CICS update

240

November 2005

In this issue

A tc PUBLICATION

CICS Update

Published by
Xephon Inc
PO Box 550547
Dallas, Texas 75355
USA

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

Editor
Trevor Eddolls
E-mail: trevore@xephon.com

Publisher
Colin Smith
E-mail: info@xephon.com

Subscriptions and back-issues
A year's subscription to *CICS Update*, comprising twelve monthly issues, costs $270.00 in the USA and Canada; £175.00 in the UK; £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 2000 issue, are available separately to subscribers for $24.00 (£16.00) 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.

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

The Open Transaction Environment – history, evolution, and background

CICS has continually evolved throughout its lifetime. With the introduction of CICS Transaction Server 3.1, CICS has now been enhanced to provide support for new functionality such as channels and containers, Web services, and further extensions to its Open Transaction Environment (OTE) support. This article describes the background to OTE within CICS, and explains how the evolution of this component of CICS has led to support for multiple types of programming environment being able to execute in parallel within the CICS environment.

AN OPEN TRANSACTION ENVIRONMENT WITHIN CICS

OTE makes use of open TCBs. It is important to realize that the term 'open TCB' refers to the Open Transaction Environment. It does not specifically refer to 'open MVS' (ie OMVS, an earlier term for Unix Systems Services, or USS). Unix System Services is the component of z/OS that supports various Unix services such as the Hierarchical File System (HFS).

OTE allows particular programs within CICS to be dispatched and run independently of the CICS QR TCB. They execute under separate open TCBs. These are dispatched independently to the QR TCB by the z/OS dispatcher, and so can execute in parallel with the Quasi-Reentrant (QR) TCB, being dispatched on different Central Processors (CPs) concurrently. This separation of work across multiple TCBs within a particular CICS region helps to address the transaction throughput constraint that could otherwise occur when sub-dispatching all the work under a single TCB. It also addresses the problem of TCB suspension (or blocking), which can be encountered when using z/OS services from within a CICS application environment. These services are restricted from use by those CICS applications running under the QR TCB because they would cause the QR TCB to be suspended while being performed by z/OS. Suspending the QR TCB would in turn prevent CICS from sub-dispatching other tasks during the time the QR TCB was suspended. It would impact CICS transaction throughput.

OTE avoids this problem by limiting the effect of TCB blocking to the particular application running under its own open TCB. By doing this, the impact of suspending the TCB does not affect the transaction throughput from other applications running under other TCBs within the CICS system.

CICS automatically handles the switching of tasks between the OTE open TCB modes and the QR TCB. The open TCB modes are predefined by CICS; users cannot define their own types of open TCB for OTE to use.

True parallel processing within CICS exploits multi-engine hardware – prior to OTE, work running under the single QR TCB could utilize only a single CP at any one time. By providing dedicated open TCBs for specific application programs to use, OTE can support programming environments that could not otherwise be used with the traditional QR TCB dispatching mechanism. The different programming environments that exploit the different types of OTE TCB are all very specific, and each has its own characteristics and functional requirements. These environments are described below.

OTE DEPLOYMENT HISTORY

To understand the way in which OTE is used within CICS it is important to appreciate the history of the development of this component of the product. OTE has been developed in a phased manner, and its functionality has grown from release to release of CICS.

The original support for OTE was introduced in the base level of CICS Transaction Server 1.3. It provided the ability to run one specific type of program environment under a separate open TCB to the CICS QR TCB, which until then was the ‘workhorse’ TCB used for sub-dispatching traditional CICS workloads. When CICS Transaction Server 1.3 was shipped, the only type of supported OTE-managed open TCB was the J8 open TCB mode. This was provided to enable CICS to support Java application programs. J8 TCBs were provided to execute Java Virtual Machines (JVMs), which in turn could interpret users’ Java applications (Java class files comprised interpretable bytecodes).

The naming convention of open TCB names mapped to the type of OTE open TCB mode. For J8 TCBs, the letter J meant JVM programs and the number 8 denoted that the program environment would run in storage protection key 8 (‘CICS key’ storage).

After CICS Transaction Server 1.3 was shipped, CICS Java support was enhanced to support ‘hot-pooling’ for compiled Java applications processed using the Enterprise Toolkit for OS/390 Compiler and Binder. This type of Java program environment was also referred to as the High Performance Java (HPJ) compiler and run-time. With hot-pooling, a new OTE open TCB mode was introduced. This was the H8 mode. (Hot-pooling was intended as a transient solution to Java programming within the CICS environment. The JVM was the strategic platform for running Java programs, and while its performance and scalability were being improved by IBM, hot-pooling provided a means of executing a reasonable Java program throughput for CICS applications.)

The different types of open TCB mode are very specific to their type of program environment, and OTE handles them in different ways. The type of open TCB mode assigned to a particular program (J8 for a JVM, H8 for a Java hot-pooled program) was determined by CICS, and not specified by the CICS systems programmer nor by the application program itself. Also, it was not possible to exploit OTE for other types of program. Only JVMs and Java hot-pooled programs could utilize OTE open TCBs in CICS Transaction Server 1.3.

Support for OTE was extended with CICS Transaction Server 2.2. A new open TCB mode was introduced – the L8 mode, for OPENAPI Task-Related User Exit programs. CICS Transaction Server 2.2 changed the CICS DB2 Attachment facility to be able to exploit OTE and use L8 open TCBs to process DB2 requests from CICS directed to DB2 6.1 systems (and above). For connections to DB2 5.1 systems (and below), the CICS Transaction Server 2.2 DB2 Adapter still used the old, privately-managed, TCB mechanism that was used in earlier releases of CICS.

The big advantage of utilizing open TCBs for DB2 requests, instead of the privately-managed ones, was seen when CICS DB2 application programs were defined with a CONCURRENCY program definition attribute of THREADSAFE instead of the default value of QUASIRENT. The CICS DB2 Adapter in CICS Transaction Server 2.2 could return control to a threadsafe application under the L8 TCB used for the DB2 request, rather than having to resume the QR TCB, as was the case when the old privately-managed TCB mechanism was used in earlier releases of CICS. (This used a private wait/post mechanism, rather than the full TCB switching as implemented by the CICS dispatcher domain for OTE support.) Such an approach meant the need for TCB switches (from QR TCB to CICS DB2 Adapter TCB, then back again) could be avoided for calls to DB2 from the application. In an ideal case, the application could remain running under its L8 TCB for the duration of its work, until its eventual syncpoint when CICS would need to use the QR TCB for some of the syncpointing work, and so TCB switches would be seen during the commitment of the Unit Of Work (UOW). In reality, switches back to QR from the L8 TCB were dictated by what work the application had to perform upon return from DB2. Not all the EXEC CICS API or SPI functionality was itself threadsafe, and so CICS had to perform automatic switches back to the QR TCB in order to process such requests.

CICS Transaction Server 2.3 continued the enhancements to

OTE by making additional parts of the CICS logic threadsafe, so reducing the likelihood that a threadsafe application, running under an open TCB, would be forced to switch back to the QR TCB in order to honour the command.

OTE has been further enhanced with CICS Transaction Server 3.1. There are now additional open TCB modes. The JVM environment provides J8 and J9 TCBs (for CICS-key and user-key Java applications). These are capped by the MAXJVMTCBS System Initialization (SIT) parameter. Similarly, the OPENAPI support provides L8 and L9 TCBs (again, for CICS-key and user-key programs), capped by MAXOPENTCBS. A new form of OTE open TCB mode has been introduced to support the use of the XPLINK of z/OS within C and C++ applications. This is X mode, and X8 and X9 open TCBs are provided for such operations in CICS, capped by MAXXPTCBS. In addition, the CICS Security domain has been enhanced to utilize OTE for its own (previously privately-managed) pools of TCBs. The Security domain utilizes the SO, SL, SP, and S8 TCBs. Note that the old mechanism for capping the number of sockets domain TCBs was via the SSLTCBS SIT parameter. This is now deprecated, because the function has been replaced by MAXSSLTCBS in CICS Transaction Server 3.1. The latter should be used instead. Note also that support for Java hot-pooling has been withdrawn in CICS Transaction Server 3.1.

The use of the OPENAPI attribute has also been extended, to allow applications themselves to specify that they are to execute under an open TCB when executing application logic. This further frees up the QR TCB to execute other work concurrently. CICS Transaction Server 3.1 has also extended the use of OTE within itself. For example, it runs some of its own programs under L8 TCBs to access doctemplates and HTTP static responses stored on the HFS, and when processing Web service requests and Extensible Mark-up Language (XML). OTE improvements (such as making the EXEC CICS WEB API commands threadsafe) allow this open TCB exploitation within CICS itself.

THREADSAFETY IN OTE

To be able to exploit OTE functionality within CICS, it is important to fully understand the terminology associated with it. One term that is very important from the perspective of TCB usage within CICS is threadsafety. In part this is the ability to ensure that shared resources and internal state data are accessed in a serialized manner by program logic. Threadsafe programs do not have an affinity with being executed on the QR TCB because they are not at risk from other concurrently executing programs running under different TCBs. This in turn means that threadsafe application programs and task-related user exits, global user exit programs, and user-replaceable modules cannot rely on quasi-reentrancy to manage their serialization, because they can run on an open TCB concurrently with the QR TCB.

It is worthwhile re-emphasizing that, by redefining them to be threadsafe, programs and open task-related user exits, global user exit programs, and user-replaceable modules can no longer rely on the quasi-reentrancy they used to acquire by virtue of being dispatched under the QR TCB, because they can now run concurrently on an open TCB. In addition, even those quasi-reentrant programs that are not threadsafe, and so dispatched under the QR TCB as usual, are exposed to problems if they access resources that are also accessible by threadsafe programs running concurrently under an OTE open TCB. Because of this, techniques used to access shared resources must take into account the possibility of simultaneous access by other programs. Programs that use appropriate serialization techniques when accessing shared resources, and have no affinity to the QR TCB, are described as threadsafe.

Particular CICS system programs and user applications may or may not be threadsafe, depending on what work they have to perform, and the way in which they have been written. There are two aspects of threadsafety to be understood – threadsafety of CICS application program logic, and threadsafety of CICS functionality itself when processing EXEC CICS commands issued by the applications, and when processing other work within CICS.

Some CICS commands are capable of being executed under TCBs other than the QR TCB (an example is the EXEC CICS ASSIGN command). These commands are therefore threadsafe EXEC CICS commands and thet have no affinity with being executed under particular TCBs. CICS is content to execute them under the TCB that is currently dispatching the running application program. Being threadsafe, multiple instances of such commands can be executed concurrently, from within multiple applications dispatched under their own TCBs executing under separate processors on the hardware.

Conversely, other CICS commands require the use of the QR TCB, and so are not threadsafe. An example of such a non-threadsafe CICS command is the EXEC CICS READ command to CICS File Control. Such non-threadsafe EXEC CICS commands will automatically be moved to execute under the QR TCB when such a non-threadsafe CICS command is being executed. This movement across TCBs is handled by DFHEIP, the CICS EXEC Interface Program. Whether or not CICS moves the application back from the QR TCB to the open TCB depends on the program's definition attributes, and this leads into the second aspect of threadsafety – the threadsafety of CICS application programs.

When an application program is defined to CICS, the CONCURRENCY attribute specifies whether or not a program is threadsafe. The options available are QUASIRENT and THREADSAFE (the default is QUASIRENT). Quasi-reentrant programs need to execute their own (ie their non EXEC CICS) logic under the QR TCB. Conversely, threadsafe programs are free to execute their own code on other open TCBs; they do not require serialization by being dispatched under the QR TCB.

There is another related program definition attribute that has been provided with CICS Transaction Server 3.1. This is the

API attribute. It can be set to CICSAPI or OPENAPI. The default is CICSAPI. CICSAPI means that the program uses CICS application programming interfaces only. CICS will in turn use the value specified by the CONCURRENCY attribute to decide whether the program executes under the QR TCB or an appropriate OTE open TCB. Conversely, specifying OPENAPI means that the program is not restricted to using the CICS application programming interface. The CICS TS V3.1 *Resource Definition Guide* states that CICS executes the program on its own L8 or L9 mode open TCB (depending on the value of the EXECKEY attribute in the program resource definition). If CICS requires a switch to the QR TCB while processing an EXEC CICS command, then it returns to the open TCB before handing control back to the application program on completion of the command. OPENAPI requires the program to be coded to threadsafe standards and as such defined with CONCURRENCY(THREADSAFE).

This approach (of insisting that the application logic itself be executed under an OPENAPI open TCB) can be contrasted with a threadsafe application that is defined with an API attribute of CICSAPI instead. The THREADSAFE definition means that the application is happy to run under an open TCB, and has no affinity with the QR TCB. However, the lack of OPENAPI means it is not mandatory for the application logic to execute under an open TCB; it could just as easily be executed upon the QR TCB instead.

The main reason for defining programs with an API attribute of OPENAPI is to allow application workloads to be moved off the QR TCB onto multiple open TCBs. This allows better exploitation of machine resources to achieve better throughput. In addition, the use of other (non-CICS) APIs in OPENAPI programs is possible. This is because the blocking of an open TCB by an operating system wait affects only the single application on its own open TCB; it does not affect the whole of CICS, as it would if such a wait occurred under the QR TCB. This means that such OPENAPI programs are not permitted to execute on the QR TCB, because of the risk of blocking the

TCB by an operating system wait and so affecting CICS throughput. OPENAPI programs do still have obligations to the CICS system as a whole, however. IBM has explicitly stated in the documentation describing OPENAPI that the 'use of other (non-CICS) APIs within CICS is entirely at the discretion and risk of the user. No testing of other (non-CICS) APIs within CICS has been undertaken and use of such APIs is not supported by IBM Service.'

Note that an application program may itself be threadsafe, and yet issue non-threadsafe EXEC CICS commands. A program does not have to issue only threadsafe CICS commands in order to be deemed threadsafe. It is important to appreciate that it is the threadsafety of the program itself (not the EXEC CICS commands it issues) that determines how its CONCURRENCY attribute should be specified on the program definition. The threadsafety of a program is specific to its own internal application logic. If it exploits commands such as EXEC CICS ADDRESS CWA, EXEC CICS EXTRACT EXIT, or EXEC CICS GETMAIN SHARED, then a program may not be threadsafe, because these commands provide application access to global storage areas within CICS.

Another reason for non-threadsafety in applications is if they modify themselves at run-time. Such self-modifying programs are safe to operate when they execute in a serialized manner under the QR TCB, provided that they ensure that their state is consistent once more before issuing another EXEC CICS command. This is because the QR TCB dispatches only one task at a time, and if the program contents are consistent across EXEC CICS commands, any transient modifications it makes to itself are shielded from other tasks driving the same program. These other tasks can be sub-dispatched by the CICS dispatcher only when it receives control, and since CICS is a cooperative programming model, this in turn requires the currently running program to issue another EXEC CICS command to reinvoke CICS services and the dispatcher domain once again.

Such non-reentrancy is not due to the exploitation of specific EXEC CICS commands that provide access to shared data, but to the way the program was written to modify itself dynamically.

DFHEIDTH

A sample command table (DFHEIDTH) is provided for use with the CICS load module scanner utility program DFHEISUP. The load module scanner is a z/OS batch utility program. It scans load modules within CICS application program libraries and locates their CICS commands by matching the load module contents with the bit strings generated by the translation of EXEC CICS statements in their source.

The load module scanner produces a report, listing the modules that contain the commands specified by the utility, with their offsets into the programs. By using DFHEIDTH, DFHEISUP can identify those programs that issue EXEC CICS ADDRESS CWA, EXEC CICS EXTRACT EXIT, or EXEC CICS GETMAIN SHARED commands. These commands indicate non-threadsafe application logic. To ensure that the programs are indeed threadsafe, they would need to include appropriate logic to synchronize their behaviour. For example, to protect against concurrent updates to pieces of shared storage by another CICS application program (or programs) executing at the same time under another TCB.

TRACE OF AN OPENAPI APPLICATION IN CICS TRANSACTION SERVER 3.1

The following edited trace shows an example application program that has been defined with an API attribute of OPENAPI and a CONCURRENCY attribute of THREADSAFE:

```
ØØØ37 QR     AP Ø591 APXM  EXIT  INIT_XM_CLIENT/OK
=ØØØ246=
ØØØ37 QR     RM FAØ1 RMUC  ENTRY CREATE_UOW            NO,BACKWARD,Ø
=ØØØ263=
ØØØ37 QR     RM FAØ2 RMUC  EXIT  CREATE_UOW/OK
=ØØØ265=
ØØØ37 QR    XS Ø7Ø1 XSRC  ENTRY CHECK_CICS_RESOURCE   THAW,TRANSATTACH,
=ØØØ266=
ØØØ37 QR    XS Ø7Ø2 XSRC  EXIT  CHECK_CICS_RESOURCE/OK
=ØØØ267=
ØØØ37 QR    AP Ø59Ø APXM  ENTRY BIND_XM_CLIENT
=ØØØ268=
ØØØ37 QR    AP Ø591 APXM  EXIT  BIND_XM_CLIENT/OK
=ØØØ269=
ØØØ37 QR    AP 179Ø TFXM  ENTRY BIND_XM_CLIENT        14C4897Ø ,
Ø25ØØØØØ   =ØØØ278=
ØØØ37 QR    AP 1791 TFXM  EXIT  BIND_XM_CLIENT/OK     YES,THREAD,YES
=ØØØ279=
ØØØ37 QR    PG Ø9Ø1 PGPG  ENTRY INITIAL_LINK          THREAD
=ØØØ28Ø=
ØØØ37 QR    LD ØØØ1 LDLD  ENTRY ACQUIRE_PROGRAM       148F7Ø3Ø
=ØØØ283=
ØØØ37 QR    LD ØØØ2 LDLD  EXIT  ACQUIRE_PROGRAM/OK
=ØØØ29Ø=
ØØØ37 QR    AP 194Ø APLI  ENTRY ESTABLISH_LANGUAGE    THREAD,ØØØC1ØØØ
=ØØØ291=
ØØØ37 QR    AP 1941 APLI  EXIT  ESTABLISH_LANGUAGE/OK ASSEMBLER,
=ØØØ298=
ØØØ37 QR    AP 194Ø APLI  ENTRY START_PROGRAM         THREAD
=ØØØ299=
ØØØ37 QR    DS ØØØ8 DSIT  ENTRY ADD_TCB               L8 -OPEN,L8
=ØØØ3ØØ=
ØØØ37 QR    DS ØØØ9 DSIT  EXIT  ADD_TCB/EXCEPTION     MODE_NOT_ACTIVE
=ØØØ3Ø5=
ØØØ37 QR    DS ØØØ8 DSIT  ENTRY ACTIVATE_MODE         L8 -OPEN,L8,EXEC
=ØØØ3Ø6=
ØØØ37 QR    DS ØØØ9 DSIT  EXIT  ACTIVATE_MODE/OK      ØØØØØØØC
=ØØØ311=
ØØØ37 QR    DS ØØØ8 DSIT  ENTRY ADD_TCB               L8 -OPEN,L8
=ØØØ312=
ØØØ37 QR    DS ØØØ9 DSIT  EXIT  ADD_TCB/OK            13DØ37D8 ,
ØØØØØØØ1   =ØØØ319=
DSTCB L8ØØØ DS ØØ12 DSKE  ENTRY TCB_REPLY             13CF28ØØ,13DØ37D8
=ØØØ322=
ØØØ37 L8ØØØ AP ØØE1 EIP   ENTRY WRITEQ-TS
=ØØØ325=
ØØØ37 L8ØØØ TS ØCØ1 TSMB  ENTRY MATCH                 ANDY
=ØØØ326=
ØØØ37 L8ØØØ TS ØCØ2 TSMB  EXIT  MATCH/OK              ,,,ANDY,,
=ØØØ329=
ØØØ37 L8ØØØ TS Ø2Ø1 TSQR  ENTRY WRITE                 ANDY,AUXILIARY
=ØØØ33Ø=
ØØØ37 L8ØØØ TS Ø9Ø1 TSAM  ENTRY WRITE_AUX_DATA        ØØØC1ØDE
=ØØØ341=
ØØØ37 L8ØØØ TS Ø9Ø2 TSAM  EXIT  WRITE_AUX_DATA/OK     1,ØØØØØØØ1
=ØØØ342=
ØØØ37 L8ØØØ TS Ø2Ø2 TSQR  EXIT  WRITE/OK               1
=ØØØ343=
ØØØ37 L8ØØØ AP ØØE1 EIP   EXIT  WRITEQ-TS              OK
=ØØØ344=
ØØØ37 L8ØØØ AP ØØE1 EIP   ENTRY SEND-TEXT
=ØØØ345=
ØØØ37 QR    AP ØØFA BMS   ENTRY SEND-OUT               CTRL
=ØØØ35Ø=
ØØØ37 QR    AP ØØFA BMS   EVENT RLR-ENTRY
=ØØØ355=
ØØØ37 QR    AP ØØFA BMS   EVENT RLR-EXIT
=ØØØ358=
ØØØ37 QR    AP ØØFA BMS   EVENT PBP-ENTRY
=ØØØ359=
ØØØ37 QR    AP ØØFA BMS   EVENT M32-ENTRY
=ØØØ362=
ØØØ37 QR    AP ØØFA BMS   EVENT M32-EXIT
=ØØØ371=
ØØØ37 QR    AP ØØFA BMS   EVENT TPP-ENTRY
=ØØØ372=
ØØØ37 QR    AP FDØ1 ZARQ  ENTRY APPL_REQ               14C4897Ø,WRITE
=ØØØ373=
ØØØ37 QR    AP FD81 ZARQ  EXIT  APPL_REQ
=ØØØ374=
ØØØ37 QR    AP ØØFA BMS   EVENT TPP-EXIT
=ØØØ375=
ØØØ37 QR    AP ØØFA BMS   EVENT PBP-EXIT
=ØØØ376=
ØØØ37 QR    AP ØØFA BMS   EXIT
=ØØØ379=
ØØØ37 L8ØØØ AP ØØE1 EIP   EXIT  SEND-TEXT              OK
=ØØØ38Ø=
ØØØ37 L8ØØØ AP ØØE1 EIP   ENTRY RETURN
=ØØØ381=
ØØØ37 L8ØØØ AP 1941 APLI  EXIT  START_PROGRAM/OK       ,NO,THREAD
=ØØØ384=
ØØØ37 L8ØØØ LD ØØØ1 LDLD  ENTRY RELEASE_PROGRAM        148F7Ø3Ø,ØØØC1ØØØ
=ØØØ385=
ØØØ37 L8ØØØ LD ØØØ2 LDLD  EXIT  RELEASE_PROGRAM/OK     ØØØC1ØØØ,16Ø,SDSA
=ØØØ386=
ØØØ37 QR    PG Ø9Ø2 PGPG  EXIT  INITIAL_LINK/OK
=ØØØ389=
ØØØ37 QR    RM FA11 RMUO  ENTRY COMMIT_UOW             NO
=ØØØ39Ø=
ØØØ37 QR    RM FA12 RMUO  EXIT  COMMIT_UOW/OK
=ØØØ419=
ØØØ37 QR    AP Ø59Ø APXM  ENTRY RELEASE_XM_CLIENT      NORMAL
=ØØØ424=
```

When task 37 receives control, transaction initialization work takes place. This has to execute under the QR TCB as aspects of CICS transaction initialization are not themselves threadsafe. Once the initial link to the application program takes place (called THREAD in this example), CICS will load the program and prepare to start it (AP 1941 at trace number =000299=). Because the program has been defined with an API attribute of OPENAPI and a CONCURRENCY attribute of THREADSAFE, the CICS dispatcher domain will switch environments to an open TCB. In this example, the L8 mode had yet to be activated, so the dispatcher had to activate the mode (DS 0008 at =000306=) then add the new L8000 TCB (DS 0008 at =000312=). The application then received control under the control of this open TCB. The TCB name (QR or L8000 in this example section of trace) is given in the second column of the formatted trace data.

The first EXEC CICS command it issued was an EXEC CICS WRITEQ to a temporary storage queue (AP 00E1 at =000325=). Since the Temporary Storage domain logic is itself threadsafe, CICS was able to process this command under the open L8000 TCB. When control returned to the application, the program logic itself continued running under the open TCB. This is because the application program definition of OPENAPI made it expect to receive control from CICS under an open TCB environment. Eventually, it issued another EXEC CICS command; in this example, the next command it issued was an EXEC CICS SEND TEXT (AP 00E1 at =000345=). Because BMS is not threadsafe, CICS switched control from the open L8000 TCB back to the QR TCB for the duration of the request. Upon completion of the EXEC CICS SEND TEXT command, control returned to the application from DFHEIP under the L8 TCB once more. The application then issued an EXEC CICS RETURN command (shown by trace entry AP 00E1 at =000381=). Transaction termination was initiated under the open TCB, and CICS had to switch back to the QR TCB to perform aspects of the syncpoint for the end of task processing. Note that an EXEC CICS RETURN command is one of the very few ENTRY trace points that have no corresponding EXIT also traced. This is because an EXEC CICS RETURN, by definition, is returning control from the current application program, and so the thread of execution and flow of control will not return into this particular program upon completion of the request. Finally, the transaction completed and task 37 left the CICS system.

SUMMARY AND CONTACT INFORMATION

I hope that this article has explained the background and evolution of OTE from its inception in CICS Transaction Server 1.3 through to the latest features supported in CICS Transaction Server 3.1, and also the implications for multiple TCBs executing within a CICS system.

Readers who wish to discuss the material in this article further may contact me via e-mail, at andy_wright@uk.ibm.com.

Andy Wright
CICS Change Team Programmer
IBM (UK)

Presenting CICS DB2 resource statistics from SMF 110 records – part 2

This month we conclude the code for an application designed to present the subset of information collected by type 110, subtype 002, SMF records that deals with CICS DB2 resource statistics.

```
L2 = requestinfo.Ø + 1
requestinfo.Ø = L2
requestinfo.L2 = X2D(D2R_COMMITS)
L2 = requestinfo.Ø + 1
requestinfo.Ø = L2
requestinfo.L2 = X2D(D2R_ABORTS)
L2 = requestinfo.Ø + 1
requestinfo.Ø = L2
```

```
  requestinfo.L2 = X2D(D2R_SINGLE_PHASE)
  L2 = requestinfo.Ø + 1
  requestinfo.Ø = L2
  requestinfo.L2 = X2D(D2R_THREAD_REUSE)
  L2 = requestinfo.Ø + 1
  requestinfo.Ø = L2
  requestinfo.L2 = X2D(D2R_THREAD_TERM)
  L2 = requestinfo.Ø + 1
  requestinfo.Ø = L2
  requestinfo.L2 = X2D(D2R_THREAD_WAIT_OR_OVERFL)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_THREAD_LIMIT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_THREAD_CURRENT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_THREAD_HWM)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_PTHREAD_LIMIT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_PTHREAD_CURRENT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_PTHREAD_HWM)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_TASK_CURRENT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_TASK_HWM)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_TASK_TOTAL)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_READYQ_CURRENT)
  L3 = performanceinfo.Ø + 1
  performanceinfo.Ø = L3
  performanceinfo.L3 = X2D(D2R_READYQ_HWM)
End   /* do forever */
Address TSO "execio Ø diskr indd2(finis"
Queue Copies(' ', 45) || "Resource statistics - resource information"
Queue Copies(' ', 132)
Queue "     DB2Entry        Plan        PlanExit                 ",
      "Auth        Account           Thread        Thread"
Queue "     name            name        name            Auth id   ",
      "type         records           wait          prty"
Queue "     " || Copies('=', 1Ø5)
Address TSO "Execio 5 Diskw outdd1"
Do IRES = 1 To resourceinfo.Ø By 8
  I11 = IRES
  I21 = IRES + 1
  I31 = IRES + 2
  I41 = IRES + 3
  I51 = IRES + 4
  I61 = IRES + 5
  I71 = IRES + 6
  I81 = IRES + 7
  Queue "     " || Left(resourceinfo.I11, 16) || ,
                   Left(resourceinfo.I21, 12) || ,
                   Left(resourceinfo.I31, 16) || ,
                   Left(resourceinfo.I41, 11) || ,
                   Left(resourceinfo.I51, 12) || ,
                   Left(resourceinfo.I61, 18) || ,
                   Left(resourceinfo.I71, 14) || ,
                   Left(resourceinfo.I81, 6)
  Address TSO "Execio 1 Diskw outdd1"
End
Queue Copies(' ', 132)
Queue Copies(' ', 45) || "Resource statistics - request information"
Queue Copies(' ', 132)
Queue "     DB2Entry  Call   Signon  Partial  Commit  Abort  Single",
      "  Thread  Thread  Thread         "
Queue "     name      count  count   signon   count   count  phase",
      "   reuse   terms   waits/overflows"
Queue "     " || Copies('=', 89)
Address TSO "Execio 6 Diskw outdd1"
Do IREQ = 1 To requestinfo.Ø By 1Ø
  I12 = IREQ
  I22 = IREQ + 1
  I32 = IREQ + 2
  I42 = IREQ + 3
  I52 = IREQ + 4
  I62 = IREQ + 5
  I72 = IREQ + 6
  I82 = IREQ + 7
  I92 = IREQ + 8
  I1Ø2 = IREQ + 9
  Queue "     " || Left(requestinfo.I12, 1Ø) || ,
                   Right(requestinfo.I22, 5) || ,
                   Right(requestinfo.I32, 8) || ,
                   Right(requestinfo.I42, 9) || ,
                   Right(requestinfo.I52, 8) || ,
                   Right(requestinfo.I62, 7) || ,
                   Right(requestinfo.I72, 8) || ,
                   Right(requestinfo.I82, 9) || ,
                 Right(requestinfo.I92, 8) || ,
                 Right(requestinfo.I1Ø2, 8)
  Address TSO "Execio 1 Diskw outdd1"
End
Queue Copies(' ', 132)
Queue Copies(' ', 43)||"Resource statistics - performance information"
Queue Copies(' ', 132)
Queue "     DB2Entry  Thread  Thread   Thread  Pthread  Pthread  ",
      "Pthread  Task     Task  Task   Readyq   Readyq"
Queue "     name      limit   current  HWM     limit    current  ",
      "HWM      current   HWM   total  current  HWM"
Queue "     " || Copies('=', 1Ø1)
Address TSO "Execio 6 Diskw outdd1"
Do IPER = 1 To performanceinfo.Ø By 12
  I13 = IPER
  I23 = IPER + 1
  I33 = IPER + 2
  I43 = IPER + 3
  I53 = IPER + 4
  I63 = IPER + 5
  I73 = IPER + 6
  I83 = IPER + 7
  I93 = IPER + 8
  I1Ø3 = IPER + 9
  I113 = IPER + 1Ø
  I123 = IPER + 11
  Queue "     " || Left(performanceinfo.I13, 1Ø) || ,
                 Right(performanceinfo.I23, 6) || ,
                 Right(performanceinfo.I33, 9) || ,
                 Right(performanceinfo.I43, 8) || ,
                 Right(performanceinfo.I53, 9) || ,
                 Right(performanceinfo.I63, 9) || ,
                 Right(performanceinfo.I73, 1Ø) || ,
                 Right(performanceinfo.I83, 9) || ,
                 Right(performanceinfo.I93, 7) || ,
                 Right(performanceinfo.I1Ø3, 7) || ,
                 Right(performanceinfo.I113, 9) || ,
                 Right(performanceinfo.I123, 8)
  Address TSO "Execio 1 Diskw outdd1"
End
Queue Copies(' ', 132)
Queue "   *** End-Of-Report ***"
Queue Copies(' ', 132)
Address TSO "Execio 3 Diskw outdd1"
Address TSO "Execio Ø Diskw outdd1(finis"
Address TSO "free fi(indd2)"
Address TSO "free fi(outdd1)"
Address ISPEXEC "Browse Dataset("outdsn1")"
Address TSO "Delete "outdsn
Address TSO "Delete "outdsn1
Address TSO "Delete "indsn
Exit Ø
```

Nikola Lazovic, Gordana Kozovic
DB2 System Administrators
Postal Savings Bank (Serbia and Montenegro)

CICS case study

Lloyd's of London is the world's leading insurance market, the second largest commercial insurer, and sixth largest reinsurer. The International Underwriting Association of London (IUA) is the world's largest representative organization for international and wholesale insurance and reinsurance companies. The two giants have united with Xchanging, a provider of outsourced services to the insurance industry, to modernize the London insurance market.

In order to be successful, Xchanging needed to resolve incompatibilities between Lloyd's and the IUA's claims and premiums processes, in terms of both business operations and systems integration. As importantly, the group had to avoid the potential problems of a soon-to-be unsupported hardware platform for one of the systems. The business goals were equally compelling: align back-office claims and premiums processing; realize consistent operations processes; share IT resources effectively; lower costs; and improve quality.

As a result of the convergence, Xchanging will provide a single claims and single premiums system for the whole of the London Market.

The Xchanging staff consulted with representatives from the Lloyd's market claims systems user groups and claims practitioners. They also worked with Attunity and 22 other software houses and end users.

The first step was to replace COSS (Claims Office Support System) – the legacy Lloyd’s market electronic claims system used by Xchanging – with CLASS (Claims Loss Advice and Settlement System).

Then, Xchanging set out to consolidate methods of entering claims data while leveraging its existing function-rich software and significant in-house expertise.

To accomplish this, the team elected to put a Java- and HTML-based front end (based on a J2EE Application Server) on a validation and processing engine built using CICS. This decision led to a search for a way to establish communications between J2EE and Web clients on one end and the mainframe element on the other. Xchanging brought in the IT services company Steria to assist in the integration process and to develop the Java/HTML layer. Steria started out with a legacy adapter that was supposed to connect Java with the CICS layer, but developers were frustrated by the fact that the software didn’t make their lives any easier.

The original adapter proved to be inflexible, requiring the development of stored procedures – one for every CICS transaction – and the daunting requirement of another stored procedure for every new CICS transaction. After all parties had spent a great deal of time and effort working towards a suitable answer, Steria suggested Attunity Connect.

The converged system went live without any significant issues. The process converted all COSS-based open claims data to the new system free of errors.

At no time during the testing and implementation phases did current users of CLASS experience any downtime or any serious disruption to their business.

By making data entry more intuitive and user friendly, we could improve throughput, which allowed us to be flexible with how we use people. This also increased our capacity for taking on potential new business without additional headcount.

Attunity accelerated implementation by simplifying connection between the front and back-end systems. The standards-based solution integrates a CICS-based claims processing engine with an Oracle 9*i* Application Server, allowing users to transact business through a Web interface.

In addition, Attunity enables Xchanging to shorten the cycle of what can be a lengthy process of resolving claims. Claims can be very complex. In the past, claims technicians might have to search reams of paper for the correct codes, and it was even more difficult when a claim involved US or Canadian regulations. A wrong code can take a long time to sort out, involving underwriters, brokers, adjusters, and our technicians. Now, input is validated at the Java level, using drop-down boxes and the like. Attunity provides transparent access through the Java front end to the database of codes on the mainframes.

The solution will save the joint venture money by eliminating the need for on-going support for the older platform. However, that was never the key business driver. Xchanging expects to build a totally flexible workforce where skills are interchangeable – enabling the company to focus resources where and when they are most advantageous.

With data entry made more intuitive and user friendly, throughput will improve, which will allow flexibility with how people are used. This will also increase the capacity for taking on potential new business without additional headcount.

Attunity Connect allowed the Xchanging team to merge systems in steps, first processes and then data, without waiting for a complete transfer of operations. Attunity also makes it possible to phase out older systems, reducing exposure to the potential cessation of long-term support.

Because Attunity provides a full complement of native adapters, Xchanging can elect to work through CICS transactions or go directly to the DB2 data itself, in any combination, at any time. It’s pretty simple to introduce new transactions, so the team can expand the claims system in almost any direction.

Xchanging is also looking at Attunity Connect's support for Web services to expose legacy transactions as standard reusable services. By using the flexible and standard integration middleware, Xchanging will be able to increase its business agility. As a result, the Lloyd's Market of investors and underwriters will enjoy better returns and decreased exposure to losses. Brokers, managing agents, and member agents will appreciate expedited and expanded services. And insurees will appreciate speedy claims resolution.

Xchanging plans to use the same strategy to converge the premium-handling systems for Lloyd's and the IUA, and the company will be using Attunity Connect to achieve the same kinds of results.

Tony Burke
Systems Development Manager
Xchanging (UK)

BigCommarea utility user guide

ABSTRACT

CICS is a proven workhorse for many online applications in the mainframe world. Currently when an application has to transfer large amounts of data to another application, CICS hits a roadblock. This is due to the 32KB theoretical limit on the communication area (COMMAREA). Business Transaction Services (BTS), which were introduced in CICS TS 1.3, provided an indirect means, called data containers, that could be used to overcome this 32KB limit. They are popularly known as BigCommareas.

In order to make BigCommarea popular among the mainframe

developers' community and to help in a quick and easy implementation of proof-of-concept applications using BigCommarea, this article presents a simple utility and some sample code written in COBOL highlighting BigCommarea usage.

This document assumes a working knowledge of CICS and COBOL in the mainframe environment.

INTRODUCTION

There are many ways of transferring data between programs and transactions in CICS. Among all those techniques, COMMAREA is the most popular one. However, there is a 32KB theoretical limit on the COMMAREA size. Practically, it is good to limit the COMMAREA size to 24KB. The *CICS Application Programming Guide* specifies that, 'The length of a COMMAREA on a RETURN command can vary from transaction to transaction, up to a theoretical upper limit of 32,763 bytes. However to be safe, you should not exceed 24KB …, because of a number of factors that can reduce the limit from the theoretical maximum.'

One common method employed to overcome the limit is to issue a GETMAIN SHARED for the required amount of memory, place the data in that location, and transfer the location address and length through the COMMAREA. However, one should remember that this method creates region affinity and should be avoided. Other methods, like VSAM files, TSQs, database tables, etc, can also be used for transferring larger amounts of data across transactions and programs. However, for each special case, where the data to be passed is greater than 24KB, either a separate VSAM file or a separate TSQ needs to be created and/or the programs are forced to split the content into multiple blocks of smaller size and manipulate the data read/write operations.

In a white paper entitled 'BigCommareas: How to bypass the 32K Commarea restriction in CICS Transaction Server', Robert

Harris of IBM Hursley provides an alternative method, using BTS data containers to transfer data across transactions and programs. The paper is dotted with beautiful explanations, and covers the usage of BigCommarea in applications to the administration of BigCommareas. However, source code is given in Assembly language. In today's IT environment, we do not have many people who can understand Assembler.

The aim of this article is two-fold.

- To bring the BigCommarea concept to the vast number of COBOL users.
- To abstract usage of BTS APIs and provide a utility that makes the use of BigCommarea simpler, so that more and more users will start experimenting with this concept. This utility enables the usage of BTS containers by anyone who knows simple COBOL and CICS programming techniques. Usage of the tool is explained with a few examples.

BUSINESS TRANSACTION SERVICES (BTS) AND KEY CONCEPTS

Business Transaction Services is provided under CICS to support long-running transactions. Many objects are defined under this support service but the following are the ones of interest to us in order to explain the concept of BigCommarea. They are provided only to make this article easier to understand. For a complete description of these key words, readers are advised to refer to the *CICS Business Transaction Services* manual.

1 Repository – a VSAM file used for storing data. Depending on the FCT entry made in the system, the repository's scope may be limited to a single CICS region or it may be shareable within a CICSPlex.

2 Container – a named location in the repository where data is stored. Applications can create containers dynamically in their program and use them for passing data to other

transactions and programs. The name of a container can be up to 16 bytes. In this document, container and data container are used interchangeably.

3 Process – all containers should belong to one or other process. One process can hold many containers. The process and container name combination uniquely identifies a container within the system at any point of time. Applications can create processes dynamically in their programs and create containers within them. The name of a process can be up to 36 bytes. In this document Process is also called as a Container Manager.

4 Process type – all processes need an anchor (a process type). An RDO entry (Resource Definition Online or CICS system entry) for the process type is required. The name of a process can be up to eight bytes.

To sum up, a VSAM file is to be created and an FCT entry is to be made. That is the repository. Once a repository is created, make an RDO entry for process type. To make a process type entry, the repository file name must be mentioned along with the process type name, which creates a relationship between the process type and the repository. Repository

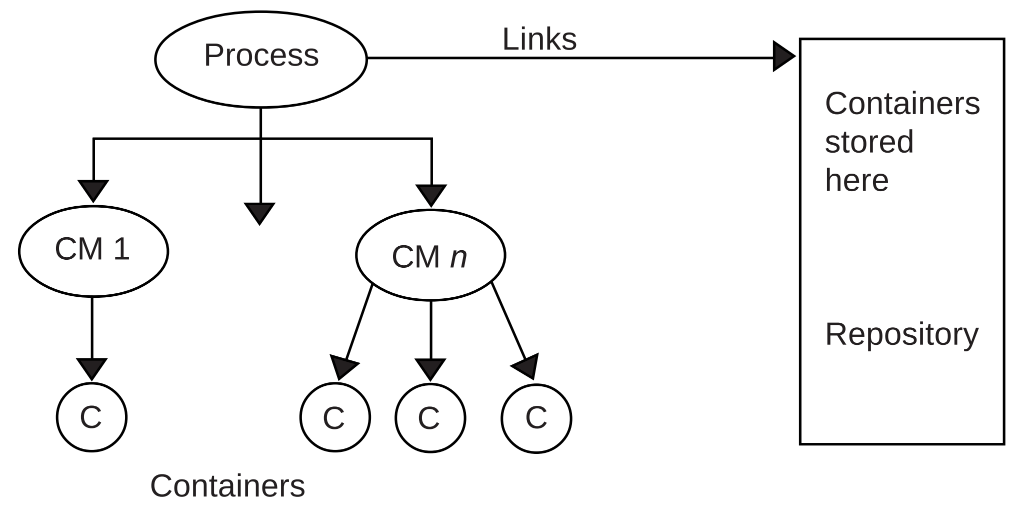

Figure 1: Process type repository relationship

creation and process type entry are one-time administrative tasks, outside the purview of the application programmer.

Assuming the repository and process type are created and available, applications can create many processes (container managers) belonging to a specific process type, and many containers belonging to a process (container manager). Applications can put data into these containers. Physically, data belonging to these containers is stored in the repository file associated with the process type to which these containers indirectly belong. Thus, the process type links processes (container managers) to a particular repository. This is illustrated in Figure 1.

Assume that application APP1 creates a container manager called CM1, which belongs to process type PTYPE1. Then it creates a container CONTAINER1 belonging to CM1, and puts some data into it. Assume that this data is to be transferred to another application, APP2, by APP1. Now, APP1 passes the names of the process type, container manager, and container to APP2, through a COMMAREA. Using the details provided by APP1, APP2 will be able to identify the length of data in the container and then read it from the container. After reading the data, APP2 may delete the container.

Note 1: if all the applications in a system are using a single process type, it may not be necessary to pass the name of the process type. It is sufficient to pass the container manager and container names along with data length.

Note 2: if data length is also passed along with other details by APP1, APP2 need not try to find out the length. It can directly read data.

ASSUMPTIONS

For the next part of the discussion let us assume:

1 The name of the VSAM file is OPERN.CICS1.BIGCOMM. This is the repository file.

2 The FCT entry for the above file is made with a logical file name of BIGCOMM. This is the entry for the repository file.

3 The RDO entry for a process type called BIGCOMM is created. This is the entry of the anchor for all processes. All applications in the system use this single process type only.

BIGCOMMAREA UTILITY

The BigCommarea utility has been created using BTS APIs. Below are the different functionalities provided by this tool:

1 Create container manager

2 Create container

3 Write data to container

4 Re-write data to container

5 Get data length of container

6 Get data from container

7 Clear data from container

8 Delete container

9 Delete container manager.

To understand how to use the above services, let us draw a parallel with databases:

1 In a DBMS, tables store data. Here, containers store data.

2 In a DBMS, a database can have multiple tables. Similarly here we create a container manager. Within a container manager, we can create many containers and store data within them.

3 In a database, we can have multiple tables and use them to store data. Similarly, we can have multiple containers in a container manager and store data in them.

The similarities end here. Now, let us see the differences:

1 In a table, we can write many records. In a container, we can write only one record.

2 In a table, we can update part of the data, by updating one or more columns. In a container, we have to re-write the entire record. It is not possible to update a part of the record written to a container.

3 In a table, we can read a part of the record, ie one or more columns. In a container, we have to read the entire record at once.

As you can see, the services provided by the utility are to create a container manager, create a container, read/write/delete/modify data in a container, delete a container, delete a container manager, and get the length of data in a container.

The usage sequence of the services for an application writing data is:

1 Create the container manager

2 Create a container

3 Write data to the container.

The usage sequence of the services for an application reading data is:

1 Get data length of the container

2 Read data from the container

3 Delete container

4 Delete container manager.

BTS COPY-BOOK STRUCTURE

The BigCommarea utility uses the following copybook. All programs that plan to use the tool need to include this copybook in their programs as part of the working storage section. (See sample programs for usage details.)

```
    05  BIG-COMM-DETAILS.
        10  BTS-PROCESS-TYPE                PIC X(08).
        10  BTS-CONTAINER-MGR-NAME          PIC X(36).
        10  BTS-CONTAINER-NAME              PIC X(16).
        10  BTS-DATA-LENGTH                 PIC S9(08) COMP.
        10  BTS-ACTION-REQUEST              PIC X(01).
            88 BTS-CREATE-CONTAINER-MGR     VALUE '0'.
            88 BTS-CREATE-CONTAINER         VALUE '1'.
            88 BTS-PUT-TO-CONTAINER         VALUE '2'.
            88 BTS-GET-FROM-CONTAINER       VALUE '3'.
            88 BTS-GET-CONTAINER-LENGTH     VALUE '4'.
            88 BTS-REWRITE-TO-CONTAINER     VALUE '5'.
            88 BTS-DELETE-CONTAINER         VALUE '6'.
            88 BTS-CLEAR-CONTAINER          VALUE '7'.
            88 BTS-DELETE-CONTAINER-MGR     VALUE '8'.
        10  BTS-RETURN-INFO.
            15  BTS-RETURN-CODE             PIC X(01).
                88 BTS-SUCCESSFUL           VALUE 'S'.
                88 BTS-FAILURE              VALUE 'F'.
            15  BTS-FAILURE-REASON-CODE     PIC S9(04) COMP.
                88 NO-FAILURE               VALUE +00.
                88 PREOCESS-TYPE-DOES-NOT-EXSIT
                                            VALUE -01.
                88 PROCESS-DOES-NOT-EXIST   VALUE -02.
                88 DATA-LENGTH-MISMATCH     VALUE -03.
                88 PROCESS-BUSY             VALUE -04.
                88 CONTAINER-DOES-NOT-EXIST VALUE -05.
                88 REPO-FILE-NOT-AVAILABLE  VALUE -06.
                88 IO-ERROR-ON-REPOSITORY-FILE
                                            VALUE -07.
                88 RECORD-IN-REPO-FILE-IN-USE
                                            VALUE -08.
                88 RECORD-LOCKED-IN-REPO-FILE
                                            VALUE -09.
                88 USER-NOT-AUTH-ON-REPO-FILE
                                            VALUE -10.
                88 REQUEST-TIMED-OUT        VALUE -11.
                88 INVALID-BTS-REQUEST      VALUE -12.
                88 UNKNOWN-BTS-REQUEST      VALUE -13.
                88 USER-NOT-AUTHORIZED      VALUE -14.
                88 PROCESS-ALREADY-IN-USE   VALUE -15.
                88 PROCESSTYPE-ALREADY-IN-USE
                                            VALUE -16.
                88 TRANSID-CAN-NOT-BE-FOUND VALUE -17.
                88 PROCESS-NOT-ENABLED      VALUE -18.
                88 OTHER-ERROR              VALUE -99.
            15  BTS-FUNCTION-N-RESP.
```

```
ØØ48ØØ              2Ø  BTS-FUNCTION            PIC X(Ø3).
ØØ49ØØ                  88  BTS-PUT             VALUE 'PUT'.
ØØ5ØØØ                  88  BTS-GET             VALUE 'GET'.
ØØ51ØØ                  88  BTS-DELETE          VALUE 'DEL'.
ØØ52ØØ                  88  BTS-DEFINE          VALUE 'DEF'.
ØØ53ØØ                  88  BTS-CANCEL          VALUE 'CAN'.
ØØ54ØØ                  88  BTS-ACQUIRE         VALUE 'ACQ'.
ØØ55ØØ              2Ø  BTS-RESP1               PIC S9(Ø8) COMP.
ØØ56ØØ              2Ø  BTS-RESP2               PIC S9(Ø8) COMP.
```

An explanation of each of the variables discussed in the copybook is given below:

- BTS-PROCESS-TYPE – this is the PROCESS TYPE as defined to the system by the system administrator. This is assumed to be BIGCOMM in the sample programs.
- BTS-CONTAINER-MGR-NAME – this is the container manager name. (In BTS terms, this is called as the process name. Once a PROCESS TYPE is defined by the system administrator, users can create any number of processes of that type within a program.)
- BTS-CONTAINER-NAME – this is the container name. Once a container manager (process) is created, any number of containers can be created under the same name.
- BTS-DATA-LENGTH – this specifies the length of data to be written/re-written/read from the container.
- BTS-ACTION-REQUEST – this specifies the type of request. The requests can be to create a container manager, delete a container manager, create a container, delete a container, write to a container, read from a container, re-write to a container, get the length of data in a container, or clear the contents of a container.
- BTS-RETURN-CODE – this specifies whether the requested operation succeeded (BTS-SUCCESSFUL) or failed (BTS-FAILED).
- BTS-FAILURE-REASON-CODE – if the requested operation succeeded (BTS-SUCCESSFUL), this contains

00 (BTS-NOFAILURE). If the requested operation failed (BTS-FAILED), it has different values, depending on the error. The most important of them are:

- PROCESS-TYPE-DOES-NOT-EXSIT – the system administrator did not make entries for this process type or you are using the wrong process type.
- PROCESS-DOES-NOT-EXIST – the container manager under which you are trying to create containers was not created.
- DATA-LENGTH-MISMATCH – you are trying to read contents from a container and the length does not match.
- CONTAINER-DOES-NOT-EXIST – you are trying to read, write, or re-write to a container and it does not exist.

The error conditions returned can be used by programs to handle errors in their programs, while using the tool. Apart from that, the tool provides additional information that can be used by advanced applications trying to use the tool to get actual operations performed and the actual RESP codes returned by CICS. These are returned by the BTS-FUNCTION-N-RESP variable.

USING THE BIGCOMMAREA UTILITY

BTSUTIL1 is the tool program that provides all the BigCommarea services.

To use the BigCommarea utility services, create a container manager and issue a commit for the unit of work. Once a container manager is created, it will be there, even after system shutdown, until it is deleted by a program or by the system administrator. It is not necessary to have multiple container managers. Create a single container manager per terminal at sign-on and delete it when the terminal session is closed. This will keep maintenance work to a minimum.

BTSSAMP1 is an example program that shows how to create a container manager. (Note: these examples assume that all the necessary entries are made by the system administrator and a process-type called BIGCOMM is created and available for use. To check the existence of the process type, use the CBAM transaction.) If successful, BTSSAMP1 creates a container manager with the name PROCESS1.

Once a container manager is created, any number of containers can be created in it. BTSSAMP2 shows how to create a container, write data into it, read data from it, and delete it. Once a commit is issued, even if the user logs out of the system, data is retained by the system. Hence the BigCommarea utility can also be used as a persistent COMMAREA.

All the data should be written into a container in one go. It is not possible to write part of the data and append the next part later. Although it is possible to modify the data in a container, the new data will replace all the existing data.

To read data from a container, the exact length of the data should be known, and a memory structure to hold the data should be available.

BTSSAMP3 shows how to delete a container manager. It is best to keep this as a separate unit of work.

CALL STRUCTURE

The BigCommarea utility uses the COBOL Dynamic CALL mechanism. (Note: it is not possible to use LINK or XCTL because they also have the same 32KB limitation.) The call structure used is as follows:

```
CALL BTS-PROG  USING DFHEIBLK,
                     DFHCOMMAREA,
                     BTS-COPY-BOOK,
                     BTS-DATA-AREA
```

Note: DFHCOMMAREA is not manipulated by the utility.

Sl No	*Operation*	*Input*	*Output*
1	Create container manager	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) Set BTS-ACTION-REQUEST to BTS-CREATE-CONTAINER-MGR	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP
2	Delete container manager	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) Set BTS-ACTION-REQUEST to BTS-DELETE-CONTAINER-MGR	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP
3	Create container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) Set BTS-ACTION-REQUEST to BTS-CREATE-CONTAINER	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP
4	Write data to container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) BTS-DATA-LENGTH (e) Set BTS-ACTION-REQUEST to BTS-PUT-TO-CONTAINER (f) Variable with container data (BTS-DATA-AREA)	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP

Figure 1: Details of input/output for the services (continues)

Note: in some cases, BTS-DATA-AREA (the variable assumed to be containing data to be written/read from container) has no meaning. For example, when creating a container manager, there is no meaning for a data area. In such cases, use a

Sl No	Operation	Input	Output
5	Re-write data to container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) BTS-DATA-LENGTH (e) Set BTS-ACTION-REQUEST to BTS-REWRITE-TO-CONTAINER (f) Variable with container data (BTS-DATA-AREA)	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP
6	Clear data from container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) Set BTS-ACTION-REQUEST to BTS-CLEAR-CONTAINER	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP
7	Delete container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) Set BTS-ACTION-REQUEST to BTS-DELETE-CONTAINER	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP

Figure 1: Details of input/output for the services (continues)

dummy variable with a length greater than 1. However, it will not be used by the utility.

USAGE

Figure 1 shows details of input/output for the services.

Sl No	***Operation***	***Input***	***Output***
8	Get data from container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) BTS-DATA-LENGTH (e) Set BTS-ACTION-REQUEST to BTS-GET-FROM-CONTAINER (f) Variable with container data (BTS-DATA-AREA). Length of this variable should be greater than or equal to the length specified above.	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP (d) Data through BTS-DATA-AREA
9	Get data length of container	(a) BTS-PROCESS-TYPE (b) BTS-CONTAINER-MGR-NAME (c) BTS-CONTAINER-NAME (d) Set BTS-ACTION-REQUEST to BTS-GET-CONTAINER-LENGTH	(a) BTS-RETURN-CODE (b) BTS-FAILURE-REASON-CODE (c) BTS-FUNCTION-N-RESP (d) Data length returned through BTS-DATA-LENGTH variable

Figure 1: Details of input/output for the services (continued)

BTS UTILITY

```
000010 CBL XOPTS(SP,NOEDF)
000100 IDENTIFICATION DIVISION.
000200 PROGRAM-ID. BTSUTIL1.
000300 AUTHOR. SARMAKVRS.
000400 ENVIRONMENT DIVISION.
000500 DATA DIVISION.
000600 WORKING-STORAGE SECTION.
000610 01  BTS-FILE-NAME                  PIC X(08) VALUE SPACES.
000620 01  FILE-OPEN-STATUS               PIC S9(08) COMP VALUE +0.
000700 LINKAGE SECTION.
000800 01  DFHCOMMAREA                    PIC X(01).
000900 01  BTS-COPY-BOOK.
```

```
001000     COPY BTSCOPY1.
001100*
001200 01  BTS-DATA-BUFFER                PIC X(01).
001300*
001400 PROCEDURE  DIVISION USING BTS-COPY-BOOK, BTS-DATA-BUFFER.
001500 A000-MAIN-PARA.
001600     SET BTS-SUCCESSFUL    TO  TRUE
001700     SET NO-FAILURE        TO  TRUE
001800     MOVE SPACES           TO  BTS-FUNCTION
001900     MOVE +0               TO  BTS-RESP1
002000     MOVE +0               TO  BTS-RESP2
002100
002200     EVALUATE TRUE
002300         WHEN  BTS-PUT-TO-CONTAINER
002400             OR BTS-REWRITE-TO-CONTAINER
002500             PERFORM A100-WRITE-TO-CONTAINER
002600
002700         WHEN  BTS-CREATE-CONTAINER
002800             OR BTS-CLEAR-CONTAINER
002900             MOVE +0 TO BTS-DATA-LENGTH
003000             PERFORM A100-WRITE-TO-CONTAINER
003100
003200         WHEN  BTS-GET-FROM-CONTAINER
003300             PERFORM A200-READ-FROM-CONTAINER
003400
003500         WHEN  BTS-GET-CONTAINER-LENGTH
003600             MOVE +0 TO BTS-DATA-LENGTH
003700             PERFORM A200-READ-FROM-CONTAINER
003800
003900         WHEN  BTS-DELETE-CONTAINER
004000             PERFORM A300-DELETE-CONTAINER
004100
004200         WHEN  BTS-CREATE-CONTAINER-MGR
004300             PERFORM A400-BTS-CREATE-PROCESS
004400
004500         WHEN  BTS-DELETE-CONTAINER-MGR
004600             PERFORM A400-BTS-DELETE-PROCESS
004700
004800         WHEN  OTHER
004900             SET BTS-FAILURE TO TRUE
005000             SET UNKNOWN-BTS-REQUEST TO TRUE
005100*
005200     END-EVALUATE
005300     GOBACK
005400     .
005500
005600
005700* INPUT - PROCESS-TYPE, PROCESS, CONTAINER, DATA-BUFFER, DATA-LENG
005800* OUTPUT - ERROR CODES
005900 A100-WRITE-TO-CONTAINER.
```

```
     PERFORM B000-ACQUIRE-PROCESS
     IF BTS-SUCCESSFUL
         SET BTS-PUT TO TRUE
         EXEC CICS PUT
             CONTAINER(BTS-CONTAINER-NAME)
             ACQPROCESS
             FROM(BTS-DATA-BUFFER)
             FLENGTH(BTS-DATA-LENGTH)
             RESP(BTS-RESP1)
             RESP2(BTS-RESP2)
         END-EXEC
         IF BTS-RESP1 = DFHRESP(NORMAL)
             CONTINUE
         ELSE
             SET BTS-FAILURE TO TRUE
             SET OTHER-ERROR TO TRUE

             IF BTS-RESP1 = DFHRESP(CONTAINERERR)
                 SET CONTAINER-DOES-NOT-EXIST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(INVREQ)
                 SET INVALID-BTS-REQUEST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(IOERR)
                 IF BTS-RESP2 = 30
                     SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
                 END-IF
                 IF BTS-RESP2 = 31
                     SET RECORD-IN-REPO-FILE-IN-USE TO TRUE
                 END-IF
             END-IF

             IF BTS-RESP1 = DFHRESP(LOCKED)
                 SET RECORD-LOCKED-IN-REPO-FILE TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(PROCESSBUSY)
                 SET PROCESS-BUSY TO TRUE
             END-IF
         END-IF
     END-IF
     .

* INPUT - PROCESS-TYPE, PROCESS, CONTAINER, DATA-BUFFER, DATA-LENG
* OUTPUT - DATA AND ERROR CODES
 A200-READ-FROM-CONTAINER.
     PERFORM B000-ACQUIRE-PROCESS
     IF BTS-SUCCESSFUL
```

```
         SET BTS-GET TO TRUE
         EXEC CICS GET
             CONTAINER(BTS-CONTAINER-NAME)
             ACQPROCESS
             INTO(BTS-DATA-BUFFER)
             FLENGTH(BTS-DATA-LENGTH)
             RESP(BTS-RESP1)
             RESP2(BTS-RESP2)
         END-EXEC
         IF BTS-RESP1 = DFHRESP(NORMAL)
             CONTINUE
         ELSE
             SET BTS-FAILURE TO TRUE
             SET OTHER-ERROR TO TRUE

             IF BTS-RESP1 = DFHRESP(CONTAINERERR)
                 SET CONTAINER-DOES-NOT-EXIST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(INVREQ)
                 SET INVALID-BTS-REQUEST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(IOERR)
                 IF BTS-RESP2 = 30
                     SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
                 END-IF
                 IF BTS-RESP2 = 31
                     SET RECORD-IN-REPO-FILE-IN-USE TO TRUE
                 END-IF
             END-IF

             IF BTS-RESP1 = DFHRESP(LOCKED)
                 SET RECORD-LOCKED-IN-REPO-FILE TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(PROCESSBUSY)
                 SET PROCESS-BUSY TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(LENGERR)
                 SET DATA-LENGTH-MISMATCH TO TRUE
             END-IF
         END-IF
     END-IF
     .

* INPUT - PROCESS-TYPE, PROCESS, CONTAINER
* OUTPUT - ERROR CODES
 A300-DELETE-CONTAINER.
```

```
     PERFORM BØØØ-ACQUIRE-PROCESS
     IF BTS-SUCCESSFUL
         SET BTS-DELETE TO TRUE
         EXEC CICS DELETE
             CONTAINER(BTS-CONTAINER-NAME)
             ACQPROCESS
             RESP(BTS-RESP1)
             RESP2(BTS-RESP2)
         END-EXEC
         IF BTS-RESP1 = DFHRESP(NORMAL)
             CONTINUE
         ELSE
             SET BTS-FAILURE TO TRUE
             SET OTHER-ERROR TO TRUE

             IF BTS-RESP1 = DFHRESP(CONTAINERERR)
                 SET CONTAINER-DOES-NOT-EXIST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(INVREQ)
                 SET INVALID-BTS-REQUEST TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(IOERR)
                 IF BTS-RESP2 = 3Ø
                     SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
                 END-IF
                 IF BTS-RESP2 = 31
                     SET RECORD-IN-REPO-FILE-IN-USE TO TRUE
                 END-IF
             END-IF

             IF BTS-RESP1 = DFHRESP(LOCKED)
                 SET RECORD-LOCKED-IN-REPO-FILE TO TRUE
             END-IF

             IF BTS-RESP1 = DFHRESP(PROCESSBUSY)
                 SET PROCESS-BUSY TO TRUE
             END-IF
         END-IF
     END-IF
     .

* INPUT - PROCESS-TYPE, PROCESS
* OUTPUT - ERROR CODES
 A4ØØ-BTS-CREATE-PROCESS.
     PERFORM B1ØØ-CHECK-FILE-STATUS
     IF  BTS-SUCCESSFUL
         SET BTS-DEFINE TO TRUE
         EXEC CICS DEFINE
```

```
               PROCESS(BTS-CONTAINER-MGR-NAME)
               PROCESSTYPE(BTS-PROCESS-TYPE)
               TRANSID(EIBTRNID)
               RESP(BTS-RESP1)
               RESP2(BTS-RESP2)
           END-EXEC
           IF BTS-RESP1 = DFHRESP(NORMAL)
               CONTINUE
           ELSE
               SET BTS-FAILURE TO TRUE
               SET OTHER-ERROR TO TRUE

               IF BTS-RESP1 = DFHRESP(INVREQ)
                   IF BTS-RESP2 = 12
                       SET PROCESS-NOT-ENABLED TO TRUE
                   END-IF
                   IF BTS-RESP2 = 22
                       SET INVALID-BTS-REQUEST TO TRUE
                   END-IF
               END-IF

               IF BTS-RESP1 = DFHRESP(IOERR)
                   IF BTS-RESP2 = 30
                       SET IO-ERROR-ON-REPOSITORY-FILE TO TRUE
                   END-IF
                   IF BTS-RESP2 = 31
                       SET RECORD-IN-REPO-FILE-IN-USE TO TRUE
                   END-IF
               END-IF

               IF BTS-RESP1 = DFHRESP(NOTAUTH)
                   IF BTS-RESP2 = 101
                       SET USER-NOT-AUTH-ON-REPO-FILE TO TRUE
                   END-IF
                   IF BTS-RESP2 = 102
                       SET USER-NOT-AUTHORIZED TO TRUE
                   END-IF
               END-IF

               IF BTS-RESP1 = DFHRESP(PROCESSERR)
                   IF BTS-RESP2 = 02
                       SET PROCESS-ALREADY-IN-USE TO TRUE
                   END-IF
                   IF BTS-RESP2 = 09
                       OR BTS-RESP2 = 16
                       SET PREOCESS-TYPE-DOES-NOT-EXSIT TO TRUE
                   END-IF
               END-IF

               IF BTS-RESP1 = DFHRESP(TRANSIDERR)
```

```
025800                 SET TRANSID-CAN-NOT-BE-FOUND TO TRUE
025900             END-IF
026000          END-IF
026010     END-IF
026100     .
026200
026300
026400* INPUT - PROCESS-TYPE, PROCESS
026500* OUTPUT - ERROR CODES
026600  A400-BTS-DELETE-PROCESS.
026700      PERFORM B000-ACQUIRE-PROCESS
026800      IF  BTS-SUCCESSFUL
026900          SET BTS-CANCEL TO TRUE
027000          EXEC CICS CANCEL
027100              ACQPROCESS
027200              RESP(BTS-RESP1)
027300              RESP2(BTS-RESP2)
027400          END-EXEC
027500          IF BTS-RESP1 = DFHRESP(NORMAL)
027600              CONTINUE
027700          ELSE
027800              SET BTS-FAILURE TO TRUE
027900              SET OTHER-ERROR TO TRUE
028000
028100              IF BTS-RESP1 = DFHRESP(INVREQ)
028200                  SET INVALID-BTS-REQUEST TO TRUE
028300              END-IF
```

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

K V R S Sarma, K Viswanathan
Technical Architect, System Programmer
Infosys Technologies Ltd (India)

It was 20 years ago today...

It was December 1985 that *CICS Update* was first published by Xephon. It was produced on an Apple II running Zardax, and was warmly welcomed by the CICS community as a way of sharing information and programs.

In those days, the articles tended to be quite short. That first issue had 18 articles plus 'CICS news' in its 44 pages. There was a forward-looking piece on CICS 1.7, and one about monitoring using Omegamon. It also contained three pages of CICS programming guidelines as well as VSE-related articles – one on PL/I storage and one looking at VSE BUFSIZE problems.

So far, 240 issues of *CICS Update* have been produced, and there has been a huge number of articles published on a wide range of CICS-related topics.

The first editor was Steve Piggott, who edited it for a number of years up to his untimely death. Robert Burgess then took over the editorship, and now the position is held by Trevor Eddolls.

The idea behind the *Update* publications was Chris Bunyan's, then Managing Director of Xephon. *CICS Update* was an immediate success and was followed soon afterwards by *VM Update* and *MVS Update*. As well as *CICS Update*, the *Update* family currently boasts *AIX Update*, *DB2 Update*, *MVS Update*, *MQ Update*, *RACF Update*, and *TCP/SNA Update*.

The real reason for the success of this publication has been the willingness of CICS users and programmers to share their ideas and, more importantly, their code and REXX EXECs, etc. This has not only given others a short-cut to implementing the same solutions at their sites, but has given people ideas about what is possible and alternative ways to make them happen. I would like to take this opportunity to thank the hundreds of people who have contributed so far to *CICS Update*.

If you have an idea for an article, you can download a copy of our *Notes for Contributors* from www.xephon.com/nfc. Articles can be sent to Trevor Eddolls at trevore@xephon.com.

December 2002 – November 2005 index

Items below are references to articles that have appeared in *CICS Update* since issue 205, December 2002. References show the issue number followed by the page number(s). Subscribers can download copies of all issues in Acrobat PDF format from Xephon’s Web site.

IBM has announced a new Tivoli branded Composite Application Management (ITCAM) family of solutions. These are based on its Cyanea acquisition, which had a tool for monitoring composite applications.

The ITCAM suite will mediate and monitor Web services, response time tracking, and problem tracking for WebSphere Application Server and Business Integration, CICS transaction management, MQ messaging, and IMS databases. It integrates data from a number of IBM acquisitions, including Cyanea, Candle, and Rational.

The products will track performance and response of composite applications. The first one is the Tivoli Enterprise Portal, which provides different views (workspaces) for system administrators, database administrators, and application development teams. Using Tivoli's Change Configuration Management (CCM) database as the primary source, the portals can present information from the Omegamon monitoring tools for WebSphere and MQSeries, Rational code profiling, and Tivoli event consoles. The portal provides the front end for all the data delivered by the ITCAM products.

For further information contact:
URL: www-306.ibm.com/software/tivoli/services/consulting/offerings/offers-composite-app-mgmt.html.

* * *

Intensity Software has announced Version 2.0 of KICKS for .NET, its CICS migration tool that was formerly known as NetKicks. New features include support for other COBOL compilers such as Micro Focus Net Express with .NET in addition to NetCOBOL for .NET. There's additional language support so the product now supports Assembler, PL/I, and C.

In addition to the language and platform flexibility, KICKS for .NET includes more complete support of CICS features, for example there is now support for extended attributes in the product. Screen information stored within extended attributes can now be utilized to generate a more accurate representation of the mainframe screens. Field-level validation specified within extended attributes is now automatically converted into ASP.NET validation controls during the automated translation of the mainframe screens into ASP.NET screens.

For further information contact:
URL: www.intensitysoftware.com/Products/KICKSforNET8482/KICKSforNETSolutionDetails/tabid/67/Default.aspx.

* * *

ClearNova has announced the release of ThinkCAP JX, a rapid application development (RAD) platform for building Internet applications that combines the power of AJAX, J2EE, and open source to help organizations build intuitive and responsive Web-based applications.

ThinkCAP JX Enterprise allows developers to visually build Web services to more than 60 different enterprise platforms including CICS, SAP, PeopleSoft, Siebel, Oracle Applications, Salesforce.com, Tibco, and Lotus Notes/Domino.

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