly between data in XML, EDI, or data formats. It also allows users to directly import XML DTDs into DataInterchange Client V4.1 and map them. The products affected include DataInterchange/MVS, DataInterchange/MVS-CICS, and DataInterchange Client.

DataInterchange also enables communication with trading partners, either via Information Exchange commerce engine by using Expedite software or via MQSeries messaging queues.

The software provides XML translation for any-to-any mapping, translation capability, and document type definition import. The upgraded client runs on a Windows OS and provides a GUI for DataInterchange host products.

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

* * *

Candle has announced its new OMEGAMON XE (Extended Edition) and OMEGAMON DE (Dashboard Edition) Java-based systems management tools for managing performance and availability.

The OMEGAMON XE software, sporting a new GUI, provides access to its features through a Web-enabled management portal. OMEGAMON DE provides alert and data integration and a single view of the health of enterprise infrastructure. It allows users to combine information from multiple XE monitors, as well as from third-party software.

Candle says that, starting in 2002, it will extend the OMEGAMON XE structure to its OMEGAMON II monitors for CICS, MVS, and DB2 systems.

For further information contact:
Candle, 201 N Douglas St, El Segundo, CA 90245, USA.
Tel; (310) 535 3600.
Web address: http://www.candle.com/news_events/press_releases/corporate/omegamon_xe_1022001.html.

* * *

IBM has announced VSE/ESA Version 2 Release 6, which offers expanded capabilities to create integrated solutions in a hybrid environment.

The new release adds TCP/IP support for CICS External Call Interface (ECI), VSE connectors have been updated to the Java 2 standard, Internet security is improved with the addition of SSL to TCP/IP, and SSL exploitation includes CICS Web Support (CWS).

Version 2.6 also supports zSeries 900 (31-bit mode only), FICON, 2074, VSAM support for large 3390-9 disks, support for OSA Express, and FastCopy exploitation of the ESS FlashCopy feature.

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