211

CICS update

June 2003

In this issue

3 Designing RCT entries

4 Implications of certain CICS log manager messages

13 CICS Transaction Gateway Version 5 and CICS Transaction Server Version 2.2: Part 3 – Visual Basic code

36 QMF goes on-the-fly to Adobe Acrobat Reader, MS Word, MS Excel, …

49 CICS questions and answers

51 CICS news

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

Disclaimer

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

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.

Designing RCT entries

When we first installed our DB2 RCT entries for CICS we used a very simple concept – we just put as many transactions as possible in one entry. As we were preparing to transfer from CICS 4.1 to CICS TS 1.3, we realized that it would be necessary to completely redesign our RCT concept, otherwise no-one would know where to find the corresponding information for a transaction and all its components. Many of the RCT parameters had not been used before, so the development of a new concept involved a lot of trial and error. As the development went along, shortcomings appeared that made a restart of the development necessary – one of the biggest problems was to have meaningful names, which is quite hard to do when you have a name length of only eight characters. On the third try I understood a lot more, and developed a concept for our company. I will share that concept here.

It was important to develop a concept that would prove easy to maintain and that each system programmer could easily identify with (and find in the CSD). There are those RCT entries that apply to the database itself, and should be grouped together so that, if anything is changed on the database, they will probably not be forgotten. Then there are the entries that apply to the transactions – they should be in the transaction groups so that they are not forgotten or overlooked.

We have a CSD group called RCTTn (where Tn is a test CICS with n being the test CICS number). This RCTTn group contains the entries applicable to the DB2 database. Here is where DB2CONN and DB2ENTRYs are defined. The DB2ENTRYs are named after the transaction group they apply to. This allows us now to get DB2 statistics that show how much a transaction group is being used, and we can tune our DB2 accordingly. After the group name we have a short form of the AUTHID and PLAN. Let me give an example:

```
…ROM  Read Only AUTHID, Master PLAN
…RWRM Read Write AUTHID, Remote Master Plan
```

The '…' stands for the transaction group, like SYS (system), PER (personnel), or ADR (address). The DB2TRAN entries are defined in the corresponding RDO transaction group (for our example ADR) and provide the connection between the transactions and the DB2ENTRYs. The DB2TRAN name is created using the RDO group name (ADR) and the transaction name (INF2), which gives a DB2TRAN of (ADRINF2). This entry refers to the ADRROM DB2ENTRY.

Generic references would be called ADRINF; the DB2TRAN would contain INF+ as a generic transaction, and could reference ADRRWRM. It should be noted that specific transaction names are checked first and then the generic DB2TRAN entries – in other words the more specific definitions will always be taken first.

Maybe this little article can help you when you are designing your RCT for CICS TS and save you some of that trial-and-error time.

William Kim Mongan
Systems Programmer (Germany)

Implications of certain CICS log manager messages

INTRODUCTION

It is important to be aware of the implications for CICS log management if certain messages are seen during CICS activity keypoint and recovery processing. This article describes the background to these and explains the potential issues that may be associated with them. *Please note that CICS is a trademark of IBM Corporation in the United States and other countries.*

CICS LOG MANAGER IMPLEMENTATION

The CICS Log Manager domain is itself based on the exploitation of the MVS System Logger subsystem. This is a component of z/OS, running as a separate subsystem called IXGLOGR. It provides an application programming interface for other jobs, such as CICS, to write, read, browse, and delete data to and from the various logs under its control. The MVS System Logger manages its log data in sequential logstreams. A logstream is a series of blocks of data. Each logstream is identified by its own unique logstream identifier. This is the logstream name. The CICS Log Manager handles various logstreams for CICS's own use; these include the CICS system log. Other logstreams are available for user purposes, such as holding audit trail information.

The CICS system log is used to record all records required to provide dynamic transaction backout of an abending Unit Of Work (UOW) – for example, when a user task abends having modified a recoverable VSAM file. In addition, CICS system log data is used for recovering a CICS system to a committed state when performing an emergency restart. The CICS system log therefore records 'before-images' of changes to resources managed under CICS. Managed by the CICS Log Manager domain, and therefore shielded from the rest of CICS, the CICS system log is physically implemented by two separate logstreams. The primary logstream is also referred to as journal name DFHLOG, and the secondary logstream is also known as journal name DFHSHUNT. The CICS Log Manager is responsible for handling the movement and manipulation of UOW log data on these two CICS system logstreams.

ACTIVITY KEYPOINTING AND LOG TRIMMING

CICS uses CSKP tasks attached to perform activity keypoint operations as an opportunity for trimming the system logstreams. This means they issue IXGDELET calls to the MVS System Logger subsystem to delete unwanted data on the DFHLOG and DFHSHUNT system logs. In the time between successive CSKP tasks running in a CICS system, it is likely that a fair number of

user tasks will have completed their processing and ended. Once their UOWs have successfully committed any recoverable changes by means of a syncpoint, the associated log data for these changes is no longer required by CICS for use in a subsequent backout operation. This data (held on the system logs) is therefore redundant. CICS will therefore analyse the log data for each inflight UOW at the time of a keypoint, and determine the position on the logs where the oldest piece of relevant information is held. This will reflect the earliest recoverable operation made by any of the inflight UOWs. CICS can then call the MVS System Logger to delete logstream data up to this point. In this way, successive keypoints within CICS are used to trim the logstreams and ensure that they do not contain unwanted data.

Typically, DFHLOG would expect to be trimmed fairly often, and most keypoint operations would therefore be expected to find that the oldest point of interest held on this logstream has moved forwards in time between successive keypoint tasks. There are exceptions to this rule of thumb, however – a task may be 'long-running', which means it is performing recoverable operations across a period of time that spans several activity keypoints. If these recoverable changes are not delimited by periodic syncpoints, then the task will contain one long-lived UOW and hence have recorded log data required for backout purposes over this long time period. In real terms, of course, a 'long time period' may be quite a short actual period of time (say a minute), but one which is long in the CICS sense because it has the potential to span a number of activity keypoint operations. Once such a task's log data becomes the oldest point of interest on the logstream, as other tasks complete and leave the system, it will prevent subsequent trimming of the logstream by CICS activity keypoints. This situation will persist until the long-running task completes, or else issues a syncpoint command to terminate its current UOW. Only by doing this will its log data become redundant to CICS, and so eligible for deletion at a future keypoint. The alternative outcome is that for some reason the task fails to write to the system log in the time between two successive activity keypoint operations, at which point CICS will

automatically move its log data to the secondary system logstream DFHSHUNT. (This could be as the result of a conversational task waiting for terminal input, for example.) DFHSHUNT is used by CICS to store log data for UOWs that are not actively logging, such as those tasks described above, or those which fail indoubt within a syncpoint operation and are shunted by CICS until the cause of the indoubt failure can be resolved and their syncpoint resolution achieved.

As a general rule, therefore, it is normal to see DFHLOG being trimmed by CICS on activity keypoints, since it is likely that most tasks logging to it are short-lived and will complete and so leave the system in the time between activity keypoints. Conversely, it is rarer to see DFHSHUNT be trimmed by an activity keypoint task, since this holds log data for longer-lived tasks, and by definition these will not terminate so quickly as shorter-lived ones.

In order to help recognize potential problems with the system logstreams, CICS will issue informational messages to the CSMT TD queue during its activity keypoints, to record the state of its management of DFHLOG and DFHSHUNT. These messages are an important diagnostic tool when investigating problems with the CICS system log logstreams, such as if they are seen to be growing in size unexpectedly.

An edited example of these informational messages is given below:

```
DFHRM0205 An activity keypoint has been successfully taken.

DFHLG0743  Tail of logstream TEST.ANDYCICS.DFHLOG deleted at block id
X'0000000013D97D38C'.
```

This example shows that a successful trim of DFHLOG occurred during the activity keypoint. Since DFHSHUNT was not eligible for trimming at the time that the CSKP transaction ran, CICS did not request that data be deleted from it, and no corresponding DFHLG0743 message for the secondary logstream was seen. This is not untypical, and not in itself a concern. What is more serious is when DFHRM0205 messages are issued (indicating

that CICS has performed an activity keypoint) and yet no corresponding DFHLG0743 messages for DFHLOG are issued too. The absence of these messages may go unnoticed, and over a period of time it may be the case that a considerable number of activity keypoints have been unable to trim the primary system logstream.

DFHLG0760 MESSAGES

Failure to trim DFHLOG means that CICS is allowing the volume of log data in this logstream to grow. Unless CICS issues the appropriate IXGDELET calls to the MVS System Logger to trim the logstreams, the data held there will be retained indefinitely. This is because the logstream definition for DFHLOG and DFHSHUNT should use default values of AUTODELETE(NO) and RETPD(0).

The MVS System Logger subsystem supports these two optional parameters for use in the automation of logstream data management. They were introduced in OS/390 Version 1 Release 3. Prior to their introduction, data written to a logstream would remain there indefinitely. In order for it to be removed, a user of the MVS System Logger (such as CICS) would have to issue an explicit IXGDELET call to mark the data for deletion. Such an IXGDELET call would not result in physical deletion of the log data. Instead, the MVS System Logger would regard the section of log data as being logically deleted. Physical deletion (and freeing up of the associated logstream storage) would occur asynchronously to this, at the next point in time that the MVS System Logger subsystem performed its appropriate housekeeping activities. Typically, this would be at the next offload processing event.

The introduction of RETPD and AUTODELETE helped to simplify log data management by providing a means of automating some of the housekeeping functions of the logstream. The RETPD option defines a retention period (in days) for log data before it is physically deletable from a logstream. This means that (regardless of whether an IXGDELET call has been issued or

not) log data will not be physically deletable by the MVS System Logger until the retention period associated with that logstream has been reached. Once the retention period has elapsed, normal MVS System Logger housekeeping activity can physically delete any logically deleted log data on the logstream.

The use of RETPD is coupled with the AUTODELETE option. AUTODELETE (YES or NO) specifies whether the MVS System Logger will perform an automatic physical deletion of log data**,** or whether an explicit IXGDELET macro call is required to perform the delete. In other words, AUTODELETE(YES) takes control of when deletion of log data can occur away from other jobs using the logstream, and gives this role to the MVS System Logger.

Since only CICS knows whether or not its system log data is still required it would be inappropriate to allow the use of RETPD and AUTODELETE to automate deletion of logstream data based on these criteria. Hence the default values of AUTODELETE(NO) and RETPD(0) should always be specified for DFHLOG and DFHSHUNT's logstreams; the follow-on from this is that if CICS fails to ask for log data to be trimmed from the log at activity keypoint time, the MVS System Logger will not automatically delete the data at a later date. Since this is the case, a long-running task can lead to DFHLOG not being trimmed for some considerable time, and hence the logstream's primary storage can fill and result in log data being offloaded to secondary logstream storage by the MVS System Logger.

It should be noted that, for general logs such as audit trails, the use of a non-zero retention period and/or autodeletion may be appropriate in certain cases. CICS does not manage the deletion of data on such logstreams, and hence how long the data is retained is purely a site-specific consideration.

In order to help identify situations where DFHLOG was not being trimmed correctly, CICS has been enhanced to issue a new message if this does not occur during an activity keypoint. This is message DFHLG0760. Each successive keypoint that is unable to trim DFHLOG will issue a DFHLG0760 with a count value stating how many keypoints have elapsed since DFHLOG

was able to be trimmed. An edited example of such CSMT output is given below:

```
DFHRMØ2Ø5 An activity keypoint has been successfully taken.

DFHLGØ76Ø  Log stream TEST.ANDYCICS.DFHLOG not trimmed by keypoint
processing. Number of keypoints since last trim occurred: 3,973.
DFHRMØ2Ø5 An activity keypoint has been successfully taken.

DFHLGØ76Ø  Log stream TEST.ANDYCICS.DFHLOG not trimmed by keypoint
processing. Number of keypoints since last trim occurred: 3,974.
```

This example shows an extreme case where almost 4000 (decimal) activity keypoints have taken place and none has been able to delete log data from DFHLOG. Clearly this signifies a problem to be investigated. It may be that a particularly long-running 'batch-style' program is executing that is performing a great many recoverable operations against a resource such as a recoverable VSAM file. In such a case, it may be appropriate to review the use of EXEC CICS SYNCPOINTs within such a program, to break down its large UOW size into several smaller ones and so allow for more periodic trimming of DFHLOG. Alternatively (as in this example case), it may signify a program that was invalidly looping on the system, repeatedly performing a recoverable operation such as an EXEC CICS READ UPDATE/ EXEC CICS REWRITE of a given record on a recoverable file. Such a yielding loop would prevent CICS from abending the task since control was repeatedly being given back to CICS from the application, signifying that it was performing valid work rather than (say) an unyielding loop purely within application logic itself. In either case, analysis of the tasks in the system, and their UOWs, should be performed with CEMT I TASK and CEMT I UOW to investigate the nature of the problem more closely.

Note: while DFHLG0760 is provided to identify problems with tasks preventing trimming of DFHLOG, there is no corresponding DFHLG0760 message for when an activity keypoint does not trim DFHSHUNT. This is because DFHSHUNT is expected to be trimmed far more rarely than DFHLOG, and so this is the normal case (rather than the exception to it).

DFHLG0760 message support was added to CICS TS 1.3 by

APAR PQ34528 (PTF UQ44303): it is present at base code level in CICS TS 2.2.

CICS LOG MANAGER MESSAGES ISSUED DURING SYSTEM RESTART

During CICS emergency restart processing, the CICS Log Manager Domain scans backwards through the DFHLOG system logstream (and, possibly, DFHSHUNT too) to read back the data needed to rebuild the inflight UOWs that were present when the previous run of CICS was terminated. This in turn allows CICS to back out these rebuilt UOWs and so restore the CICS system's recoverable resources to a committed state once more. Such a backwards scan of the system log would normally be expected to complete quite quickly. However, if the inflight UOWs present at the time CICS had terminated were long-running transactions (or batch-style programs), they may have updated many thousands (or more) of recoverable resources in the same Unit Of Work. While not regarded as good CICS application programming practice, this is not invalid under CICS.

If such UOWs were present at the time of the emergency restart, the time spent scanning backwards through the system log during emergency restart would take much longer to complete. CICS would be very busy during the period, with the region utilizing CPU as it repeatedly drove MVS System Logger code to read back the UOW's log records.

CICS documents this activity by issuing a series of console messages as it performs the backwards scan of the system log. Message DFHLG0745 shows when the sequential backwards scan of the system log begins. DFHLG0747 messages are issued periodically to indicate how many records have been processed so far. These messages are issued after a certain number of records have been read back from the system log, where this number is the greater of either 500 or half the CICS AKPFREQ value. During the scan, if a DFHLG0748 message is issued it indicates that CICS is able to optimize the backwards scanning process. Finally, at the end of the backwards log scan, CICS issues message DFHLG0749.

It would be unusual to see many DFHLG0747 messages during an emergency restart, since relatively few log records would need to be read back from a typical system log to allow CICS to rebuild all the relevant UOW information needed for the recovery of the system. However, long-running tasks (as described above) can lead to these messages being seen. When this is the case, the restart time can be impacted by the time taken to retrieve all the log records in order to allow CICS to back out the changes made by the task. It is also likely that any such UOW(s) leading to such an extended emergency restart would have revealed their presence by causing DFHLG0760 messages to have been issued during activity keypoints prior to the termination of the previous run of CICS.

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

Why not share your expertise and earn money at the same time? *CICS Update* is looking for macros, program code, etc, that experienced CICS users have written to make their life, or the lives of their users, easier. We will publish it (after vetting by our expert panel) and send you a cheque when the article is published. Articles can be of any length and can be sent or e-mailed to Trevor Eddolls at any of the addresses shown on page 2. A free copy of our *Notes for contributors* is available from our Web site at www.xephon.com/nfc.

CICS Transaction Gateway Version 5 and CICS Transaction Server Version 2.2: Part 3 – Visual Basic code

In the third of a series of articles on the CICS Transaction Gateway Version 5 (CTG), I present some Visual Basic code that can be used within an Excel spreadsheet. This Visual Basic code provides an ECI linkage to CICS Transaction Server Version 2.2 (CICS) from within the Excel spreadsheet.

The first in this series of articles described my view of how to arrange a CICS Transaction Gateway residing on a workstation (or, generally, on a non-MVS platform) to communicate using TCP/IP to a production CICS region – *CICS Transaction Gateway Version 5 and CICS Transaction Server Version 2.2: Part 1 – Architecture for connecting a workstation CICS Transaction Gateway to CICS*, April 2003, issue 209.

The second in the series considered my view of arrangements when the CICS Transaction Gateway was residing in MVS (ie OS/390 or z/OS) in communication with a production CICS region in the same LPAR – *CICS Transaction Gateway Version 5 and CICS Transaction Server Version 2.2: Part 2 – Architecture for connecting an OS/390 or z/OS CICS Transaction Gateway to CICS*, May 2003, issue 210.

Naturally, I'm going to ignore all the advice given in these previous articles about not exposing CTG communication definitions to workstation applications! I'm going to assume a test environment whereby the CTG is communicating directly with an AOR. I'm doing this so that you have an overview of what needs to be done. Consequently, I'm not going to use the CTG exit.

Before describing the code, this article describes how to configure the CTG and CICS for communication.

Both the CTG and CICS need to be configured for a successful ECI flow. The CTG has to reside on the same workstation as the Excel spreadsheet.

Configuring the CTG for communication

The workstation CTG needs to be configured to connect to CICS TS 2.2. You should define a TCP/IP connection as shown below:

```
SECTION SERVER = IYCKRAH6
    DESCRIPTION=RAH 2.2
    UPPERCASESECURITY=Y
    USENPI=N
    PROTOCOL=TCPIP
    NETNAME=winmvs2c.hursley.ibm.com
    PORT=1Ø123
    CONNECTTIMEOUT=Ø
    TCPKEEPALIVE=N
ENDSECTION
```

The port number must match that defined in an installed TCPIPSERVICE in the CICS region.The NETNAME is the TCP/IP address of the MVS image which is running CICS. The relevant machine is WINMVS2C at Hursley in my case.

The server parameter is externally visible and has to be quoted in the supplied code – it does not relate to the APPLID used by the CICS region.

If you are following my advice given in the previous two articles, this will be the only CTG definition required. The CTG exit should fix this definition name (so allowing the workstation application to be configuration-independent in a production environment).

Configuring CICS for communication

You must create and install a TCPIPSERVICE definition for use by the CICS Transaction Gateway. The TCP/IP port number is that defined in the CTG configuration and the protocol must be ECI. The mirror transaction invoked for CTG usage is CIEP. The TCPIPSERVICE definition is shown below:


```
 TCpipservice   : CTG
 GROup          : RAHCTG
 DEscription    : TCPIP FOR CTG
 Urm            :
 POrtnumber     : 10123
 STatus         : Open
 PRotocol       : Eci
 TRansaction    : CIEP
 Backlog        : 00001
 TSqprefix      :
 Ipaddress      :
 SOcketclose    : No
SECURITY
 SSl            : No
 Certificate    :
 AUthenticate   :
 ATtachsec      : Verify
DNS CONNECTION BALANCING
 DNsgroup       :
 GRPcritical    : No
```

If you are using multiple CRRs (CTG Routing Regions) as discussed previously you will have multiple regions with the same TCPIPSERVICE definition active (all listening on the same TCP/IP port).

DEFINING THE CICS ENVIRONMENT

Defining the linked-to program in CICS

My code uses the EC01 sample supplied as part of the CTG. Copy this to MVS, compile the COBOL module, and place it in a library within DFHRPL. Below is the associated RDO definition:

```
PROGram        : EC01
 Group          : RAHCTG
 DEscription    :
 Language       : CObol
 RELoad         : No
 RESident       : No
 USAge          : Normal
 USElpacopy     : No
 Status         : Enabled
 RSl            : 00
 CEdf           : Yes
 DAtalocation   : Below
 EXECKey        : User
```

```
 COncurrency    : Quasirent
REMOTE ATTRIBUTES
 DYnamic        : No
 REMOTESystem   :
 Transid        :
 EXECUtionset   : Fullapi
JVM ATTRIBUTES
 JVM            : No
 JVMClass       :
 JVMProfile     : DFHJVMPR
JAVA PROGRAM OBJECT ATTRIBUTES
 Hotpool        : No
```

This program definition would not be present in a CRR, only the relevant AORs. The Dynamic Routing Program will decide on the relevant AOR.

However, if the CRR is using static routing, a program definition is required in the CRR which has the AOR set in the REMOTESystem field.

Creating the conversion table

The EC01 program returns a COMMAREA containing the date and time. This COMMAREA will have been created in the codepage of the CICS AOR region. Data has to be converted into the codepage used by the Excel Spreadsheet. Conversion is done by coding a DFHCNV table within CICS. DFHCNV itself is RDOed via the DFHISC group.

```
TITLE 'RAH DFHCNV table '
 DFHCNV TYPE=INITIAL
 DFHCNV TYPE=ENTRY,
       RTYPE=PC,
       CLINTCP=85Ø,
       SRVERCP=Ø37,
       RNAME=ECØ1,
       USREXIT=NO
 DFHCNV TYPE=SELECT,
       OPTION=DEFAULT
 DFHCNV TYPE=FIELD,
       OFFSET=Ø,
       DATATYP=CHARACTER,
       DATALEN=32767,
       LAST=YES
 DFHCNV TYPE=FINAL
 END   DFHCNVBA
```

(Contination character in column 72 is omitted.)

The example above shows that my CICS region is running with codepage 850 and the workstation uses 037. You should change these appropriately for your environment.

This is compiled as an Assembler table into a library in DFHRPL.

The TYPE=ENTRY names the program and sets the involved codepages. Observe that the whole of the COMMAREA is converted (TYPE=FIELD). This exemplifies an important point about using a workstation CTG to run a CICS application: things work best if only character data (A-Z, a-z, 0-9, and a space) is used in the COMMAREAs.

You can place this definition into the AORs as well as the CRRs, but it will be used only in the CRR (because this is where the information leaves the CICS environment).

CODE AND THE EXCEL SPREADSHEET

Adding the Visual Basic code to the Excel spreadsheet

Activate the Visual Basic Editor (**Tools/Macro/VisualBasic Editor** or ALT+F11).

In the editor use **File/Import** to insert the downloaded Visual Basic code and then close this Editor window.

In the spreadsheet window, **Tools/Macro/Macros** (or ALT+F8) will now show the three public subroutines of callctg, callctgcell, and CTGCBarAdd.

Running the callctg routine

When callctg is used, all required definitions are contained within the routine. Consequently, you can just run callctg as a macro.

Running the callctgcell routine

The control information for callctgcell is a string placed in a cell.

Select the cell containing the command string and run the callctgcell macro.

The string consists of Key=Value pairs, with comma delimiters.

Internal routines

The callctg and callctgcell routines are wrappers that attempt to check settings before calling the private procedure doctg, which does the flow to CICS and determines the returned information.

Other internal routines are used to check that a cell name actually refers to a cell (CTGIsCell) and to update a cell (CTGUpdateCell).

The author does not really consider himself to be a Visual Basic programmer, so you will undoubtedly find the code in these routines rather crude.

The code is merely an example (and unsupported by IBM) of how to use the CICS Transaction Gateway, and you can freely change it for your own use.

USING THE CODE WITHIN AN EXCEL SPREADSHEET

The Visual Basic routines permit access to the CTG where all values are encoded in cells (the callctg routine) or where values are specified in a single cell (callctgcell). The CTGCBarAdd routine will add a command bar to the spreadsheet containing buttons to run these routines.

You may need the CTG programming documentation to hand when setting the various parameters.

Setting the userid and password

The userid and password are those used by CICS to run the program. These values must be known to the External Security Manager (such as RACF) used by the CICS region. The settings are forced into uppercase:

- Using callctg – update the cUserid and cPassword Dims to

point to the cell containing these values. They default to B1 and B2 and are read-only .

- Using callctgcell – the values must be contained in the selected cell used to contain settings. They must be specified as USERID= and PASSWORD=.

Setting the server

The server is the name of the CICS region to contact as defined in a SECTION SERVER=XXXXXX setting in the CTG configuration file. The name is forced into uppercase:

- Using callctg – update the cServer Dim to point to the cell containing the name of the server as known to the CTG. It defaults to B3 and is read-only .
- Using callctgcell – the value must be contained in the selected cell used to contain settings. It must be specified as SERVER=.

If you are using a CRR, this CTG configuration name will address a definition naming the CRR. However, when the CTG exit is being used, this field would not normally be supplied.

Setting the program to run in CICS

The program to run in CICS (via the equivalent of an EXEC CICS LINK) must be supplied. The program must have been RDOed to CICS and be available for use. The setting is forced to uppercase:

- Using callctg – update the cProgram Dim to point to the cell containing the program to run. It defaults to B4 and is read-only.
- Using callctgcell – the value must be contained in the selected cell used to contain settings. It must be specified as PROGRAM=.

Setting the CICS transactions

The mirror task and the transaction id used by CICS to run the

program can be specified if the defaults are not satisfactory. These transaction-ids must be defined to CICS and be available for use. The settings are forced to uppercase:

- Using callctg – update the cMirror and cAttach Dims to point to the cell containing these values. They default to B5 and B6 and are read-only.
- Using callctgcell – the values must be contained in the selected cell used to contain settings. They must be specified as MIRROR= and ATTACH=.

Setting the timeout for the communication

The timeout can be used to fail a CTG operation if CICS does not respond in a reasonable time. The setting is be checked for a numeric value:

- Using callctg – update the cTimeout Dim to point to the cell containing this value. It defaults to B7 and is read-only.
- Using callctgcell – the value must be contained in the selected cell used to contain settings. It must be specified as TIMEOUT=.

Supplying COMMAREAs for the program

The COMMAREA supplied to the program is specified as the contents of a cell. The cell is not updated with the returned COMMAREA, instead this is placed into another cell.

The COMMAREA will be converted to/from the codepage used by CICS according to the DFHCNV table residing in the CICS region contacted by the CTG (a CRR).

The output COMMAREA cell is cleared before a flow is made to CICS via the CTG. However, if an error occurs before then, the output COMMAREA cell may not be cleared. Similarly, an error occurring after the CTG has been invoked could prevent the updating of the output COMMAREA cell from taking place. Thus, it is recommended that the output COMMAREA cell be emptied before invoking the CTG flow.

The values supplied for COMMAREAs are names/identities of cells: these are checked to ensure that they actually refer to cells:

- Using callctg – update the cInCOMMAREA and cOutCOMMAREA Dims to name cells. The Input COMMAREA cell is read-only and the output COMMAREA cell is write-only. The input COMMAREA cell defaults to B8 (read-only) and the output COMMAREA cell (write-only) to B9.
- Using callctgcell – the cell names must be contained in the selected cell used to contain settings. They must be specified as INCOMMAREA= and OUTCOMMAREA=. The input COMMAREA cell is read-only and the output COMMAREA cell is write-only.

Getting the return code, abend code, and error message

The potential abend code, a possible message, and a return code are returned by the Visual Basic code. Each of these items is returned in a cell (write-only) specified for the utility. The abend code and message are also part of the return code string, in addition to their own cells.

The abend code is set to '****' if no abend code was returned by CICS.

The return code consists of a string in three parts, with spaces delimiting the components:

- The first word is the numeric return code returned by the CTG and will be set to -1 if an abend code was returned by CICS.
- The second word is the abend code returned by CICS (which will be **** if no abend code was returned).
- Subsequent words will comprise an error message (if available).

The first word of the return code cell (up to the first space) should be checked after usage of the utility to determine whether or not data in the output COMMAREA cell is valid:

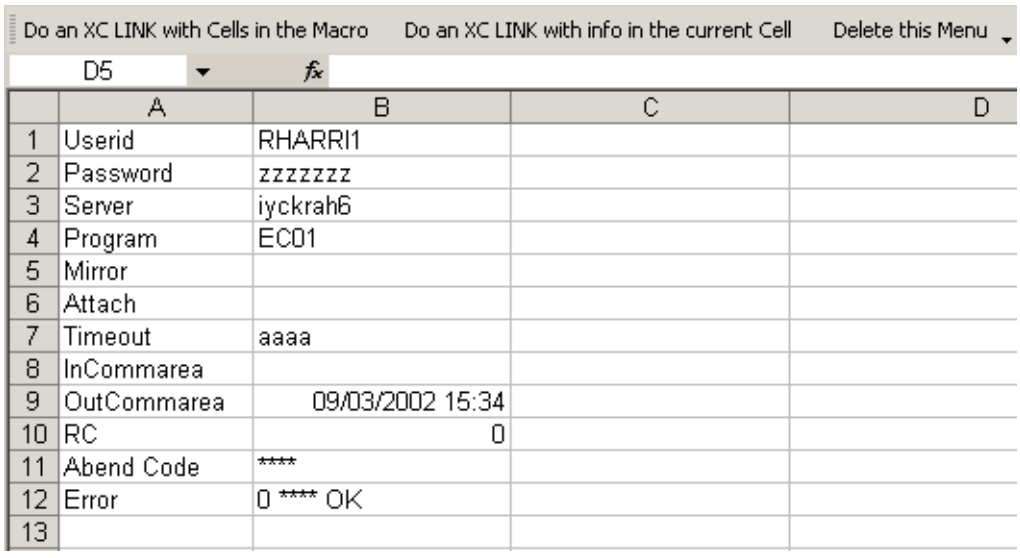
- Using callctg – update the cRC, cAbend and cError Dims to name cells into which the values will be placed. cRC defaults to B10, cAbend to B11 and cError to B12. The cells themselves are write-only.
- Using callctgcell – the cell names must be contained in the selected cell used to contain settings. They must be specified as RC=, ABEND= and ERROR=. The cells themselves are write-only.

The return code, message, and abend cells are cleared before a flow to CICS via the CTG, but errors may cause them not to be updated.

Running the code

The macro CTGCBarAdd adds a command bar to the spreadsheet, which contains buttons to execute the routines.

If you are using callctg (with items defined in the macro) then the relevant button just invokes the macro. It does not need any parameters as these are encoded in the subroutine/macro. If you do not use the command bar, just run the callctg macro.



	A	B	C	D
1	Userid	RHARRI1		
2	Password	zzzzzzz		
3	Server	iyckrah6		
4	Program	EC01		
5	Mirror			
6	Attach			
7	Timeout	aaaa		
8	InCommarea			
9	OutCommarea	09/03/2002 15:34		
10	RC	0		
11	Abend Code	****		
12	Error	0 **** OK		
13				

Figure 1: Example results using callctg

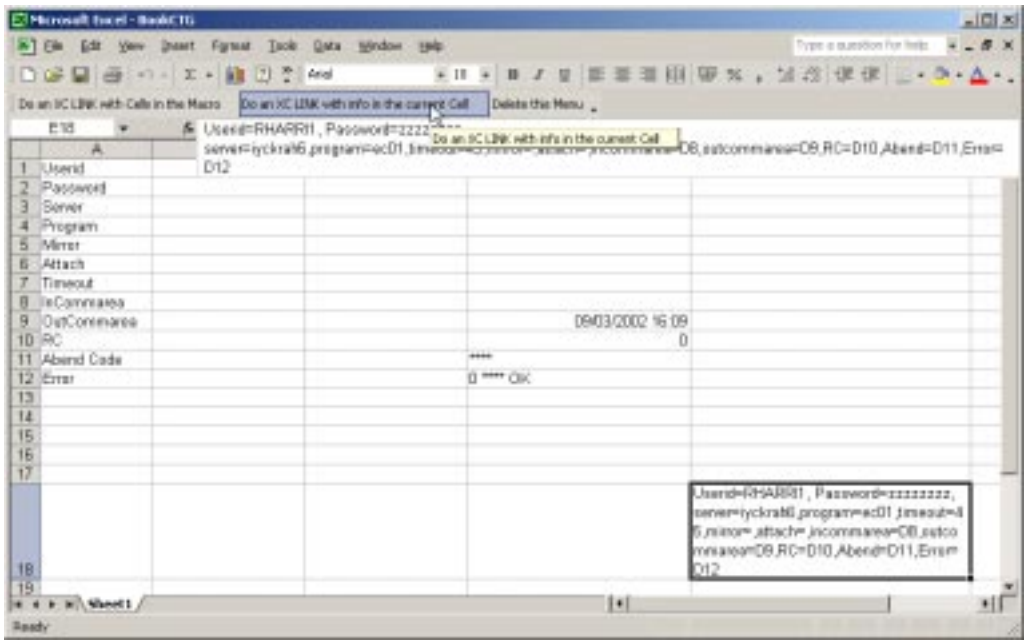
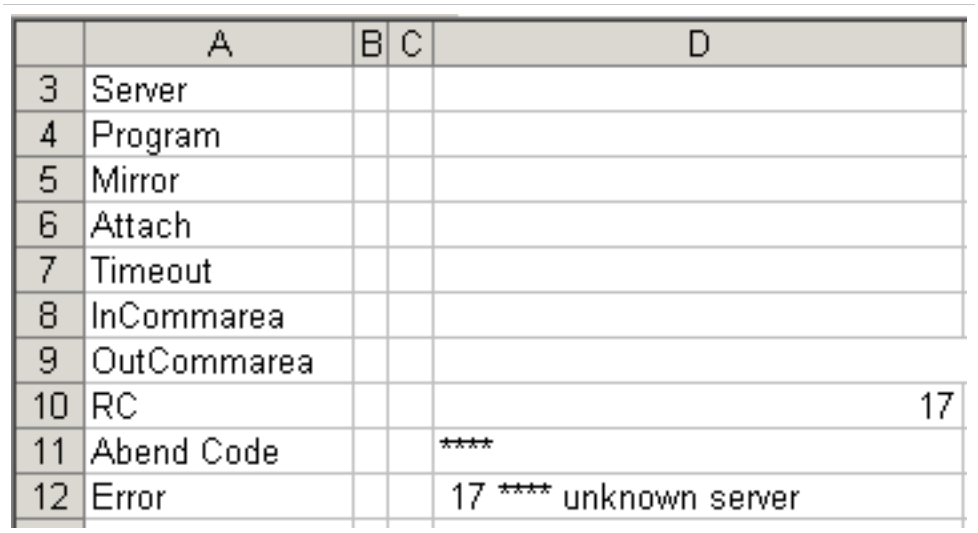


Figure 2: Callctgcall example

To use callctgcell, select the cell containing the command string and press the relevant button. Alternatively, run the callctgcell

	A	B	C	D
1	Userid			
2	Password			
3	Server			
4	Program			
5	Mirror			
6	Attach			
7	Timeout			
8	InCommarea			
9	OutCommarea			
10	RC			18
11	Abend Code			****
12	Error			18 **** security error

Figure 3: Security failure



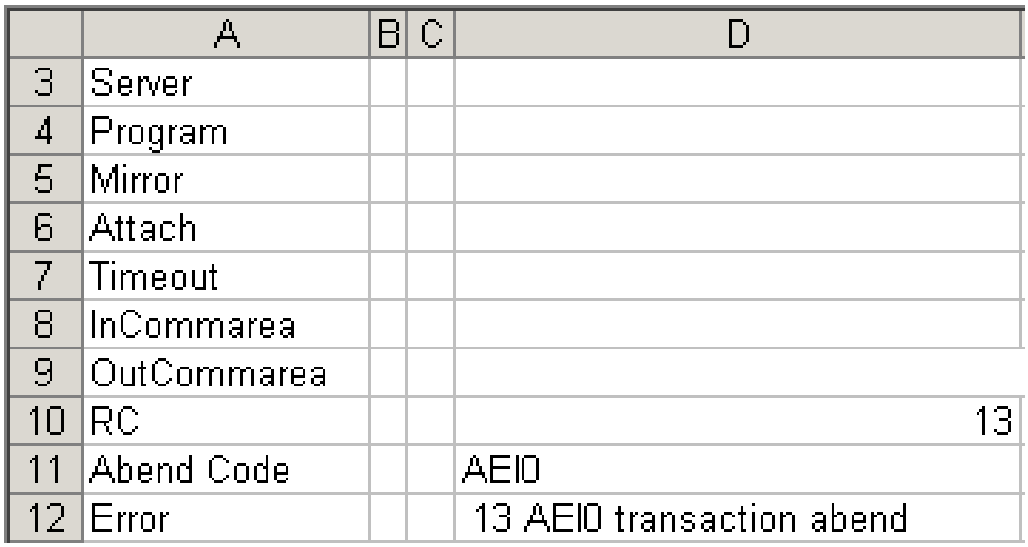
	A	B	C	D
3	Server			
4	Program			
5	Mirror			
6	Attach			
7	Timeout			
8	InCommarea			
9	OutCommarea			
10	RC			17
11	Abend Code			****
12	Error			17 **** unknown server

Figure 4: Server failure

macro with the command string cell selected.

Example returns

After the macro/routine has executed, the output cells should



	A	B	C	D
3	Server			
4	Program			
5	Mirror			
6	Attach			
7	Timeout			
8	InCommarea			
9	OutCommarea			
10	RC			13
11	Abend Code			AEI0
12	Error			13 AEI0 transaction abend

Figure 5: Unknown program

contain output sent from CICS.

The cells defined within callctg are shown below:

```
' Define the default Cells for the CTG usage
' Update this block with the relevant Cells
' Don't update anything before this comment

Dim cUserid As String: cUserid = "B1"
Dim cPassword As String: cPassword = "B2"
Dim cServer As String: cServer = "B3"
Dim cProgram As String: cProgram = "B4"
Dim cMirror As String: cMirror = "B5"
Dim cAttach As String: cAttach = "B6"
Dim cTimeout As String: cTimeout = "B7"
Dim cInCommarea As String: cInCommarea = "B8"
Dim cOutCommarea As String: cOutCommarea = "B9"
Dim cRC As String: cRC = "B1Ø"
Dim cAbend As String: cAbend = "B11"
Dim cError As String: cError = "B12"

' Don't update anything after this comment
```

Figure 1 is the result of running this macro. Figure 2 shows how the equivalent function is executed via callctgcell.

Figure 3 shows what happens if the userid or password is incorrect, and Figure 4 the error generated if the server key/ identity is not found in the CTG definition file.

An incorrectly specified program (or one that is not known to CICS) returns an abend, as shown in Figure 5.

These figures show a specific server being used. If you are following my advice from the previous two articles, this field should be left unspecified because the CTG exit will fix the desired value. Indeed, only the userid, password, program, and input COMMAREA would generally be specified, the CTG exit supplying any other needed definitions.

CONCLUSION

This series of articles has shown how to arrange connections between a CICS Transaction Gateway Version 5 and a production CICS Transaction Server Version 2.2 to permit maximum availabililty and speediest performance.

In a non-MVS environment, I have shown how to hide this configuration from applications and use TCP/IP commmunication securely to CICS.

In the MVS environment, I have shown how different architectural options can be used to obtain the best combination of performance and availability.

Finally, some Visual Basic Code demonstrated how CICS-sourced information can be incorporated into an Excel spreadsheet.

So now it is up to you. Follow the guidelines in these articles and you will find that external access to CICS function is reliable, secure, and easy to maintain!

CODE

```
Attribute VB_Name = "Module1"
' Legal Notice
'
' This program is not supported in any way by IBM. It is distributed
' only to show techniques for CICS Transaction Server and CICS
' Transaction Gateway usage. It may be freely modified and adapted by
' your installation.
'
' (C) Copyright IBM UK Labs Limited, 2003.
'
Sub callctg()
' Define the default Cells for the CTG usage
' Update this block with the relevant Cells
' Don't update anything before this comment
Dim cUserid As String: cUserid = "B1"
Dim cPassword As String: cPassword = "B2"
Dim cServer As String: cServer = "B3"
Dim cProgram As String: cProgram = "B4"
Dim cMirror As String: cMirror = "B5"
Dim cAttach As String: cAttach = "B6"
Dim cTimeout As String: cTimeout = "B7"
Dim cInCommarea As String: cInCommarea = "B8"
Dim cOutCommarea As String: cOutCommarea = "B9"
Dim cRC As String: cRC = "B10"
Dim cAbend As String: cAbend = "B11"
Dim cError As String: cError = "B12"
' Don't update anything after this comment
' Define locals for the parms
```

```
Dim aUserid As String * 16: aUserid = " "
Dim aPassword As String * 16: aPassword = " "
Dim aServer As String * 8: aServer = " "
Dim aProgram As String * 8: aProgram = " "
Dim aMirror As String * 4: aMirror = " "
Dim aAttach As String * 4: aAttach = " "
Dim aTimeout As Integer: aTimeout = Ø
Dim aInCommarea As String * 32500: aInCommarea = ""
Dim aOutCommarea As String * 32500: aOutCommarea = ""
Dim aRC As Integer: aRC = Ø
Dim aError As String: aError = ""
Dim aAbend As String * 4: aAbend = "    "
' Check that all the Cells are accessible
If (CTGIsCell(cUserid, "Userid", True) = False) Then Exit Sub
If (CTGIsCell(cPassword, "Password", True) = False) Then Exit Sub
If (CTGIsCell(cServer, "Server", True) = False) Then Exit Sub
If (CTGIsCell(cProgram, "Program", True) = False) Then Exit Sub
If (CTGIsCell(cMirror, "Mirror Task", True) = False) Then Exit Sub
If (CTGIsCell(cAttach, "Attach Transaction", True) = False) Then Exit
Sub
If (CTGIsCell(cTimeout, "Timeout Commarea", True) = False) Then Exit Sub
If (CTGIsCell(cInCommarea, "Input Commarea", True) = False) Then Exit
Sub
If (CTGIsCell(cOutCommarea, "Output Commarea", True) = False) Then Exit
Sub
If (CTGIsCell(cRC, "returned Return Code", True) = False) Then Exit Sub
If (CTGIsCell(cAbend, "returned Abend Code", True) = False) Then Exit
Sub
If (CTGIsCell(cError, "returned Error String", True) = False) Then Exit
Sub
' Read data from the Cells
If ((Application.Range(cUserid).Value <> "") _
     And (Len(Application.Range(cUserid).Value) > Ø) _
     And (Left(Application.Range(cUserid).Value, 1) <> " ")) Then
     aUserid = Application.Range(cUserid).Value
     End If
If ((Application.Range(cPassword).Value <> "") _
     And (Len(Application.Range(cPassword).Value) > Ø) _
     And (Left(Application.Range(cPassword).Value, 1) <> " ")) Then
     aPassword = Application.Range(cPassword).Value
     End If
If ((Application.Range(cServer).Value <> "") _
     And (Len(Application.Range(cServer).Value) > Ø) _
     And (Left(Application.Range(cServer).Value, 1) <> " ")) Then
     aServer = Application.Range(cServer).Value
     End If
If ((Application.Range(cProgram).Value <> "") _
     And (Len(Application.Range(cProgram).Value) > Ø) _
     And (Left(Application.Range(cProgram).Value, 1) <> " ")) Then
     aProgram = Application.Range(cProgram).Value
```

```
     End If
If ((Application.Range(cMirror).Value <> "") _
     And (Len(Application.Range(cMirror).Value) > Ø) _
     And (Left(Application.Range(cMirror).Value, 1) <> " ")) Then
     aMirror = Application.Range(cMirror).Value
     End If
If ((Application.Range(cAttach).Value <> "") _
     And (Len(Application.Range(cAttach).Value) > Ø) _
     And (Left(Application.Range(cAttach).Value, 1) <> " ")) Then
     aAttach = Application.Range(cAttach).Value
     End If
If ((Application.Range(cTimeout).Value <> "") _
     And (Len(Application.Range(cTimeout).Value) > Ø) _
     And (Left(Application.Range(cTimeout).Value, 1) <> " ") _
     And (IsNumeric(Application.Range(cTimeout).Value) = True)) Then
     aTimeout = Application.Range(cTimeout).Value
     End If
' Set a Zero RC and Clear the Error Field
If (CTGUpdateCell(cRC, "Ø", "ReturnCode", True) = False) Then Exit Sub
If (CTGUpdateCell(cError, "", "Error Text", True) = False) Then Exit Sub
' Get the input commarea and clear the output one
If ((Len(Application.Range(cInCommarea).Value) <> Ø)) Then
    aInCommarea = Application.Range(cInCommarea).Value
    End If
aOutCommarea = ""
' Call the CTG routine
Call doctg(aUserid, aPassword, aServer, aProgram, _
           aMirror, aAttach, aTimeout, _
           aInCommarea, aOutCommarea, _
           aRC, aAbend, aError)
' Return the commare to the requested cell
If (CTGUpdateCell(cOutCommarea, aOutCommarea, "Output Commarea", True) =
False) Then Exit Sub
' Return Return Code and possible Error Message
If (CTGUpdateCell(cRC, Str(aRC), "ReturnCode", True) = False) Then Exit
Sub
If (CTGUpdateCell(cError, aError, "Error Text", True) = False) Then Exit
Sub
If (CTGUpdateCell(cAbend, aAbend, "AbendCode", True) = False) Then Exit
Sub
End Sub
Sub callctgcell()
' UpdateCell returned
Dim tf As Boolean: tf = True
' Contents of the current Cell
Dim cInfo As String: cInfo = ""
' Current cell split up by , delimiters
Const maxcArray = 15
Dim cArray() As String
' Define Obtained Info & Cell settings
```

```
Dim cUserid As String * 16: cUserid = " "
Dim cPassword As String * 16: cPassword = " "
Dim cServer As String * 8: cServer = " "
Dim cProgram As String * 8: cProgram = " "
Dim cMirror As String * 4: cMirror = " "
Dim cAttach As String * 4: cAttach = " "
Dim cTimeout As String: cTimeout = " "
' Define locals for the parms
Dim aUserid As String * 16: aUserid = " "
Dim aPassword As String * 16: aPassword = " "
Dim aServer As String * 8: aServer = " "
Dim aProgram As String * 8: aProgram = " "
Dim aMirror As String * 4: aMirror = " "
Dim aAttach As String * 4: aAttach = " "
Dim aTimeout As Integer: aTimeout = Ø
Dim aInCommarea As String * 32500: aInCommarea = ""
Dim aOutCommarea As String * 32500: aOutCommarea = ""
Dim aRC As Integer: aRC = Ø
Dim aError As String: aError = ""
Dim aAbend As String * 4: aAbend = "    "
' Get the Current Cell Info and split into comma delitimed fields
cInfo = Application.ActiveCell.Value
cArray = Split(cInfo, ",", maxcArray, vbTextCompare)
' now loop through the array, saving values in variables
' and checking returned cells for existance
For i = Ø To UBound(cArray)
    j = Ø
    k = ""
    v = ""
    j = InStr(cArray(i), "=")
    If (j <> Ø) Then
        k = Mid(cArray(i), 1, (j - 1))
        v = Mid(cArray(i), (j + 1))
        k = Trim(UCase(k))
        v = Trim(UCase(v))
        Select Case k
           Case "USERID"
                cUserid = v
           Case "PASSWORD"
                cPassword = v
           Case "SERVER"
                cServer = v
           Case "PROGRAM"
                cProgram = v
           Case "MIRROR"
                cMirror = v
           Case "ATTACH"
                cAttach = v
           Case "INCOMMAREA"
                cInCommarea = v
```

```
                If (CTGIsCell(v, "input Commarea", True) = False) _
                   Then Exit Sub
           Case "OUTCOMMAREA"
                cOutCommarea = v
                If (CTGIsCell(v, "returned Commarea", True) = False) _
                   Then Exit Sub
           Case "RC"
                cRC = v
                If (CTGIsCell(v, "returned ReturnCode", True) = False) _
                   Then Exit Sub
           Case "ABEND"
                cAbend = v
                If (CTGIsCell(v, "returned AbendCode", True) = False) _
                   Then Exit Sub
           Case "ERROR"
                cError = v
                If (CTGIsCell(v, "returned Error String", True) = False)
_
                   Then Exit Sub
           Case Else
           End Select
        End If
    Next
' Check and set the Fixed Text info
' Cannot check the existance of Returned Cell areas
If ((Len(cUserid) > Ø) _
     And (Left(cUserid, 1) <> " ")) Then
     aUserid = cUserid
     End If
If ((Len(cPassword) > Ø) _
     And (Left(cPassword, 1) <> " ")) Then
     aPassword = cPassword
     End If
If ((Len(cServer) > Ø) _
     And (Left(cServer, 1) <> " ")) Then
     aServer = cServer
     End If
If ((Len(cProgram) > Ø) _
     And (Left(cProgram, 1) <> " ")) Then
     aProgram = cProgram
     End If
If ((Len(cMirror) > Ø) _
     And (Left(cMirror, 1) <> " ")) Then
     aMirror = cMirror
    End If
If ((Len(cAttach) > Ø) _
     And (Left(cAttach, 1) <> " ")) Then
     aAttach = cAttach
     End If
If ((Len(cTimeout) > Ø) _
```

```
     And (Left(cTimeout, 1) <> " ") _
     And (IsNumeric(cTimeout) = True)) Then
     aTimeout = cTimeout
     End If
' Set a Zero RC and Clear the Error Field
If (CTGUpdateCell(cRC, "Ø", "ReturnCode", True) = False) Then Exit Sub
If (CTGUpdateCell(cError, "", "Error Text", True) = False) Then Exit Sub
' Get the input commarea and clear the output one
If ((Len(Application.Range(cInCommarea).Value) <> Ø)) Then
    aInCommarea = Application.Range(cInCommarea).Value
    End If
aOutCommarea = ""
' Call the CTG routine
Call doctg(aUserid, aPassword, aServer, aProgram, _
           aMirror, aAttach, aTimeout, _
           aInCommarea, aOutCommarea, _
           aRC, aAbend, aError)
' Return the commare to the requested cell
If (CTGUpdateCell(cOutCommarea, aOutCommarea, "Output Commarea", True) =
False) Then Exit Sub
' Return Return Code and possible Error Message
If (CTGUpdateCell(cRC, Str(aRC), "ReturnCode", True) = False) Then Exit
Sub
If (CTGUpdateCell(cError, aError, "Error Text", True) = False) Then Exit
Sub
If (CTGUpdateCell(cAbend, aAbend, "AbendCode", True) = False) Then Exit
Sub
End Sub
Private Sub doctg(ByVal pUserid As String, _
          ByVal pPassword As String, _
          ByVal pServer As String, _
          ByVal pProgram As String, _
          ByVal pMirror As String, _
          ByVal pAttach As String, _
          ByVal pTimeout As Integer, _
          ByRef pInCommarea As String, _
          ByRef pOutCommarea As String, _
          ByRef pRC As Integer, _
          ByRef pAbend As String, _
          ByRef pError As String)
On Error Resume Next
' Define locals for the parms
Dim Userid As String * 16: Userid = " "
Dim Password As String * 16: Password = " "
Dim Server As String * 8: Server = " "
Dim Program As String * 8: Program = " "
Dim Mirror As String * 4: Mirror = " "
Dim Attach As String * 4: Attach = " "
Dim Timeout As Integer: Timeout = Ø
Dim RC As Integer: RC = Ø
```

```
Dim RCError As String: RCError = ""
' Define some locals for error processing
Dim Abend As String * 4: Abend = "   "
Dim State As String: State = " "
Dim Runrc As Integer: Runrc = Ø
Dim runerror As CclECIExceptionCodes: runerror = cclNoError
Dim runerrormsg As String: runerrormsg = ""
Dim runerror_old As Integer: runerror_old = Ø
Dim runerrormsg_old As String: runerrormsg_old = ""
Dim runerror_new As Integer: runerror_new = Ø
Dim runerrormsg_new As String: runerrormsg_new = ""
' Copy the parms doing case conversions
Userid = UCase(pUserid)
Password = UCase(pPassword)
Server = UCase(pServer)
Program = UCase(pProgram)
Mirror = UCase(pMirror)
Attach = UCase(pAttach)
Timeout = pTimeout
' Create the CTG objects
Dim ECI As CclOECI: Set ECI = New CclOECI
Dim Connection As CclOConn: Set Connection = New CclOConn
Dim Flow As CclOFlow: Set Flow = New CclOFlow
Dim Commarea As CclOBuf: Set Commarea = New CclOBuf
' Clear Output parms
pRC = Ø
pAbend = "    "
pError = ""
' Clear the output commarea
pOutCommarea = ""
' Set Commarea Length to be the max and copy the input area
Commarea.SetString pInCommarea
Commarea.SetLength 32500
' Populate the ECI Objects
' UOW only exists on CICS (XC LINK SYNCONRETURN used)
ECI.SetErrorFormat (1)
Connection.Details Server, Userid, Password
Connection.TranDetails aAttach, aMirror
Flow.SetSyncType cclSync
Flow.SetTimeout Timeout
' Do an XC LINK to CICS without any UOW
Connection.Link Flow, Program, Commarea, Nothing
' Get hold of any Abend Codes etc.
Abend = Flow.AbendCode
State = Flow.Diagnose
Runrc = Err.Number
If ((Runrc <> Ø) And ((Runrc And 65535) >= ECI.ErrorOffset)) Then
     runerror_new = (Runrc And 65535) - ECI.ErrorOffset
     runerrormsg_new = Err.Description
     End If
```

```
runerror_old = ECI.ExCode
runerrormsg_old = ECI.ExCodeText
' Decide between the old and new error settings!
If (runerror_new <> Ø) Then
    runerror = (runerror_new And 65535) - ECI.ErrorOffset
    runerrormsg = runerrormsg_new
    Else
     If (runerror_old <> Ø) Then
        runerror = runerror_old
        runerrormsg = runerrormsg_old
        Else
        runerror = Ø
        runerrormsg = ""
        End If
    End If
' Format up the Return Code String
RCError = ""
If (runerror = Ø) Then
    If (Left(Abend, 1) <> " ") Then
        RCError = "-1 "
        Else
        RCError = "Ø "
        End If
    Else
    RCError = Str(runerror) + " "
    End If
If (Left(Abend, 1) = " ") Then
    RCError = RCError + "**** "
    Else
    RCError = RCError + UCase(Left(Abend, 4)) + " "
    End If
If ((Len(runerrormsg) = Ø) Or (Left(runerrormsg, 1) = " ")) Then
    RCError = RCError + "OK "
    Else
      If ((Len(runerrormsg) <> Ø) Or (Left(runerrormsg, 1) <> " ")) Then
      RCError = RCError + runerrormsg + " "
      Else
      RCError = RCError + runerrormsg_old
      End If
    End If
' Return the RC and possible Error/Abends
pRC = runerror
pError = RCError
If (Left(Abend, 1) = " ") Then
    pAbend = "****"
    Else
    pAbend = UCase(Left(Abend, 4))
    End If
' Return the Commrea
pOutCommarea = Commarea.String
```

```
End Sub
Sub CTGCBarAdd()
' This routine will create a new CTG Command Bar
Const CBName = "CTG"
' Remove an old version
Call CTGMenuDelete
' Create the CTG Command Bar
Dim nb1 As CommandBar
Set nb1 = Application.CommandBars.Add
nb1.Position = msoBarTop
nb1.Visible = True
nb1.Name = "CTG"
' Creat Entries in the Bar
Dim cb1 As CommandBarButton
Set cb1 = nb1.Controls.Add(Type:=msoControlButton)
cb1.Caption = "Do an XC LINK with Cells in the Macro    "
cb1.Visible = True
cb1.Style = msoButtonCaption
cb1.OnAction = "callctg"
Dim cb2 As CommandBarButton
Set cb2 = nb1.Controls.Add(Type:=msoControlButton)
cb2.Caption = "Do an XC LINK with info in the current Cell     "
cb2.Visible = True
cb2.Style = msoButtonCaption
cb2.OnAction = "callctgcell"
Dim cb3 As CommandBarButton
Set cb3 = nb1.Controls.Add(Type:=msoControlButton)
cb3.Caption = "Delete this Menu"
cb3.Visible = True
cb3.Style = msoButtonCaption
cb3.OnAction = "CTGMenuDelete"
End Sub
Private Sub CTGMenuDelete()
Const CBName = "CTG"
' Remove an old version
For Each cb In Application.CommandBars
    If (cb.Name = CBName) Then
        Application.CommandBars(CBName).Delete
        Exit For
        End If
     Next
End Sub
Private Function CTGUpdateCell(ByVal Cell As String, _
                               ByRef Data As String, _
                               ByVal Because As String, _
                               ByVal DoBox As Boolean) _
                 As Boolean
' This Function will update the given cell with the
' supplied data. It will return TRUE if this worked or
' FALSE if this operation raised an error.
```

```
'
' The Data is passed by Ref for efficiency purposes
'
' If the DoBox parm is TRUE, then a msgbox will
' be sent if there is an error (presumably the
' Cell parm is not actually a valid cell id)
Dim UcRC As Boolean: UcRC = True
On Error GoTo UCERROR
Application.Range(Cell).Value = Data
' This next statement will get executed whether
' or not there is an error: but the error routine
' will have changed UcRc
CTGUpdateCell = UcRC
Exit Function
' The Error handling routine
UCERROR:
   UcRC = False
   If (DoBox = True) Then
       i = MsgBox("Attempt to update Cell " + Cell + _
                     " to return the " + Because + _
                     " failed. Is this really a Cell? ", _
                  vbOKOnly + vbExclamation, _
                  "CTG Cell update error" _
                  )
       End If
      Resume Next
End Function
Private Function CTGIsCell(ByVal Cell As String, _
                           ByVal Because As String, _
                           ByVal DoBox As Boolean) _
                 As Boolean
' This Function will check the existence of the given cell
' It will return TRUE if the cell exists (by reading it) or
' FALSE if this operation raised an error.
'
' If the DoBox parm is TRUE, then a msgbox will
' be sent if there is an error (presumably the
' Cell parm is not actually a valid cell id)
Dim UcRC As Boolean: UcRC = True
Dim Data As String: Data = ""
On Error GoTo UCERROR
Data = Application.Range(Cell).Value
' This next statement will get executed whether
' or not there is an error: but the error routine
' will have changed UcRc
CTGIsCell = UcRC
Exit Function
' The Error handling routine
UCERROR:
   UcRC = False
```

```
   If (DoBox = True) Then
       i = MsgBox("An attempt to access " + Cell + _
                    " for the " + Because + _
                    " failed. Is this really a Cell? ", _
                  vbOKOnly + vbExclamation, _
                  "CTG Cell access error" _
                  )
       End If
      Resume Next
End Function
```

Robert Harris
CICS Technical Strategist
IBM Hursley (UK)

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

INTRODUCTION

In my previous article (*QMF goes to the Web*, issue 205, December 2002), I explained how it was possible to reuse existing QMF reports and put them on the Web. An HTML page is a powerful way to show data, but when you print it the result is sometimes disappointing, eg page breaks may occur in the middle of a page. Another annoying thing is when you want to import the HTML page into a spreadsheet – the conversion may give some surprising results.

For those reasons, it's time now for QMF to go directly from the mainframe to Acrobat Reader, MS Word, or MS Excel.

THE PRINCIPLE

As shown in Figure 1, when a user types a URL (1), the HTTP request handler starts a transaction (2) running the QMF callable interface. This transaction reads the parameters from the request,

initializes the QMF session, executes a procedure (3), and prints the result in a temporary storage (TS) queue (4). The program then reads the TS queue (5) and, if a PDF document is

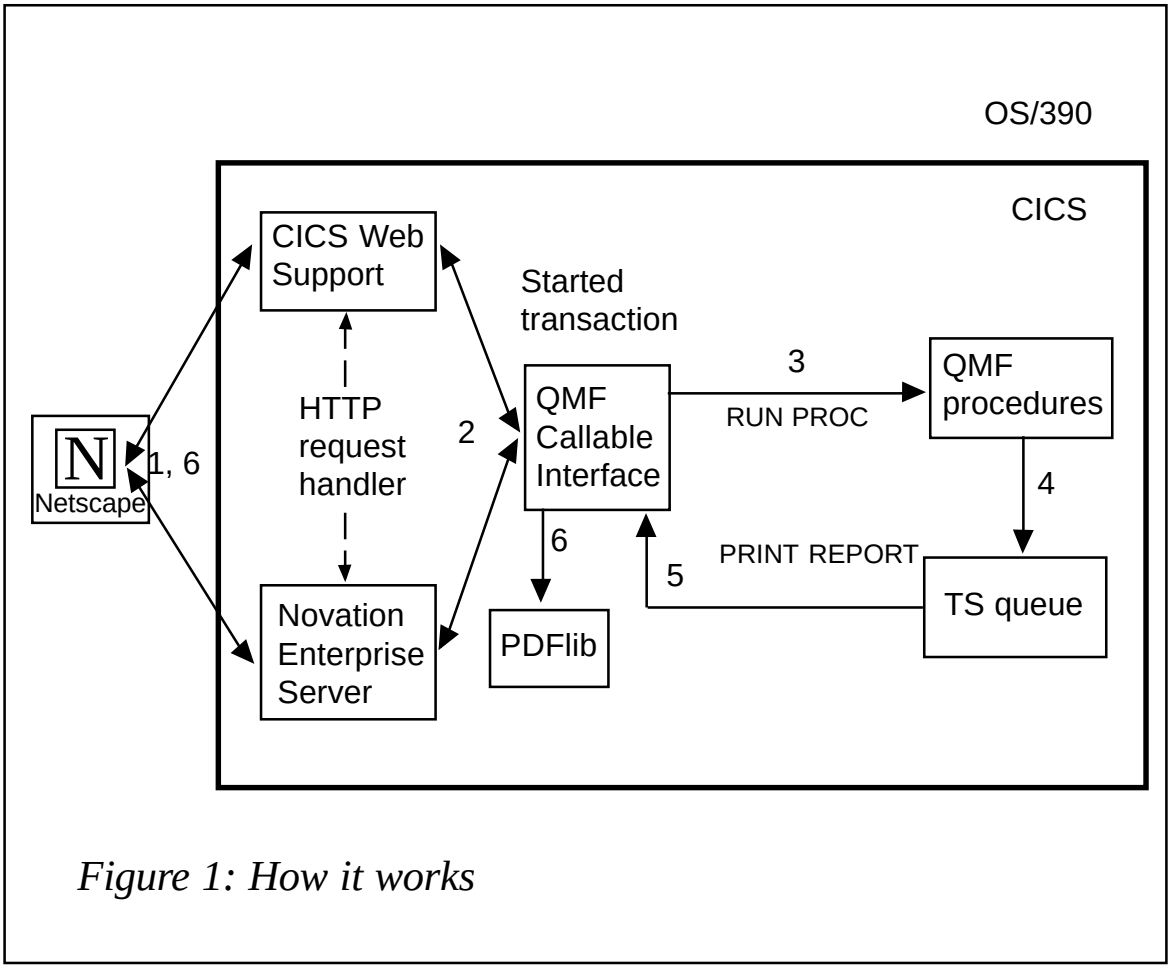


Figure 1: How it works

requested, calls the PDFLIB routines to build it (6), builds the HTTP header (see below), and puts the result in the DFHCOMMAREA. The HTTP request handler sends it back to the user (7).

This differs from my previous article in:

- The HTTP header.
- The PDFlib routines. Those routines generate PDF documents either in MVS batch, in Unix System Services, or in memory.

PDFlib is a development tool for PDF-enabling your software, or

generating PDFs on your server. PDFlib saves you the intricate details of PDF generation by offering a simple-to-use API for programmatically creating PDF files from within your own server or client-side software.

PDFlib doesn't make use of Adobe Acrobat for generating PDFs, nor does it require any other third-party software. The PDFlib core is written in the ANSI C language. A dedicated COBOL language wrapper for PDFlib on MVS is planned.

PDFlib offers an easy-to-use-programming interface for the application programmer. The PDFlib API shields the programmer from the technicalities of PDF generation. Any programmer with decent graphics or print output experience is able to use PDFlib quickly. The PDFlib reference manual explains the basics of PDFlib programming, and provides a detailed reference to all API functions. Sample programs are provided for all supported environments.

Fully-functional evaluation versions of PDFlib including documentation and programming examples are available from http://www.pdflib.com.

If you have a good knowledge of C/C++, you can also surf to http://www.fastio.com to download CLIBPDF routines. The only problem is that those routines have not been ported to MVS and you have to compile and link-edit the load modules yourself.

If you want to generate MS Word or MS Excel documents, you don't need PDFlib routines. The guidelines for creating those files are explained below.

THE HTTP HEADER

When you request a static Web page, the Web server inserts, before sending the HTML, an HTTP header to tell the browser what type of file it will receive. Although you never see it, the Web browser uses it to start the corresponding viewer, including itself for the HTML pages. For example, the header:

```
HTTP/1.Ø 2ØØ OKLF
Content-Length: 2526LF
```

```
Content-Type: application/PDFLFLF
```

will start the Adobe Acrobat reader. LF is the character corresponding to the line feed. In EBCDIC, its hexadecimal value is X'15'. 2526 is the length of the HTML page not including the header. The array below shows some typical HTTP headers and their corresponding viewers and file extensions:

HTTP/1.0 200 OKLF
Content-Length: 1045LF
Content-Type: text/htmlLFLF – Web browser – Html, htm

HTTP/1.0 200 OKLF
Content-Length: 2526LF
Content-Type: application/rtfLFLF – MS Word – rtf

HTTP/1.0 200 OKLF
Content-Length: 3228LF
Content-Type: text/comma-separated-valuesLFLF – MS Excel – csv

When you request an HTML page, you don't have to look at this, the Web server does it for you. But, for other file types, most of the time, you will have to set it. Please refer to Novation Enterprise documentation or CICS Web support guides.

More information on HTTP headers is available at http://www.w3.org/Protocols/HTTP/Object_Headers.html.

SAMPLE PROGRAM

You will find below a sample that:

- Executes the QMF report.
- Prints it or exports it in temporary storage queue (TSQ).
- Generates the requested document from the TSQ.
- Builds the HTTP header if needed.
- Deletes the TSQ.

****

```
*   EXECUTION OF QMF REPORTS WITHIN A CGI COBOL PROGRAM
*   ================================================
*
*   PROGRAM   : QMFWEB2
*   AUTHOR    : DELAUNOY P.
*   DATE      : Ø8/2ØØ2
*   VERSION   : 2.Ø
*
*   FUNCTIONS :
*   ----------
*
*   1) EXECUTES A QMF REPORT USING QMF CALLABLE INTERFACE.
*   2) PRINT OR EXPORT THE REPORT TO A TEMPORARY STORAGE
*      QUEUE (TSQ).
*   3) DEPENDING ON THE FILE FORMAT SPECIFIED IN THE URL,
*      BUILD THE CORRESPONDING DOCUMENT. IEE IF THE URL IS LIKE
*      HTTP://MYHOST.MYDOMAIN:PORT/CGI-BIN/QMFWEB2.CSV
*      THE CGI WILL GENERATE A CSV DOCUMENT (EXCEL FILE).
*   4) IF THE REQUESTED DOCUMENT IS A PDF FILE, THE PROGRAM WILL
*      CALL THE C++ QMFPDF ROUTINE WICH WILL BUILD THE PDF. IN
*      THIS CASE, THE FILE IS BINARY. THAT'S WHY, WE
*      MUST BUILD AN HTTP HEADER. PLEASE REFER TO NOVATION
*      ENTERPRISE SERVER DOCUMENTATION OR CICS CWS GUIDES.
*   5) WHEN THE FILE AND EVENTUALLY THE HTTP HEADER IS BUILT,
*      PUT IT IN THE COMMAREA.
*   6) DELETE THE QMF REPORT TSQ
*
*   HISTORY :
*   --------
*****************************************************************
 IDENTIFICATION DIVISION.
 PROGRAM-ID.  QMFWEB2.
 ENVIRONMENT DIVISION.
 DATA DIVISION.
 WORKING-STORAGE SECTION.
*************************************************************
* COBOL INCLUDE FOR QUERY CALLABLE INTERFACE (MVS/VM)
*************************************************************
* STRUCTURE DECLARE FOR COMMUNICATIONS AREA
  Ø1 DSQCOMM.
     Ø3 DSQ-RETURN-CODE     PIC 9(8) USAGE IS COMP.
*                              FUNCTION RETURN CODE       *
     Ø3 DSQ-INSTANCE-ID     PIC 9(8) USAGE IS COMP.
*                              IDENTIFIER FROM START CMD  *
     Ø3 DSQ-COMM-LEVEL      PIC X(12).
*                              COMMUNICATIONS LEVEL       *
     Ø3 DSQ-PRODUCT         PIC X(2).
*                              QUERY PRODUCT ID           *
     Ø3 DSQ-PRODUCT-RELEASE PIC X(2).
*                              QUERY PRODUCT RELEASE      *
```

```
     03 DSQ-RESERVE1        PIC X(28).
*                                RESERVED AREA              *
     03 DSQ-MESSAGE-ID      PIC X(8).
*                                COMPLETION MESSAGE ID      *
     03 DSQ-Q-MESSAGE-ID    PIC X(8).
*                                QUERY MESSAGE ID           *
     03 DSQ-START-PARM-ERROR PIC X(8).
*                                START PARAMETER IN ERROR   *
     03 DSQ-CANCEL-IND      PIC X(1).
*                                1 = COMMAND CANCELLED      *
*                                0 = COMMAND NOT CANCELLED  *
     03 DSQ-RESERVE2        PIC X(23).
*                                RESERVED AREA              *
     03 DSQ-RESERVE3        PIC X(156).
*                                RESERVED AREA              *
     03 DSQ-MESSAGE-TEXT    PIC X(128).
*                                QMF MESSAGE TEXT           *
     03 DSQ-Q-MESSAGE-TEXT  PIC X(128).
*                                QMF QUERY MESSAGE TEXT     *
*                                512 BYTES TOTAL            *
*  VALUES FOR DSQ-RETURN-CODE
  01 DSQ-SUCCESS            PIC 9(8) USAGE IS COMP VALUE 0.
  01 DSQ-WARNING            PIC 9(8) USAGE IS COMP VALUE 4.
  01 DSQ-FAILURE            PIC 9(8) USAGE IS COMP VALUE 8.
  01 DSQ-SEVERE             PIC 9(8) USAGE IS COMP VALUE 16.
*  VALUES FOR DSQ-INSTANCE-ID
  01 DSQ-CONTINUE           PIC 9(8) USAGE IS COMP VALUE 0.
*  VALUES FOR DSQ-COMM-LEVEL
  01 DSQ-CURRENT-COMM-LEVEL PIC X(12) VALUE 'DSQL>001002<'.
*  VALUES FOR DSQ-PRODUCT
  01 DSQ-QRW                PIC X(2) VALUE '01'.
  01 DSQ-QMF                PIC X(2) VALUE '02'.
  01 DSQ-QM4                PIC X(2) VALUE '03'.
*  VALUES FOR DSQ-PRODUCT-RELEASE
  01 DSQ-QRW-V1R2           PIC X(2) VALUE '01'.
  01 DSQ-QRW-V1R3           PIC X(2) VALUE '02'.
  01 DSQ-QMF-V2R4           PIC X(2) VALUE '01'.
  01 DSQ-QMF-V3R1           PIC X(2) VALUE '02'.
  01 DSQ-QM4-V1R1           PIC X(2) VALUE '01'.
*  VALUES FOR DSQ-CANCEL-INDE
  01 DSQ-CANCEL-YES         PIC X(1) VALUE '1'.
  01 DSQ-CANCEL-NO          PIC X(1) VALUE '0'.
*  VALUES FOR MODE
  01 DSQ-INTERACTIVE        PIC X(1) VALUE '1'.
  01 DSQ-BATCH              PIC X(1) VALUE '2'.
*  VALUES YES AND NO
  01 DSQ-YES                PIC X(1) VALUE '1'.
  01 DSQ-NO                 PIC X(1) VALUE '2'.
*  CALLABLE INTERFACE PROGRAM NAME
  01 DSQCIB                 PIC X(6) VALUE 'DSQCIB'.
```

```
*  VALUES FOR VARIABLE TYPE ON CALL PARAMETER
  Ø1 DSQ-VARIABLE-CHAR      PIC X(4) VALUE 'CHAR'.
  Ø1 DSQ-VARIABLE-FINT      PIC X(4) VALUE 'FINT'.
* QUERY INTERFACE COMMANDS
 Ø1  STARTQI                PIC X(5)  VALUE 'START'.
 Ø1  SETG                   PIC X(1Ø) VALUE 'SET GLOBAL'.
 Ø1  PROC.
     Ø2  FILLER             PIC X(9)  VALUE 'RUN PROC '.
     Ø2  PROCNAM            PIC X(26) VALUE ' '.
 Ø1  EXPORT.
     Ø2  FILLER             PIC X(17) VALUE 'EXPORT REPORT TO '.
     Ø2  USERID-EXPORT      PIC X(Ø8) VALUE SPACES.
     Ø2  FILLER             PIC X(1Ø) VALUE '(QUEUET=TS'.
     Ø2  FILLER             PIC X(17) VALUE ' C=NO S=NO D=HTML'.
 Ø1  PRINT.
     Ø2  FILLER             PIC X(21)
                               VALUE 'PRINT REPORT (QUEUEN='.
     Ø2  USERID-PRINT       PIC X(Ø8) VALUE SPACES.
     Ø2  FILLER             PIC X(17)
                               VALUE 'QUEUET=TS PR='' '' '.
     Ø2  FILLER             PIC X(18) VALUE 'DA=NO S=NO PAGE=NO'.
 Ø1  ENDQI                  PIC X(4)  VALUE 'EXIT'.
* QUERY COMMAND LENGTH
 Ø1  QICLTH                 PIC 9(8) USAGE IS COMP-4.
* NUMBER OF VARIABLES
 Ø1  QIPNUM                 PIC 9(8) USAGE IS COMP-4.
* KEYWORD VARIABLE LENGTHS
 Ø1  QIKLTHS.
   Ø3   KLTHS               PIC 9(8) OCCURS 1Ø USAGE IS COMP-4.
* VALUE LENGTHS
 Ø1  QIVLTHS.
   Ø3   VLTHS               PIC 9(8) OCCURS 1Ø USAGE IS COMP-4.
* START COMMAND KEYWORD
 Ø1  SNAMES.
   Ø3  SNAME1               PIC X(8) VALUE 'DSQSMODE'.
   Ø3  SNAME2               PIC X(8) VALUE 'DSQSBSTG'.
   Ø3  SNAME3               PIC X(8) VALUE 'DSQSPILL'.
   Ø3  SNAME4               PIC X(8) VALUE 'DSQSIROW'.
   Ø3  SNAME5               PIC X(8) VALUE 'DSQALANG'.
   Ø3  SNAME6               PIC X(8) VALUE 'DSQSSPQN'.
* START COMMAND KEYWORD VALUE
 Ø1  SVALUES.
   Ø3  SVALUE1              PIC X(1) VALUE 'B'.
*  KB ALLOCATED FOR USER 1ØØØØ=DEFAULT
   Ø3  SVALUE2              PIC X(6) VALUE '1ØØØØ'.
   Ø3  SVALUE3              PIC X(3) VALUE 'YES'.
*  NUMBER OF ROWS 1ØØØ=DEFAULT
   Ø3  SVALUE4              PIC X(4) VALUE '1ØØØ'.
*  LANGUAGE ENGLISH
   Ø3  SVALUE5              PIC X(1) VALUE 'E'.
```

```
*  QMF INTERNAL STORAGE QUEUE NAME
   Ø3  SVALUE6              PIC X(8) VALUE 'QMFSTORA'.
* SET GLOBAL COMMAND VARIABLE NAMES TO SET
 Ø1  VNAMES.
   Ø3  VNAME1               PIC X(18).
 Ø1  NNAMES.
   Ø3  NNAME1               PIC X(17) VALUE 'DSQEC_NLFCMD_LANG'.
   Ø3  NNAME2               PIC X(14) VALUE 'DSQDC_COST_EST'.
   Ø3  NNAME3               PIC X(17) VALUE 'DSQDC_DISPLAY_RPT'.
* VARIABLE VALUE PARAMETERS
 Ø1  VVALUES.
   Ø3 VVAL1                 PIC X(12).
 Ø1  NVALUES.
   Ø3   NVALS               PIC 9(8) COMP-4 OCCURS 1Ø.
* WORK VARIABLES
 Ø1  I                      PIC 9(4) USAGE IS COMP-4.
 Ø1  J                      PIC 9(4) USAGE IS COMP-4.
 Ø1  K                      PIC 9(4) USAGE IS COMP-4.
 Ø1  NB-COLS                PIC 9(4) USAGE IS COMP-4.
 Ø1  COLS.
    Ø3  COL-START           PIC 9(4) COMP-4 OCCURS 5Ø.
    Ø3  COL-LENGTH          PIC 9(4) COMP-4 OCCURS 5Ø.
 Ø1  TEMP                   PIC 9(8) USAGE IS COMP-4.
 Ø1  TMP2                   PIC 9(4) USAGE IS COMP-4.
* LENGTH
 Ø1  LONGUEUR               PIC 9(4) USAGE IS COMP-4.
 Ø1  PGM-NAME               PIC X(8).
 Ø1  C-QMF-PROC             PIC X(18).
 Ø1  CHAR-LF                PIC X(1)  VALUE X'15'.
 Ø1  DOCUMENT-TYPE          PIC X(4)  VALUE SPACES.
 Ø1  TEMP1                  PIC X(32ØØØ) VALUE SPACES.
 Ø1  TEMP2                  PIC X(32ØØØ) VALUE SPACES.
 Ø1  WS-POINTER             PIC S9(Ø8)  VALUE +1 BINARY.
* USED BY NOVATION FOR NOT TRANSLATE A BINARY DOCUMENT
 Ø1  NOVABIN-LIT            PIC X(7)  VALUE 'NOVABIN'.
* HTTP HEADER
 Ø1  OUTPUT-HEADERS.
     Ø5  MSG-OK                  PIC X(15) VALUE
         'HTTP/1.Ø 2ØØ OK'.
     Ø5  HEADER-PDF              PIC X(29) VALUE
         'Content-Type: application/pdf'.
     Ø5  CL-HEADER.
         1Ø CL-HEADER-LIT            PIC X(16) VALUE
                                  'Content-Length: '.
         1Ø CL-HEADER-VALUE          PIC 9(Ø9) VALUE ZERO
                                     USAGE DISPLAY.
 LINKAGE SECTION.
*---------------
 Ø1 DFHCOMMAREA.
    Ø3 COMMAREA          PIC X(32ØØØ).
```

```
 PROCEDURE DIVISION USING DFHCOMMAREA.
*
* INIT VARIABLES
* PUT YOU QMF PROC NAME HERE
     MOVE 'CTISØØ3.PPROVINCE' TO C-QMF-PROC.
     EXEC CICS ASSIGN USERID(USERID-PRINT)
                      NOHANDLE
     END-EXEC.
     MOVE USERID-PRINT TO USERID-EXPORT.
*
* FIND DOCUMENT TYPE REQUESTED IN HTTP REQUEST
     MOVE Ø TO TEMP.
     INSPECT COMMAREA TALLYING TEMP FOR ALL '.PDF' '.pdf'
     IF TEMP > Ø THEN
        MOVE 'PDF' TO DOCUMENT-TYPE
     ELSE
        INSPECT COMMAREA TALLYING TEMP FOR ALL '.RTF' '.rtf'
        IF TEMP > Ø THEN
           MOVE 'RTF' TO DOCUMENT-TYPE
        ELSE
           INSPECT COMMAREA TALLYING TEMP FOR ALL '.CSV' '.csv'
           IF TEMP > Ø THEN
              MOVE 'CSV' TO DOCUMENT-TYPE
           ELSE
              MOVE 'HTML' TO DOCUMENT-TYPE
           END-IF
        END-IF
     END-IF.
     MOVE LOW-VALUES TO COMMAREA.
*
* START A QUERY INTERFACE SESSION
     MOVE DSQ-CURRENT-COMM-LEVEL TO DSQ-COMM-LEVEL.
     MOVE Ø TO QICLTH.
     INSPECT STARTQI TALLYING QICLTH FOR CHARACTERS.
     MOVE Ø TO TEMP.
     INSPECT SNAME1 TALLYING TEMP FOR CHARACTERS.
     MOVE TEMP TO KLTHS(1).
     MOVE Ø TO TEMP.
     INSPECT SNAME2 TALLYING TEMP FOR CHARACTERS.
     MOVE TEMP TO KLTHS(2).
     MOVE Ø TO TEMP.
     INSPECT SNAME3 TALLYING TEMP FOR CHARACTERS.
     MOVE TEMP TO KLTHS(3).
     MOVE Ø TO TEMP.
     INSPECT SNAME4 TALLYING TEMP FOR CHARACTERS.
     MOVE TEMP TO KLTHS(4).
     MOVE Ø TO TEMP.
     INSPECT SNAME5 TALLYING TEMP FOR CHARACTERS.
     MOVE TEMP TO KLTHS(5).
     MOVE Ø TO TEMP.
```

```
      INSPECT SNAME6 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO KLTHS(6).
      MOVE Ø TO TEMP.
      INSPECT SVALUE1 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(1).
      MOVE Ø TO TEMP.
      INSPECT SVALUE2 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(2).
      MOVE Ø TO TEMP.
      INSPECT SVALUE3 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(3).
      MOVE Ø TO TEMP.
      INSPECT SVALUE4 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(4).
      MOVE Ø TO TEMP.
      INSPECT SVALUE5 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(5).
      MOVE Ø TO TEMP.
      INSPECT SVALUE6 TALLYING TEMP FOR CHARACTERS.
      MOVE TEMP TO VLTHS(6).
      MOVE 6 TO QIPNUM.
      CALL DSQCIB  USING DSQCOMM, QICLTH, STARTQI,
                         QIPNUM, QIKLTHS, SNAMES,
                         QIVLTHS, SVALUES, DSQ-VARIABLE-CHAR.
*
* INIT QMF VARIABLES WITH QMF SETGLOBAL COMMAND
      PERFORM Ø1-SETGLOBAL-QMF.
* EXEC PROC
      MOVE C-QMF-PROC TO PROCNAM.
      MOVE Ø TO QICLTH.
      INSPECT PROC  TALLYING QICLTH FOR CHARACTERS.
      CALL DSQCIB USING DSQCOMM, QICLTH, PROC.
* PRINT OR EXPORT REPORT DEPENDING ON THE REQUESTED FILE TYPE
      IF DOCUMENT-TYPE = 'HTML' OR DOCUMENT-TYPE = SPACES THEN
* EXPORT REPORT IN HTML
         MOVE Ø TO QICLTH
         INSPECT EXPORT TALLYING QICLTH FOR CHARACTERS
         CALL DSQCIB USING DSQCOMM, QICLTH, EXPORT
      ELSE
* PRINT REPORT
         MOVE Ø TO QICLTH
         INSPECT PRINT TALLYING QICLTH FOR CHARACTERS
         CALL DSQCIB USING DSQCOMM, QICLTH, PRINT
      END-IF
* END THE QUERY INTERFACE SESSION
      MOVE Ø TO QICLTH.
      INSPECT ENDQI TALLYING QICLTH FOR CHARACTERS.
      CALL DSQCIB USING DSQCOMM, QICLTH, ENDQI.
* GENERATE DOCUMENT AS REQUESTED.
      EVALUATE DOCUMENT-TYPE
```

```
        WHEN 'HTML' PERFORM Ø1-BUILD-QMFHTML
        WHEN SPACES PERFORM Ø1-BUILD-QMFHTML
        WHEN 'RTF'  PERFORM Ø1-BUILD-QMFRTF
        WHEN 'CSV'  PERFORM Ø1-BUILD-QMFCSV
        WHEN 'PDF'  PERFORM Ø1-BUILD-QMFPDF
        WHEN OTHER PERFORM Ø1-BUILD-QMFHTML
     END-EVALUATE.
* DELETE QMF REPORT TS QUEUE
     EXEC CICS DELETEQ TS QUEUE(USERID-PRINT)
                          NOHANDLE
     END-EXEC.
* END PROGRAM
     EXEC CICS RETURN
     END-EXEC.
     GOBACK.
*
* INIT QMF VARIABLES WITH QMF SETGLOBAL COMMAND
*
 Ø1-SETGLOBAL-QMF.
* SET SOME GLOBAL VARIABLES FOR QMF
     MOVE Ø TO QICLTH
     INSPECT SETG TALLYING QICLTH FOR CHARACTERS
     MOVE Ø TO TEMP
     INSPECT NNAME1 TALLYING TEMP FOR CHARACTERS
     MOVE TEMP TO KLTHS(1)
     MOVE 1 TO NVALS(1)
     MOVE 4 TO VLTHS(1)
     MOVE Ø TO TEMP
     INSPECT NNAME2 TALLYING TEMP FOR CHARACTERS
     MOVE TEMP TO KLTHS(2)
     MOVE Ø TO NVALS(2)
     MOVE 4 TO VLTHS(2)
     MOVE Ø TO TEMP
     INSPECT NNAME3 TALLYING TEMP FOR CHARACTERS
     MOVE TEMP TO KLTHS(3)
     MOVE Ø TO NVALS(3)
     MOVE 4 TO VLTHS(3)
     MOVE 3 TO QIPNUM
*
     CALL DSQCIB  USING DSQCOMM, QICLTH, SETG,
                        QIPNUM, QIKLTHS, NNAMES,
                        QIVLTHS, NVALUES, DSQ-VARIABLE-FINT.
*
* BUILD THE PDF DOCUMENT USING QMF REPORT
*
 Ø1-BUILD-QMFPDF.
* LINK TO PDF DOCUMENT GENERATION ROUTINE
     MOVE LENGTH OF TEMP1 TO LONGUEUR.
     MOVE LOW-VALUES TO TEMP1.
     MOVE 'QMFPDF' TO PGM-NAME.
```

```
     EXEC CICS LINK PROGRAM(PGM-NAME)
                    COMMAREA(TEMP1)
                    LENGTH(LONGUEUR)
                    NOHANDLE
     END-EXEC.
* COMPUTE LENGTH OF PDF DOCUMENT
     MOVE LOW-VALUES TO DFHCOMMAREA(1 : EIBCALEN)
     INSPECT TEMP1 TALLYING CL-HEADER-VALUE
             FOR CHARACTERS BEFORE X'ØØØØØØØØØØØØØØØØØØØØØØ'.
* BUILD HTTP HEADER
     STRING NOVABIN-LIT,
            MSG-OK,          CHAR-LF,
            HEADER-PDF,      CHAR-LF,
            CL-HEADER,       CHAR-LF,
                             CHAR-LF DELIMITED SIZE
         INTO DFHCOMMAREA
         WITH POINTER WS-POINTER
     END-STRING
* PUT PDF DOCUMENT AFTER HTTP HEADER
     MOVE TEMP1 TO COMMAREA(WS-POINTER:).
*
* BUILD THE RTF DOCUMENT USING QMF REPORT
*
 Ø1-BUILD-QMFRTF.
* BUILD RTF DOCUMENT
     MOVE LOW-VALUES TO TEMP1.
     STRING
        '{\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 '
        DELIMITED BY SIZE
        INTO TEMP1
     END-STRING
     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 '\par' AND LINE FEED AT THE END
             IF EIBRESP = ZEROES AND EIBRESP2 = ZEROES THEN
                STRING TEMP1  DELIMITED BY LOW-VALUES
                       TEMP2  DELIMITED BY '                    '
```

```
                    '\par' DELIMITED BY SIZE
                    x'15' DELIMITED BY SIZE
                    INTO TEMP1
                END-STRING
             END-IF
     END-PERFORM.
* CLOSE END TAG OF RTF
     STRING TEMP1  DELIMITED BY LOW-VALUES
            '}' DELIMITED BY SIZE
            INTO TEMP1
     END-STRING.
* CLEAN WORKING AND WRITE IT TO COMMAREA
     INSPECT TEMP1 REPLACING ALL LOW-VALUES BY SPACES.
     MOVE TEMP1 TO DFHCOMMAREA.
*
* BUILD THE CSV DOCUMENT USING QMF REPORT
*
 Ø1-BUILD-QMFCSV.
* BUILD CSV DOCUMENT
     MOVE LOW-VALUES TO TEMP1 TEMP2 DFHCOMMAREA.
     MOVE ZEROES TO I J K COLS.
     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
             IF EIBRESP = ZEROES AND EIBRESP2 = ZEROES THEN
                MOVE ZEROES TO TEMP
* SEARCH DASHED LINE BELOW COLUMN HEADERS
                INSPECT TEMP2 TALLYING TEMP FOR ALL '-'
                IF TEMP > 15 THEN
* IF FOUND, SEARCH SPACES BETWEEN DASHES TO FIND COLUNM STARTS
                   PERFORM VARYING I FROM 1 BY 1
                           UNTIL I > LONGUEUR
                           IF TEMP2(I:2) = ' -' THEN
                              ADD 1 TO J
                              ADD 1 TO I GIVING COL-START(J)
                           END-IF
                   END-PERFORM
                   MOVE J TO NB-COLS
* NOW COMPUTE COLUMN LENGTH
                   PERFORM VARYING I FROM 1 BY 1
                           UNTIL I >= NB-COLS
                                 ADD 1 TO I GIVING J
                                 SUBTRACT  COL-START(J)
                                    FROM   COL-START(I)
```

```
                                GIVING COL-LENGTH(I)
                            END-PERFORM
```

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

Pierre Delaunoy
Director
Ministère de la Communauté Française (Belgium)

CICS questions and answers

Q: What options do I have to handle errors with the CICS Web Services? How can I handle mistyped URLs for example?

A: There are two main options. Both enable the masking of '500 Internal Server Error', '400 Bad Request', and '404 Not Found' messages that can be thrown out by mistyped URLs, undefined transactions, and genuine abends.

The URM DFHWBEP (Web error program) is invoked whenever CICS Web Support flags an error. It receives as input any abend code, error code, and reason code set by CICS itself, the analyser, or the converter. This information can then be used to override the CICS reply to create up to a 32KB HTTP response and to set the HTTP return-code.

The error URM is invoked for most error conditions, apart from basic authentication errors, including abends and failures to start alias transactions. However, you cannot use the WEB/DOCUMENT API to build your response, so some knowledge of HTTP datastreams will be required.

Alternatively, you can check for analyser/converter errors, like the URL being wrong, and simply direct the request to a Web-aware program that can build a response. This requires more thought in determining what you want to catch and why, and how to pass this information across to the Web-aware error program. Fortunately this method allows the use of the

WEB/DOCUMENT API to construct your response – so a simple doctemplate containing a friendlier error page is a possibility.

Be careful how much helpful error information you return. Helpful diagnostics may serve to alarm customers and delight hackers, so should be returned only in testing scenarios.

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.

MacKinney Systems has announced Version 1.3 of its CICS/SignOn, which adds 17 new API functions plus two new batch cross-reference reports.

A new 150-byte user-data area in the user profile is available for storing information such as department, phone number, etc. Additionally, a password history area has been added where users can choose to store up to six passwords to prevent re-use.

For further information contact:
MacKinney Systems, 2740 S Glenstone Ave, Suite 103, Springfield, MO 65804-3737 USA.
Tel: (417) 882 8012.
URL: http://www.mackinney.com/products/cics.

* * *

Jacada has added a new connector to its Jacada Integrator for directly accessing IBM mainframe CICS and IMS transactions and is said to pave the way for additional transaction connectors to CA-IDMS/DC, IBM TPF, HP3000, and Fujitsu Siemens BS2000 mainframes.

Integrator is used for integrating core legacy business systems, including the data and processes in those systems, with any front-office, call centre, CRM, or Web application as well as with a wide variety of middleware. The new Jacada Integrator Host Transaction Connector enables any type of developer to access CICS and IMS mainframe transactions securely and non-intrusively, without relying on presentation logic such as the 3270 data streams.

It transforms CICS and IMS transactions into services that can be deployed natively in a J2EE environment as a JCA resource adapter or as an EJB, or as XML, Web services, and .NET components.

The new ability to access CICS and IMS transactions directly is released as an integrated feature within the Integrator suite of products. Combined with the existing capabilities of Jacada Integrator Host Screen Connector, sites can access the business logic and data in mainframe environments, either through a non-intrusive screen-based approach or by directly connecting to the existing CICS or IMS transactions.

For further information contact:
Jacada, 400 Perimeter Center Terrace, Suite 100, Atlanta, GA 30346, USA.
Tel: (770) 352 1300.
URL: http://www.jacada.com.

* * *

MacKinney Systems has announced its new (Version 1.0) Dump Detective.

Dump Detective is an interactive aid in the investigation and interpretation of dumps given by the operating system and CICS on encountering a program failure.

For further information contact:
MacKinney Systems, 2740 S Glenstone Ave, Suite 103, Springfield, MO 65804-3737 USA.
Tel: (417) 882 8012.
URL: http://www.mackinney.com/products/other/dump_detective.htm.