221

CICS update

April 2004

In this issue

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

Nicole Thomas
E-mail: nicole@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.

Printed in England.

Understanding the Open Transaction Environment

The Open Transaction Environment (OTE) is being exploited in several different ways with the latest release of CICS. OTE provides a rich environment for programming certain types of CICS application, but this can lead to some confusion about what is taking place within CICS to support the new functionality, and how to interpret the system's behaviour.

This article describes the use of OTE within IBM CICS Transaction Server 2.2, and gives examples of what activity may be seen within CICS traces for various OTE-managed operations taking place in the system.

WHAT IS OTE?

To understand the rationale for OTE support, it is necessary to look back at the traditional way in which workloads were processed within CICS. The CICS quasi-reentrant (QR) TCB is the primary dispatching tool for executing different CICS tasks, and the CICS dispatcher function makes use of very rapid multitasking techniques to give the various tasks within the system their own opportunity to be run under the QR TCB for short periods of time. This approach is efficient in terms of rapid throughput, but there are some disadvantages to it. For example, making use of a single TCB to process the vast majority of the CICS workload means that CICS is not able to exploit hardware with multiple central processors in a concurrent manner, since at any one time the QR TCB may be executed on only one processor by the operating system.

OTE allows certain types of program environments to execute under a TCB separate from the QR TCB. Such OTE-managed TCBs are known as open TCBs. They are dispatched separately by the MVS dispatcher and can therefore execute at the same time as the QR TCB, being dispatched under parallel central processors concurrently by the operating system.

WHAT ADVANTAGES DOES OTE PROVIDE?

Parallel processing allows better exploitation of multi-engine hardware, and can therefore improve throughput of CICS workloads as a result. Providing dedicated TCBs for programs to exploit allows specific environments to be constructed for programs to execute in; these could not be used within the traditional QR TCB model; for example, support for interpretation of Java classes under JVMs executing within CICS. Also, use of those MVS services that would suspend a TCB ('block' it) can be limited to a single application running under its own dedicated open TCB, and so not have a detrimental effect on the other programs running within CICS.

WHEN WAS OTE INTRODUCED?

OTE was first introduced with IBM CICS Transaction Server 1.3. The initial implementation provided OTE support for JVM environments, to interpret Java class files of bytecodes.

OTE support within CICS Transaction Server 1.3 was later enhanced via CICS PTF UQ44003 to support Java 'hot-pooling', for compiled Java applications generated using the Enterprise Toolkit Compiler and Binder. This type of Java program environment is also referred to as the High-Performance Java (HPJ) compiler and run-time. (Such HPJ-compiled Java applications are known as Java program objects, to differentiate them from interpreted Java class files that run in a JVM environment.)

Support for OTE has now been extended by CICS Transaction Server 2.2 to provide open TCBs to support OPENAPI Task-Related User Exit programs. DB2 can now exploit such an OPENAPI TRUE, so that OTE-managed TCBs can be used to process DB2 requests from CICS applications.

CICS support for OTE has therefore been an evolutionary process, with an increasing variety of programming environments able to exploit it.

WHAT OPEN TCB MODES EXIST UNDER OTE?

Three OTE TCB modes are currently supported by CICS. These are:

- J8 TCBs, used by CICS to provide an environment for JVMs to execute under and interpret Java applications.
- H8 TCBs, used by CICS to provide an environment for Java hot-pooling support. Note: CICS Transaction Server 2.2 retains support for Java hot-pooling, for migration purposes. The strategic platform for Java exploitation within CICS is via the JVM.
- L8 TCBs, used by CICS to provide an environment for OPENAPI-capable TRUEs such as DB2 V6 and above.

CICS manages the switching of TCBs between these various modes itself. Users cannot define their own types of open TCB for CICS to use. The different programming environments that exploit the three types of OTE TCB are all very specific, and each has its own characteristics and functional requirements.

The switching between different OTE TCBs is governed by the type of program environment being exploited, and the threadsafety of the programs being executed. To be able to understand what TCBs will be used by CICS for which types of operation, it is therefore necessary to understand the concept of threadsafety, both at an application level and a CICS command level.

WHAT IS THREADSAFETY?

A definition of threadsafety is the ability to ensure that any shared resource or internal state data is accessed in a serialized manner. This means that threadsafe programs do not mind what TCB they are executing under because they are not at risk from other concurrently executing programs running under different TCBs.

The *CICS Transaction Server 2.2 Application Programming Guide* states the following:

In the CICS Open Transaction Environment (OTE), threadsafe application programs and open task-related user exits, global user exit programs, and user-replaceable modules cannot rely on quasi-reentrancy, because they can run concurrently on an open TCB. Furthermore, even quasi-reentrant programs are at risk if they access resources that can also be accessed by a user task running concurrently under an open TCB. This means that the techniques used by user programs 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 are described as threadsafe. (The term fully re-entrant is also used sometimes, but this can be misunderstood, hence threadsafe is the preferred term.) For most resources, such as files, transient data queues, temporary storage queues, and DB2 tables, CICS processing automatically ensures access in a threadsafe manner. However, for any other resources, such as shared storage, which are accessed directly by user programs, it is the responsibility of the user program to ensure threadsafe processing. Typical examples of shared storage are the CICS CWA, global user exit global work areas, and storage acquired explicitly by the application program with the shared option.

From these definitions, it can be seen that an ability to recognize and understand threadsafety within CICS applications is vital if a system is to exploit OTE in an effective and optimal manner. There are two aspects of threadsafety to be considered – the threadsafety of CICS applications, and the threadsafety of CICS functions themselves.

APPLICATION THREADSAFETY

When defining programs to CICS, the CONCURRENCY attribute is used to inform CICS whether a program is threadsafe or not. The options are QUASIRENT and THREADSAFE. The default is QUASIRENT. Quasi-reentrant programs need to execute under the QR TCB.

Conversely, a threadsafe program does not require the

serialization provided by virtue of quasi-reentrant dispatching under the QR TCB, and so is free to execute upon other TCBs if moved there by CICS.

The threadsafety of a program is specific to its own internal function (ie its own business 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 access to global storage areas. To assist in locating instances of these commands, a sample command table (DFHEIDTH) is provided for use with the CICS load module scanner utility program DFHEISUP. The load module scanner is a batch utility program. It scans load modules within libraries and locates the EXEC CICS commands present within them. It then applies the appropriate filter that is provided to it, in order to identify only those commands that the user is interested in. The load module scanner produces one of two types of report. This is either a summary report listing the modules containing the commands specified by the filter together with the number of specified commands in each module, or a detailed report with a list for each module showing the specified commands that it contains and their offsets.

By using DFHEIDTH, DFHEISUP can therefore identify those programs that issue commands EXEC CICS ADDRESS CWA, EXEC CICS EXTRACT EXIT, or EXEC CICS GETMAIN SHARED. To ensure such programs are indeed threadsafe, they have to include the necessary synchronization logic to guard against concurrent updates to the shared storage by another program (or programs) executing at the same time under another TCB.

Another reason for a program to be non-threadsafe is if it is a non-reentrant program that modifies its own program storage dynamically. Such programs are viable when executing in a serialized manner under the QR TCB, provided that they ensure their state is consistent once more before issuing another EXEC CICS command.

Such non-reentrancy is not because of the exploitation of

specific EXEC CICS commands that provide access to shared data, but rather because of the way in which the program has been written to modify itself dynamically. It is therefore not detectable by using tools such as the load module scanner.

It is possible that some programs defined as threadsafe may not in fact be capable of running under an open TCB for some reason. If problems occur when running programs that were believed to be threadsafe, it is possible that the CICS system may need to be temporarily modified to force all applications to execute under the QR TCB until the problem can be investigated and resolved. To help achieve this without the overhead of modifying application program RDO definitions, CICS provides a SIT parameter FORCEQR. The default is FORCEQR=NO, meaning CICS will honour the CONCURRENCY attribute in the program definitions. Setting FORCEQR=YES in the SIT allows this attribute to be overridden. This is perceived as being useful when investigating problems in a production CICS environment.

CICS THREADSAFETY

Some CICS commands are capable of being executed under TCBs other than the QR TCB. An example is the EXEC CICS ASSIGN command. These are therefore threadsafe CICS commands. Other CICS commands require the use of the QR TCB for their own serialization purposes, and so are not threadsafe. An example of such a non-threadsafe CICS command is the EXEC CICS WRITE command to File Control. CICS will automatically switch to the QR TCB when such a non-threadsafe CICS command is being executed. This is handled by DFHEIP, the CICS EXEC interface program.

It is possible for an application program to be threadsafe itself, and yet issue non-threadsafe EXEC CICS commands. Therefore, a program does not have only to issue threadsafe CICS commands in order to be threadsafe. It is the threadsafety of the program itself (not the commands to CICS that it issues) that determines how its CONCURRENCY attribute should be specified

on its program definition. Specifically, does it access shared storage, or does it modify its own internal state by being non-reentrant?

EXAMPLES OF OTE TCB ACTIVITY IN CICS TRACES

The following example traces are provided to give some guidance on what to expect when OTE functions are utilized within CICS. Note that the traces have been edited for clarity, and that a considerable number of irrelevant trace events have therefore been omitted.

A NON-THREADSAFE PROGRAM INVOKING DB2

The following trace shows an example of a CICS Transaction Server 2.2 application invoking DB2 via an EXEC SQL SELECT command. The application is not defined to be threadsafe (ie it is quasi-reentrant, and defined as CONCURRENCY = QUASIRENT).

CICS switches TCBs from the QR TCB to an OTE-managed L8 TCB (L8000 in this example) because DB2 V6 and above is an OPENAPI-capable TRUE. When control returns to CICS from DB2, the CICS External Resource Manager, DFHERM, switches the TCB mode back to the QR TCB because the application is not threadsafe and so requires a single-threaded quasi-reentrant environment. The subsequent EXEC CICS SEND command is executed under the QR TCB.

```
00031 QR    AP 2520 ERM   ENTRY COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00031 QR    DS 0002 DSAT  ENTRY CHANGE_MODE           L8
00031 L8000 DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00031 L8000 AP 3180 D2EX1 ENTRY APPLICATION           REQUEST EXEC SQL
SELECT
00031 L8000 AP 3181 D2EX1 EXIT  APPLICATION-REQUEST

00031 L8000 DS 0002 DSAT  ENTRY CHANGE_MODE           00000001
00031 QR    DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00031 QR    AP 2521 ERM   EXIT  COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00031 QR    AP 00E1 EIP   ENTRY SEND-TC               0004,151084E0

00031 QR    AP FD01 ZARQ  ENTRY APPL_REQ
```

```
15082270,ERASE,WRITE
00031 QR    AP FD81 ZARQ  EXIT  APPL_REQ
00031 QR    AP 00E1 EIP   EXIT  SEND-TC                OK
```

The application would remain under the QR TCB until it invoked a service that required a different TCB environment. This could be another EXEC SQL request to DB2, where CICS would once again switch TCBs to the L8 TCB for the duration of the request. Alternatively, it could be an EXEC CICS LINK to a JVM program, when CICS would switch the thread of execution to run under a J8 open TCB.

A THREADSAFE PROGRAM INVOKING DB2

The following trace shows an example of another CICS Transaction Server 2.2 application invoking DB2 V6. This application is defined to be threadsafe. CICS switches TCBs from the QR TCB to an OTE-managed L8 TCB (L8001 in this example). When control returns to CICS from DB2, the thread of execution remains on this open TCB, since the application is defined to be threadsafe and is therefore capable of executing its code in a truly parallel environment to the QR TCB's execution. The application performs its own internal logic, then issues an EXEC CICS WRITEQ command to temporary storage. This happens to be a threadsafe CICS command, and so it is processed under the L8 TCB as well.

```
00054 QR    AP 2520 ERM   ENTRY COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00054 QR    DS 0002 DSAT  ENTRY CHANGE_MODE           0000000A
00054 L8001 DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00054 L8001 AP 3180 D2EX1 ENTRY APPLICATION           REQUEST EXEC SQL
SELECT
00054 L8001 AP 3181 D2EX1 EXIT  APPLICATION-REQUEST

00054 L8001 AP 2521 ERM   EXIT  COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00054 L8001 AP 00E1 EIP   ENTRY WRITEQ-TS             0004,151084D0

00054 L8001 TS 0201 TSQR  ENTRY WRITE                 ANDYTEST,15109978

00054 L8001 TS 0901 TSAM  ENTRY WRITE_AUX_DATA        15109978

00054 L8001 TS 0902 TSAM  EXIT  WRITE_AUX_DATA/OK     1,00000001
00054 L8001 TS 0202 TSQR  EXIT  WRITE/OK              1
```

```
00054 L8001 AP 00E1 EIP   EXIT  WRITEQ-TS            OK

00054 L8001 AP 00E1 EIP   ENTRY SEND-TC              0004,151084D0

00054 L8001 DS 0002 DSAT  ENTRY CHANGE_MODE          QR
00054 QR    DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00054 QR    AP FD01 ZARQ  ENTRY APPL_REQ
15082270,ERASE,WRITE
00054 QR    AP FD81 ZARQ  EXIT  APPL_REQ
00054 QR    AP 00E1 EIP   EXIT  SEND-TC              OK
```

There is no switch back to the QR TCB required to honour this threadsafe CICS command. Once again, control then returns to the application, and it performs its own internal logic. When it next issues a CICS command, it is an EXEC CICS SEND. This is not a threadsafe CICS command, and so CICS switches TCBs from the L8 TCB to the QR TCB in order to process the request. On completion, CICS returns control to the application under the QR TCB. This is since the application has now invoked a non-threadsafe CICS function, and so there is the possibility that it may do so again. In order to avoid the processing overhead of unnecessary TCB switching, it is considered prudent to remain running under the QR TCB at this stage. If the application issues a command that does require a TCB switch (another EXEC SQL call to DB2 for example) then CICS will switch back to the L8 TCB once more. On completion of this request, control will once again remain under the L8 TCB until another non-threadsafe CICS command is issued.

These previous two examples have shown how CICS automatically switches TCB modes in order to process requests for specific functions that require them. For OPENAPI-capable TRUEs, CICS will exploit L8 TCBs to execute the RMI requests. If such requests are issued from non-threadsafe applications, CICS will switch back to the QR TCB on completion of the requests. If they are issued from threadsafe applications, CICS will return control to the application under the open TCB, and allow it to remain executing under this TCB until it makes use of a non-threadsafe function. CICS then reverts to using the QR TCB.

A JVM PROGRAM INVOKING A THREADSAFE CICS FUNCTION

The following trace shows an example of a Java class being interpreted by a JVM executing within CICS Transaction Server 2.2. The JVM environment requires its own OTE-managed open TCB, and so CICS allocates such a TCB to the task for the JVM's use during program initialization. This means that the task will execute under a J8 TCB (J8001 in this example), while interpreting the Java program's bytecodes. It will also remain on the J8 TCB when use is made of certain threadsafe CICS functions via the JCICS API classes. This means that the flow of control will remain under this open TCB and not be switched back to execute on the QR TCB. EXEC CICS ASSIGN is an example of such a threadsafe CICS command.

```
00042 J8001 AP 1802 JRAS  ENTRY com.ibm.cics.server.Wrapper
callUserClass

00042 J8001 AP 21E0 JCICS ENTRY DTCTask_getCommonData

00042 J8001 AP 00E1 EIP   ENTRY ASSIGN

00042 J8001 AP 00E1 EIP   EXIT  ASSIGN                OK

00042 J8001 AP 21E0 JCICS ENTRY DTCSupport_MakeJavaString TD

00042 J8001 AP 21E0 JCICS EXIT  DTCSupport_MakeJavaString

00042 J8001 AP 00E1 EIP   ENTRY ASSIGN

00042 J8001 AP 00E1 EIP   EXIT  ASSIGN                OK

00042 J8001 AP 21E0 JCICS EXIT  DTCTask_getCommonData
```

Note that CICS is constructing EXEC CICS commands dynamically as part of the processing of the JCICS method calls. These appear as EIP ENTRY and EXIT trace points in exactly the same manner as those commands hard-coded within traditional CICS application programs.

A JVM PROGRAM INVOKING A NON-THREADSAFE CICS FUNCTION

The following trace shows an example of a different Java class

being interpreted by a JVM executing within CICS Transaction Server 2.2. As before, the JVM environment requires its own open TCB, and so CICS allocates one (J8002 in this case). The application executes under the J8 TCB until it makes use of a JCICS function that invokes non-threadsafe CICS code (in this case, an EXEC CICS SEND command). When DFHEIP encounters this command, it switches the TCB environment back to the QR TCB in order to execute it. On completion of the EXEC CICS SEND command, CICS switches TCBs back from the QR to the J8 TCB, before returning the thread of execution back to the Java application program. The JVM environment requires its own open TCB, so CICS has to switch back from the QR TCB after the non-threadsafe command has completed.

This behaviour is different from a threadsafe application running under an L8 TCB (after a call to DB2), which has then issued a non-threadsafe command. Remember that in such a case, CICS will leave the application running under the QR TCB after the command has completed (to try to avoid the potential for unnecessary TCB switches in the future). This approach cannot be taken with JVMs because an open TCB environment is mandatory for them.

```
00056 J8002 AP 21E0 JCICS ENTRY Wrapper_GetCommArea   341B4570

00056 J8002 AP 00E1 EIP   ENTRY ADDRESS

00056 J8002 AP 00E1 EIP   EXIT  ADDRESS              OK

00056 J8002 AP 21E0 JCICS EXIT  Wrapper_GetCommArea

00056 J8002 AP 21E0 JCICS ENTRY DTCTerminal_SEND

00056 J8002 AP 00E1 EIP   ENTRY SEND-TC

00056 J8002 DS 0002 DSAT  ENTRY CHANGE_MODE          QR

00056 QR    DS 0003 DSAT  EXIT  CHANGE_MODE/OK

00056 QR    AP FD01 ZARQ  ENTRY APPL_REQ             150C1B70,WRITE

00056 QR    AP FD81 ZARQ  EXIT  APPL_REQ

00056 QR    DS 0002 DSAT  ENTRY CHANGE_MODE          J8
```

```
00056 J8002 DS 0003 DSAT  EXIT  CHANGE_MODE/OK

00056 J8002 AP 00E1 EIP   EXIT  SEND-TC               OK

00056 J8002 AP 21E0 JCICS EXIT  DTCTerminal_SEND
```

A MIXTURE OF OTE TCB MODES

Having reviewed how CICS automatically switches open TCB modes to accommodate requests to OPENAPI TRUEs such as DB2 or Java classes running under a JVM, it is now appropriate to see how such TCB switches may be combined within a single program environment. The following trace shows such an example. A CICS Transaction Server 2.2 application invokes DB2 (via an EXEC SQL SELECT command). Being defined as threadsafe, control is returned to it under an L8 TCB (L8000 in this example). The application then issues an EXEC CICS LINK to a Java program (JVMHCW). As before, the JVM environment requires its own open TCB, and so CICS allocates one (J8000 in this case). Remember that OTE manages its open TCBs in different ways depending on the programming environment they are designed to support. An L8 TCB is therefore inappropriate for use by a JVM.

The application executes under the J8 TCB until it makes use of a JCICS function that invokes non-threadsafe CICS services (an EXEC CICS SEND in this example). CICS switches TCBs from the J8 TCB back to the QR TCB to process this request. When the command has been completed, CICS automatically switches TCBs back to the J8 TCB for the JVM environment to continue executing under (because JVMs require their own dedicated open TCB environment).

As the task continues invoking Java programs, DB2 calls threadsafe and non-threadsafe EXEC CICS commands, so this TCB switching continues automatically, as and when appropriate.

```
00053 QR    AP 2520 ERM   ENTRY COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00053 QR    DS 0002 DSAT  ENTRY CHANGE_MODE           0000000A
```

```
00053 L8000 DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00053 L8000 AP 3180 D2EX1 ENTRY APPLICATION           REQUEST EXEC SQL
SELECT
00053 L8000 AP 3181 D2EX1 EXIT  APPLICATION-REQUEST   SQLCODE 100

00053 L8000 AP 2521 ERM   EXIT  COBOL-APPLICATION-CALL-TO-TRUE(DSNCSQL )
00053 L8000 AP 00E1 EIP   ENTRY LINK                  0004,151084D0 ..d}

00053 L8000 PG 1101 PGLE  ENTRY LINK_EXEC             JVMHCW,NO,NO
00053 L8000 AP 1960 APLJ  ENTRY START_PROGRAM JVMHCW,CEDF,FULLAPI

00053 L8000 SJ 0201 SJIN  ENTRY INVOKE_JAVA_PROGRAM
JVMHCW,DCOB,AWJVMPR
00053 L8000 DS 0002 DSAT  ENTRY CHANGE_MODE           J8,C1E6D1E5

00053 J8000 DS 0003 DSAT  EXIT  CHANGE_MODE/OK        EXACT_MATCH
00053 J8000 AP 1802 JRAS  ENTRY com.ibm.cics.server.Wrapper
callUserClass
00053 J8000 AP 21E0 JCICS ENTRY DTCTerminal_SEND
00053 J8000 AP 00E1 EIP   ENTRY SEND-TC               0004,1B001E00

00053 J8000 DS 0002 DSAT  ENTRY CHANGE_MODE           QR
00053 QR    DS 0003 DSAT  EXIT  CHANGE_MODE/OK

00053 QR    DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00053 QR    AP FD01 ZARQ  ENTRY APPL_REQ              15082270,WRITE
00053 QR    AP FD81 ZARQ  EXIT  APPL_REQ
00053 QR    DS 0002 DSAT  ENTRY CHANGE_MODE           J8
00053 J8000 DS 0003 DSAT  EXIT  CHANGE_MODE/OK
00053 J8000 AP 00E1 EIP   EXIT  SEND-TC               OK
00053 J8000 AP 21E0 JCICS EXIT  DTCTerminal_SEND
00053 J8000 AP 180F JRAS  EXIT  com.ibm.cics.server.Wrapper
callUserClass
00053 J8000 SJ 0202 SJIN  EXIT  INVOKE_JAVA_PROGRAM/OK ..<.
00053 J8000 AP 1961 APLJ  EXIT  START_PROGRAM/OK      ....,NO,JVMHCW
00053 J8000 PG 1102 PGLE  EXIT  LINK_EXEC/OK          ,,,
```

A HOT-POOLED JAVA PROGRAM

Hot-pooled Java programs execute under H8 OTE TCBs. Their environment is analogous to the J8 TCBs provided by CICS for interpreted Java programs running under JVMs. However, OTE manages the H8 and J8 open TCBs in different ways, and in the same manner that J8 open TCBs are required for JVM program environments, so H8 open TCBs are required for hot-pooled ones.

The following trace shows an example of a hot-pooled Java application, executing under an H8 TCB (H8000 in this example). As with JVM programs running under J8 TCBs, CICS will switch to execute under the QR TCB when processing non-threadsafe EXEC CICS commands.

```
00049 H8000 AP 21E0 JCICS ENTRY DTC_Init              15195780
00049 H8000 AP 00E1 EIP   ENTRY ADDRESS               0004,13F043D0
00049 H8000 AP 00E1 EIP   EXIT  ADDRESS               OK
00049 H8000 AP 21E0 JCICS EXIT  DTC_Init              13500638
00049 H8000 AP 21E6 JCICS ENTRY Wrapper_callUserClass HW
00049 H8000 AP 21E0 JCICS ENTRY DTCTask_getCommonData
00049 H8000 AP 00E1 EIP   ENTRY ASSIGN  0004,13F045B8
```

SUMMARY

I hope that this article has helped explain the different aspects of the Open Transaction Environment within CICS, and the types of TCB activity that may be seen within CICS traces when utilizing OTE functions.

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

Helpful exit for shutdown assistant users

We are working with a customized version of the IBM shutdown assistant. You can find the samples (for CICS TS Version 2.2 in Assembler, COBOL, and PL/I) in your CICSTS22.CICS.SDFHSAMP library. If this exit is active during CICS shutdown processing, no user transactions can be started or running, only the CICS-supplied transactions and the SDAP transaction (shutdown assistant).

A troublefree shutdown process is ensured by using the shutdown assistant, which terminates any possibly hanging transactions.

The exit (program CSXXMATT) is activated by the shutdown

transaction SHUT and the appropriate CSSHUT program. Of course CSSHUT performs a number of important tasks during CICS termination.

The shutdown assistant takes control through a PLT table entry for CICS shutdown processing (DFHPLT TYPE=ENTRY, PROGRAM=name).

CSXXMATT

```
CSXXMATT TITLE 'XXMATT: CICS USER EXIT DURING TRANSACTION ATTACH'
*****************************************************************
* Abstract: This CICS Global User Exit is invoked during transaction  *
*           attach.                                                   *
*           If this exit is active, during the shutdown processing    *
*           from CICS only the CICS-supplied transactions and the     *
*           SDAP transaction but (eg) no user-transactions without a  *
*           terminal can be running.                                  *
* Change activity:                                                    *
*****************************************************************
         DFHUEXIT TYPE=EP,ID=(XXMATT)
         DFHUEXIT TYPE=XPIENV
         COPY  DFHSAIQY
         COPY  DFHXMIQY
CSXXMATT CSECT
CSXXMATT AMODE 31
CSXXMATT RMODE ANY
         SAVE  (14,12)                save registers
         LR    R3,R15                 R3 = base
         USING CSXXMATT,R3
         LR    R4,R1                  UEP parameter list address
         USING DFHUEPAR,R4
         L     R6,UEPATPTI            transaction name
         CLI   0(R6),C'C'    01-01    CICS-supplied transactions can
         BE    RETURN                 running
         CLC   0(4,R6),=CL4'SDAP'     Shutdown-assistant can running
         BE    RETURN
         L     R5,UEPXSTOR
         USING DFHSAIQ_ARG,R5
         L     R13,UEPSTACK
         DFHSAIQX CALL,                                              *
               CLEAR,                                                *
               IN,                                                   *
               FUNCTION(INQUIRE_SYSTEM),                             *
               OUT,                                                  *
               SHUTSTATUS(*),                                        *
               RESPONSE(*),                                          *
```

```
              REASON(*)
         CLI  SAIQ_RESPONSE,SAIQ_OK
         BNE  RETURN
         CLI  SAIQ_SHUTSTATUS,SAIQ_NOTSHUTDOWN
         BE   RETURN
SHUTDOWN DS   0H                    shutting down
         L    R6,UEPATTTK           transaction token
         MVC  TOKEN,0(R6)
         DROP R5
         USING DFHXMIQ_ARG,R5
         L    13,UEPSTACK
         DFHXMIQX CALL,                                          *
               CLEAR,                                            *
               IN,                                               *
               FUNCTION(SET_TRANSACTION),                        *
               TRANSACTION_TOKEN(TOKEN),                         *
               TCLASS_NAME('NLVTCL00'),                          *
               OUT,                                              *
               RESPONSE(*),                                      *
               REASON(*)
RETURN   DS   0H
         L    R13,UEPEPSA
         RETURN (14,12),RC=UERCNORM
TOKEN    DS   CL8
         LTORG
         END  CSXXMATT
```

The code for CSSHUT will be published next month.

Claus Reis
CICS Systems Programmer
Nuernberger Lebensversicherung AG (Germany)

Changes to Java support in CICS Transaction Server for z/OS Version 2 Release 3

CICS Transaction Server has, over a number of releases, provided support for application programs written in Java. As Java has gained in popularity, CICS has extended this functional area to deliver those features that customers have requested as well as many IBM believes will be of use to them in the future. A number of significant enhancements are included in CICS TS Version 2 Release 3.

This article describes the infrastructure used to support a Java-based workload under the control of CICS, outlining the changes introduced in CICS TS 2.3. It concentrates on Java applications rather than Enterprise JavaBeans or distributed IIOP applications, although some of the information included here applies to them as well.

CICS AND JAVA

Support for applications written in Java was introduced in CICS TS 1.3. Since then, the Java specifications have been considerably refined, and the run-time environments used to build Java Virtual Machines (JVMs) have undergone many changes. CICS has added to the function it provides to support extensions to the Java language, and to exploit enhancements from the run-time environment, which it uses to host Java based workloads.

In recent years, Java has become the language of choice for many application programmers throughout the IT industry. CICS has a history which goes back well beyond that of Java, and has adapted to many changes in the industry during its lifetime. Today, a large emphasis is placed on the Java programming model, and there is a growing number of skilled Java programmers. As a result, it is natural for CICS to extend the support it provides to follow these advances in the usage of Java.

Java continues to offer improvements in the form of increased function, and provides benefits from lower development costs, and the potential to build more robust and extendable applications. New applications are generally developed using later versions of the language. The Java run-time environments continue to be extended in support of language changes. In addition, newer versions of these run-times offer performance improvements, better memory utilization, and enhanced systems management functions. CICS TS 2.3 uses a later version of the Java run-time than previous releases and in doing so makes available infrastructure improvements and language extensions.

THE JAVA VIRTUAL MACHINE

The IBM SDK for z/OS, Java 2 Technology Edition, Version 1.4, with the PTF for APAR PQ79281, is the recommended Java run-time intended for use with CICS TS 2.3. It contains a number of advances over earlier versions. Support is provided to allow Java classes to be cached in a storage area, which can be shared between different JVMs. This offers the benefit of reduced memory requirements and fewer class loading operations for the JVMs sharing the class cache, as well as faster start-up times for individual JVMs. This and earlier versions of the run-time provide a component called the Just-In-Time (JIT) compiler, which can be used to optimize sections of Java classes when they are referenced, making them more efficient when they are used again. A small performance overhead is incurred when optimizing code, and the class cache can spread this between those JVMs which use its facilities.

Recent versions of the Java run-time support the serial reuse of classes across successive program invocations. In doing so, much of the initialization cost, which is incurred when a class is loaded, is avoided on subsequent reuse. The JVM attempts to reset its state after each program invocation. If a reset is successful, the JVM remains available for use by new requests and the significant overhead of starting up and shutting down a JVM for each program invocation is avoided. If a reset cannot be carried out, which is usually the result of the application leaving state information around which is unresettable, then the JVM is automatically destroyed. In this way, consecutive Java applications, which run in the same JVM, are completely isolated from each other.

SDK 1.3 introduced an optimized garbage-collection mechanism, which allows a separation of short-lived objects from long-lived classes and objects. Short-lived objects can be discarded, whereas long-lived classes and objects are reset when an application finishes execution, and persist for reuse by other applications. With SDK 1.4, CICS can choose which storage heap to load its middleware classes into and, in doing so, makes use of cacheing for these classes to improve the performance of the JVMs it manages.

JVMS AND CICS

The run-time environment for CICS Java applications has a number of components that control the execution of its classes. A Java program executes under the control of CICS in a particular storage key. CICS supports the use of different types for JVMs for use with the shared cache facility or to run as stand-alone JVMs. In addition, JVMs can be configured to control the way they are reused at the end of program execution. Each of these components is extended in CICS TS 2.3.

Prior to this release, all Java applications executed in CICS key storage areas. This has changed to allow customers to run applications in user key, and if they choose to do so they gain the benefits from the CICS storage protection mechanism, preventing application code from overwriting CICS control blocks. The storage key is defined on the Java program's resource definition in the same way as for programs written in any of the other languages that CICS supports. A new type of TCB is managed by CICS to allow JVMs to be started in user key storage areas.

The program resource definition also contains the name of a JVM profile, which is used to define some of the properties of the JVM needed to run the program, as well as some of the control options CICS uses to manage this JVM. A JVM profile can be used to define the type and mode of execution of a JVM.

There are three distinct types of JVM that CICS makes use of. Stand-alone JVMs work in isolation from each other, and are the only type supported by earlier releases of CICS. Worker JVMs are those that make use of the shared class cache for the execution of their Java applications. Master JVMs maintain the shared cache, but are not used directly to run Java applications.

The execution mode of a JVM controls the reset processing that takes place when a Java program finishes execution. In single-use mode, a JVM is created, used once, and then destroyed. This mode incurs the overhead of initializing and discarding a new JVM for each program execution. It provides effective transitional isolation and is useful when developing new applications because fresh versions of classes are loaded each time.

Resettable mode causes an attempt to be made to restore the JVM to a known state between transactions. If this reset is successful then the JVM is available for use by another application, which then benefits from not having to incur the cost of starting a JVM or loading these classes into it. If the reset is not successful then the JVM is destroyed. This mode offers good transaction isolation without the overhead of restarting the JVM between program invocations.

Both of these execution modes are available with CICS TS 2.2. In CICS TS 2.3, support is provided for JVMs running in a new mode, which is referred to in the CICS documentation as the Continuous JVM. Such JVMs do not undergo automatic reset processing once a Java program has finished executing there, and do not leave around any changes that they may have made to static data objects or JVM system properties. If they fail to do so then these changes may adversely affect the running of other applications in the same JVM at a later point in time. This execution mode offers significant performance benefits over that of the Resettable JVM and is recommended for any programs that fully reset their application state.

THE SHARED CLASS CACHE

In previous releases of CICS, all JVMs worked in isolation from each other, each maintaining a copy of the classes used by itself and by the Java applications that ran there. In this release, CICS takes advantage of the shared class cache facility provided by the SDK 1.4.

This facility makes use of a single JVM, referred to as the Master JVM, which CICS launches specifically to manage a shared storage area where classes are cached. A Master JVM cannot be used directly to run application workloads. Instead one or more JVMs, known as Workers, manage individual requests to run Java programs, and collectively make use of the classes loaded into the shared class cache. A Master and its Workers are referred to as a JVMset. Each CICS region can have only one active JVMset at any time. It may have a JVMset, which is in the

process of starting, and it may also have any number of JVMsets that are in the process of being phased out. The shared class cache is made up of all of the JVMsets that a CICS region is managing. However, system programming operations that query, modify, or manipulate the class cache are carried out against the active JVMset.

The shared class cache can be started during CICS initialization, explicitly using a command, or it can be configured to start when the first request is received to run a Java program requiring the shared class cache. It can be stopped as part of CICS termination or by using an SPI command. Once the instruction has been processed to stop the class cache, then no new requests are passed to any Workers that belong to the active JVMset. These Workers can be allowed to complete any work they may be processing at the time of the closure, or they can be forced to terminate immediately. Once all the Workers have ended, the Master JVM associated with them is also destroyed. A reload of the shared class cache is also possible, primarily to allow new versions of classes to be introduced. This operation causes a new JVMset to be started, which, once ready, becomes the active JVMset, while the previous one is phased out.

The shared class cache is beneficial in regions where multiple JVMs are needed to support the same workload. In these circumstances, all Worker JVMs make use of the same classes and benefit extensively once these are loaded and optimized within the shared cache facility. It has less of a role to play in development systems because these are likely to make use of stand-alone JVMs to test new applications.

JVM PROFILES AND JVM PROPERTIES FILES

CICS is responsible for controlling the operations that cause JVMs to be created. It does so in a number of steps. Firstly it assigns a TCB in a specific storage key, to allow the new JVM to have its own thread of execution. Then it requests a new Language Environment (LE) enclave to be allocated and uses this to run a CICS component, containing Java Native Interface

(JNI) calls, which creates and then controls the actions of the JVM.

A text file known as the JVM profile is used to control processing when a new JVM is created. The file contains a set of options, which CICS first interprets. Some of these are used by CICS for its own internal processes, such as the building of the control blocks it uses to represent the JVM resource or specific tasks such as adding a Worker to the active JVMset. Other options are formatted into an argument list, which is used when the Java run-time creates the JVM in response to a JNI call from the CICS code.

In this release of CICS, JVM profiles are stored in the Hierarchical File System (HFS) in a directory defined to a CICS region using the new SIT parameter JVMPROFILEDIR. In previous releases, JVM profiles were located in the dataset identified on the DFHJVM DD card in the region's JCL. JVM profile names are still limited to eight characters, but mixed case is now supported in line with other Unix file naming conventions.

The JVM profile option, JVMPROPS, which must be included in the profile, is used to locate another text file, known as the JVM properties file. This file contains the system properties used by the JVM once it has started. The file name may also include a directory path to allow the properties files to be stored in a separate directory from the one used for JVM profiles. CICS reads the options from the properties file and adds them to the argument list, which it then passes to the Java run-time on the JNI call to create the JVM.

CICS provides five sample JVM profiles, each with its own sample JVM properties file. Four of these are intended as examples of typical settings for different types of JVMs, which may be used within a CICS region. DFHJVMCC and DFHJVMPC could be used for a Master JVM and its Workers, assuming all the Workers shared the same options. DFHJVMPR and DFHJVMPS show how a stand-alone Resettable JVM and a single-use JVM might be configured. All these four sample JVM profiles and JVM properties files can be extensively customized.

The fifth sample, DFHJVMCD, is used internally for Java operations within CICS, such as running a CORBA server or publishing DJAR files. This profile needs a small amount of customization, and comments within it indicate where changes can be made. However, many of the settings need to be left unchanged because their alteration may adversely affect the operations carried out by CICS's internal Java components.

THE JVM POOL

The collection of JVMs to which a CICS region routes work is referred to as the JVM pool. This may include Stand-alone and Worker JVMs, and these will either have tasks assigned to them or be waiting for work. A Master JVM is not included in the JVM pool and, as a result, cannot be explicitly inquired on.

The size of the JVM pool is limited by the value of the MAXJVMTCBS SIT option. This may be further constrained by the amount of storage that is available to the system.

Systems that support mixed Java workloads are likely to make use of JVMs with different characteristics for specific applications. The pool could include some JVMs running in CICS key, while others might be running in user key. Each of these might be either a Worker or a Stand-alone JVM. Some of these JVMs may be running in resettable mode while others may be in continuous mode. Single-use JVMs are also part of the pool but, because they are relatively short lived, they may not be noticed if the pool is inquired on.

One restriction on the JVM pool is that all the JVMs making up the active JVMset for the shared class cache have to run in the same mode. Workers inherit their execution mode, together with some other properties, from the profile used to start their Master JVM. Workers can use different profiles to set options such as heap sizes, and can run in both CICS and user key storage within the same JVMset.

When CICS receives a request to run a Java program it will look for a JVM that is already started but currently not being used,

which matches the JVM profile and storage key for the new request. If found, the new request will be assigned to that JVM. However, if a match cannot be found and the number of JVMs is below the maximum allowed by the region, a new JVM will be created.

Once the maximum number of JVMs is reached, CICS queues new requests and uses a selection mechanism to decide on how to proceed, each time one of its JVMs completes work it has been servicing. When a JVM becomes free, it may not have the same profile or execution key as the first request in the queue. CICS uses an internal algorithm to process requests that are queued. It may choose to allocate some other request in the work queue to this JVM, or create a new JVM on the same TCB if the storage key matches, or it may recycle both TCB and JVM if the storage key is different. The aim of this procedure is to minimize the number of times JVMs and their TCBs are recycled. This mechanism is entirely under the control of CICS and cannot be configured by users.

THE JVM STORAGE MONITOR

JVMs running under CICS pass their storage requests to CICS via the Language Environment PIPI exits. There MVS GETMAINs are issued to acquire the storage outside the CICS Dynamic Storage Areas (DSAs). CICS domain services cannot be used in these PIPI exits because they can be called by non-CICS TCBs. No monitoring of this storage was done in earlier releases of CICS, leading to unpredictable results when the available MVS storage was exhausted. In this release, CICS provides a new storage monitor, which tracks the availability of MVS storage outside the CICS DSAs, reserves a cushion that can be used when the MVS storage is severely constrained, and notifies CICS domains so that they can take action to reduce the JVM storage requirements.

A JVM requests most of the storage area that it requires when it is created. During the execution of a Java application there are some circumstances where small amounts of additional storage

may be requested. The new storage monitor checks to see whether MVS storage is below a threshold each time a JVM storage request is issued through the PIPI exit. It reports on storage constraints, and notifies CICS so that it can block the request or terminate JVMs as they complete their current tasks to alleviate the situation. Once the constraint is relieved, CICS permits new Java requests to be processed normally.

The storage monitor is internally configured by CICS and cannot be customized by users. The messages it produces can be used to diagnose the cause of the problem, so that preventative steps can be taken to avoid a recurrence. Likely causes are having MAXJVMTCBS set too high for the system or running with unnecessarily large JVM heap sizes.

Advice on storage management issues can be found in the CICS *Performance Guide*.

REDIRECTING JAVA OUTPUT

Each JVM has a destination where it can write output messages, and another for error messages. These messages may be issued from the JVM's internal function or may come from a Java application running there. CICS is responsible for creating default destinations for both of these in the HFS. It makes use of options in the JVM profile to determine where these files are placed. The WORK_DIR option can be used to specify the directory path, and the STDOUT and STDERR options can specify names for these files. A further option, STDIN, is used to provide the name for an input file that some JVMs may require. If the directory path is not found in the JVM profile, CICS assumes a default of /tmp. Default file names of dfhjvmout, dfhjvmerr, and dfhjvmin are assumed if options for these are not found in the profile.

If the default settings are used, all the output messages from all of the JVMs associated with all the CICS regions that have access to the same file system will be written to the same file. The same is true for error messages. This is unlikely to be what many customers require. Some granularity of separation can be

achieved by including these options in JVM profiles, but as JVMs may share these, further separation may be needed.

If the **-generate** keyword is added to the STDOUT and STDERR option, CICS will generate a unique files name for each JVM by extending the name, provided in the option, with the CICS region's applid, task number, process ID, and time stamp. This allows separation of output and error messages between JVMs sharing the same profile. However, there may be a need to separate messages generated by individual Java invocations within the same JVM, or to direct the messages to somewhere other than the file system. In this release of CICS, a method of output redirection has been provided to meet these needs.

A single Java class can be named using the USEROUTPUTCLASS option in a JVM profile. This class can be one of two samples that CICS supplies, or it may be one provided by the user. The class is used to optionally redirect either or both output and error messages. One sample class, with the name of SJMergedStream, demonstrates how output and error messages can be redirected to the CICS TD queues CSJO and CSJE. It also shows how each message can be extended to include a time stamp, the Java program names, and other information that can be used for diagnostic purposes. The other sample is called SJTaskStream. It illustrates how messages can be directed to separate files for each invocation of a program within a JVM, the file name being qualified by task numbers. These samples are extensively documented in the CICS manuals and contain comments intended to help programmers customize them or create their own classes for this purpose.

These mechanisms should be used with care because they will have an effect on the performance of the systems that make use of them. As a result, the **-generate** extension to the STDOUT and STDERR option, and output redirection classes should be used only where necessary. They are useful aids for validating new applications or diagnosing problems with them, in a development system. In production, these options should be used only when an essential need exists.

CHANGES TO THE JAVA APPLICATION PROGRAMMING MODEL

Complex applications often comprise a number of programs and make use of the EXEC CICS LINK interface to allow one program to pass control to another. In this release, Java programs can issue LINK requests to other Java programs because the restriction of having only a single JVM associated with a CICS task has been removed. This brings the Java application programming model in line with that for the other languages CICS supports.

CICS provides a set of classes to provide the Java equivalent for many of the Application Programming Interfaces (APIs) used by other languages to access CICS resources. These are known as the JCICS classes and have been extended in this release to allow applications to directly make use of CICS Web Support. Classes have been provided for both the CICS Web API and the Document API, allowing applications to interrogate Web-based requests and to construct response in the form of HTML documents.

SUMMARY

CICS TS 2.3 continues to provide extensions to its ability to run Java applications under its control. Java programs can now run in either CICS or user key. They can do so within long-lived JVMs, which may or may not be reset between transactions, or which are used for single-use processing. The shared class cache facility allows classes to be shared between JVMs. A number of sample JVM profiles and JVM properties files are included to illustrate typical settings for different types of JVM. A selection mechanism is used to minimize the overhead of recycling JVMs, and a storage monitor is included to assist with problems when MVS storage is constrained. A facility is provided to optionally enhance and redirect output and error messages issued from JVMs or the applications that are running there.

This article outlines the function that CICS has provided to

support workloads written in Java, and focuses on these changes in the latest release of the product.

Mike Brooks
Senior IT Specialist
CICS Development (UK)

CICS TS V2.2 and V2.3 LDAP support using JNDI – configuration tips and examples

CICS TS V2.2 provided CICS EJB support, which requires the services of the Java Naming and Directory Interface (JNDI). CICS TS 2.3 further enhanced CICS EJB support, thereby greatly improving performance, and continues using the JNDI interface, using JNDI cache to improve the performance of JNDI look-ups.

So, if you are just starting with CICS EJB server implementation under CICS TS V2.2 or V2.3, you need to understand what services LDAP server provides and how it relates to CICS EJB server set-up. This article is intended to familiarize you with CICS LDAP support and help you configure the required CICS JNDI entries, using the LDAP server provided with z/OS.

JAVA NAMING AND DIRECTORY INTERFACE

JNDI enables Enterprise beans, and other Java programs running under CICS, to look up an external Enterprise bean by name in order to obtain its home interface, which can then be used to find or create an instance of the Enterprise bean. A service provider that provides the underlying repository and supports the JNDI Version 1.2 is therefore required.

The service provider can be a COSNaming service or an LDAP server. You can, for example, use the COSNaming service provided with the WebSphere Application Server or the LDAP server provided with z/OS.

A naming server is the place from which a client requesting EJB activity determines where the Enterprise bean will run. CICS supports two types of naming server:

- A COSNaming Server, which will reside on a workstation.
- An LDAP server, which can reside on a workstation or on a z/OS host.

CICS AND JNDI

CICS TS V2.*x* may be used with either a COSNaming server or LDAP server as the JNDI service provider. A CORBA Object Services (COS) naming server, such as that provided by WebSphere Application Server V3.5, or later, can be used. Alternatively an LDAP server with a CORBA object reference schema can be used. The LDAP server licensed as part of the base OS/390 or z/OS operating system meets this requirement. This is the recommended configuration because it simplifies interoperability between CICS TS and WebSphere Application Server V4 or later for OS/390 and z/OS when they share the same LDAP server.

CICS uses the same LDAP structure as WebSphere and CICS LDAP configuration is not required if the LDAP server is already set up for CICS. There is an advantage from an application perspective to publishing your CICS EJBs to the same LDAP server, and into the same directory structure, as your WebSphere EJBs. This makes it easier for beans from one system to look up beans from the other. If, for security or support reasons, you would like to separate your WebSphere LDAP from your CICS LDAP CICS, it is possible to use either a separate directory structure in the same LDAP server as WebSphere or an entirely separate LDAP server.

In order to understand how JNDI is used with CICS, it is helpful to get a high-level overview of how an IIOP request is processed.

The Request Receiver analyses the structured IIOP message. It compares the message with the templates defined in the CICS

RDO REQUESTMODEL definition, in order to select the transaction ID used to process the request.

The Request Processor:

- Locates the object identified by the request.
- For Enterprise bean requests, calls the container to process the bean method.
- For a CORBA stateless object, processes the request itself.

The Request Receiver can be installed into a listener region and the Request Processor and CICS CorbaServer can be installed into an Application Owning Region (AOR). Together the listener region and AOR comprise a logical EJB server. A CICS logical EJB server consists of one or more CICS regions configured to behave like a single EJB server.

JNDI provides a JNDI client API for accessing JNDI service providers.

An LDAP server provides the following services:

- Naming services – provides the means by which names are associated with objects and objects are found based on their names. It provides an association known as a binding between a name and an object.
- Directory services – a naming service can be extended with a directory service. A directory service associates names with objects and also allows such objects to have attributes. Thus, if you look up an object by its name you also get the object's attributes and *vice versa*. An example is the telephone company's directory service. It maps a subscriber's name to an address and phone number.
- API – the JNDI provides an API that applications can use to access a naming and directory service. The naming and directory service could be provided by any of a variety of servers, such as LDAP or COSNaming service. JNDI provides a Service Provider API (SPI), enabling access to the particular underlying directory service. The SPI is written by the vendor

of the underlying naming and directory service and is supplied as a Java class library. This allows arbitrary service providers to be plugged into the JNDI framework. In the javax.naming package, the JNDI provides classes that implement a naming interface for applications that look up only names and access objects bound to names. In the javax.naming.directory package, the JNDI provides a directory interface that extends the naming interface. This package allows applications to retrieve attributes associated with objects stored in the directory and to search for objects with specified attributes.

PROS AND CONS FOR USING LDAP SERVER CONFIGURED FOR WEBSPHERE ON Z/OS

If the name server that you have chosen for use by CICS has already been configured for WebSphere/390, there is likely to be very little configuration needed to enable CICS to use it.

Correct operation of the EJB support in CICS requires the chosen LDAP namespace to be configured with a WebSphere System Namespace – the publish and retract mechanisms of CICS both attempt to operate within a System Namespace structure. However, once inside an EJB method, or if executing a regular Java transaction in CICS, you can communicate with any LDAP namespace regardless of whether or not it supports a System Namespace.

My recommendation is to use LDAP server for z/OS configured for WebSphere, unless:

1 You need a different security configuration between CICS and WebSphere.

2 CICS needs to run in a separate domain from WebSphere. If you are building a new separate domain, WebSphere/390 and CICS will not easily be able to locate each other's Enterprise beans. If you do intend to build just a new domain, CICS needs to run in an entirely different system namespace structure on the LDAP server, that is, CICS needs to have a

containerdn that points to somewhere other than the existing namespace root location on the server.

LDAP CONCEPTS

LDAP uses contexts and initial context to define resources.

Contexts

A context is an object that contains zero or more bindings. For example, If the current context is o=MYCOMPANY,c=US, then the atomic name ou=Chicago refers to the child node in the DIT with the DN ou=Chicago,o=MyCOMPANY,c=US. A subcontext is a context within a context. The node ou=Chicago,o=MyCOMPANY,c=US is called a subcontext of o=MyCOMPANY,c=US.

InitialContext

An object factory is a class that accepts some information about how to create an object and returns an instance of that object. A context factory is a specialization of an object factory. It accepts information about how to create a context and returns an instance of the context. JNDI performs all naming and directory operations relative to a context. There are no absolute roots. To assist in finding a place to start, the JNDI specification defines an InitialContext class. This class is instantiated with properties that define the type of naming service in use. This class also provides the ID and password to use when connecting for the naming services that provide security. Once you have an initial context, you can use it to look up other contexts and objects. An application must establish an initial context as a starting point from which to do searches or traverse the name space.

CICS LDIF file

JNDI LDAP building system name space root entry (containerdn) is defined in the LDIF file located at /usr/lpp/cicsts22/utils/namespace/dfhsns.ldif.

This LDIF file creates the necessary system name space root entry, called the containerdn. You can choose any name, but shorter names are easier to type correctly. The containerdn is a distinguished name that points to an entry of type 'ibm-wsnNameTreeContainer'.

You can replace c=US (the root naming context suffix) with a name of your choice, but be aware that this must exist as a suffix on your chosen LDAP server.

Here is an example of what an LDIF file might look like:

```
dn: c=US
changetype: add
c: US
objectclass: country
description: LDAP Name Tree Area for CICS/WebSphere EJBs
entryowner: access-id:cn=clemas
aclpropagate: TRUE
ownerpropagate: TRUE

# Add the CICSUser (admin) user with the default password
dn: cn=CICSUser, c=US
changetype: add
objectclass: person
cn: CICSUser
sn: CICS Transaction Server 2.2 admin
userPassword: secret

# Add the CICSSystems (run-time) user with the default password
dn: cn=CICSSystems, c=US
changetype: add
objectclass: person
cn: CICSSystems
sn: CICS Transaction Server 2.2 run-time
userPassword: secret

# Add ACLS to the c=US entry
dn: c=US
changetype: modify
add: x
aclentry: access-id:cn=CICSUser,c=US:object:ad:normal:rwsc
aclentry: group:CN=ANYBODY:normal:rsc
```

The following two LDAP userids (principals) are used by CICS:

- The CICS LDAP Administration principal: cn=CICSUser,c=US is a principal that is given write access

to the myCompany-wsnTree node and below, so this principal can be used when needing to alter the system name space in some way. It makes a sensible alternative to a global LDAP administration principal that would have more access than strictly required.

- The CICS LDAP Run-time principal: cn=CICSSystems,c=US by default does not give the run-time principal any write access at all. It would be used if you wanted a very tight security model where CICS users can alter only their specific portion of the system name space below the legacyRoot node.

Here is how you can issue an LDAP modify command to add CICSUser ID:

```
# Add ACLS to the c=US entry
dn: o=YOUR_PLEX
changetype: modify
add: x
aclentry: access-id:cn=CICSUser,o=YOUR_PLEX:object:ad:normal:rwsc
```

An example can be found at /usr/lpp/cicsts22/utils/namespace/dfhsns1.ldif:

```
$ ldapmodify -v -p 1389 -D “cn=Admin,o=YOUR_PLEX” -w secret -f /tmp/
ldap.aclupdate ldap_init(yoursysid, 1389)
replace aclentry:
 group:CN=ANYBODY:normal:rsc
 access-id:cn=CICSUser,o=YOUR_PLEX:object:ad:normal:rwsc
modifying entry o=YOUR_PLEX
```

To add the ‘samples’ JNDI prefix entry to LDAP use the following command:

```
TEST:Userid:/tmp: $ ldapadd -h 127.0.0.1 -p 1389 -D
"cn=Admin,o=YOUR_PLEX" -w ******* \
> -f /usr/lpp/cicsts22/utils/namespace/addsamples.ldif

adding new entry myCompany-wsnName=samples,myCompany-
wsnName=legacyRoot,myCompany-wsnName= YOUR_PLEX,myCompany-
wsnName=domainRoots,myCompany-wsnTree=t1,o=YOUR_PLEX
```

A sample addsamples.ldif file to define ‘samples’ JNDI prefix should be located in /usr/lpp/cicsts22/utils/namespace and can look like this:

```
dn: myCompany-wsnName=samples,myCompany-wsnName=legacyRoot,myCompany-
wsnName=YOUR-PLEX,myCompany-wsnName=domainRoots,myCompany-
wsnTree=t1,o=YOUR_PLEX
myCompany-wsnname: samples
myCompany-wsnentrytype: PrimaryContext
myCompany-wsnnametreecontainerdn: myCompany-wsnTree=t1,o=YOUR_PLEX
myCompany-wsnpathfromcontainer:myCompany-wsnName=samples,myCompany-
wsnName=legacyRoot,myCompany-wsnName= YOUR_PLEX,myCompany-
wsnName=domainRoots
javaclassname: com.myCompany.ws.naming.ldap.WsnLdapContextImpl
objectclass: myCompany-wsnEntry
objectclass: myCompany-wsnPrimaryContextLocation
aclentry: access-id:cn=CICSUser,o=YOUR_PLEX:object:ad:normal:rwsc
aclentry: group:CN=ANYBODY:normal:rsc
aclentry: access-id:cn=CICSSystems,o=YOUR_PLEX:object:ad:normal:rwsc
JNDI environment variables
```

Some of the most common environment variables used are java.naming.factory.initial and java.naming.provider.url in the properties file for each EJB client. This provides the EJB client with the location of our JNDI Naming server and the initial context class to use. Others such as java.naming.security.authentication and java.naming.security. principal are specified in the CICS system properties file (dfjjvmpr.props), which enabled you to give our CICS region write access to our JNDI name space when using a secure LDAP server. This file can be found in the /usr/lpp/cicsts22/props/ directory.

Here is an example of what dfjjvmpr.props may look like (showing only entries relevant to LDAP):

```
com.myCompany.cics.ejs.nameserver=ldap://mycompany.com:1389
com.myCompany.ws.naming.ldap.noderootrdn=myCompany-
wsnName=legacyRoot,myCompany-wsnName= YOUR_PLEX ,myCompany-
wsnName=domainRoots
com.myCompany.ws.naming.containerdn= myCompany-wsnTree=t1,o=YOUR_PLEX
java.naming.security.authentication=simple
java.naming.security.principal=cn=CICSUser,c=US
java.naming.security.principal=cn=CICSUser,o=YOUR_PLEX
java.naming.security.credentials=secret
```

The properties file is loaded from HFS, not from the XDFHENV PDS. It should be noted that definitions made in the /usr/lpp/ cicsts/cicsts22/props/dfjjvmpr.props file will override values specified in CICSTS22.CICS.XDFHENV(DFJJVMPR). This is not applicable to CICS TS 2.3.

In CICS TS 2.2 you point to the properties file from the JVM profile member. You use the JVMPROPS parameter, in the JVMPROFILE, to specify the full path of the system properties file that CICS is to use when creating a JVM. CICS provides two sample system properties files, dfjjvmpr and dfjjvmps, in the SDFHENV partitioned dataset. These properties files are designed to support their corresponding JVM profiles (dfjjvmpr for DFHJVMPR, and dfjjvmps for DFHJVMPS). Both the sample properties files are defined with the &CICS_DIRECTORY symbol, which is replaced with your own value when you run the DFHIJVMJ installation job.

When we deploy servlets and EJBs into WAS with SMEUI, we register the EJB home instances into a namespace.

Java:comp/env/ejb/home-name is passed on the look-up method call. Make sure the declared java:comp name returns the EJB's home reference wherever it is located.

java:comp/env is a datasource look-up string for J2EE applications.

To refresh JVM settings issue:

```
CEMT SET JVM PHASEOUT
```

RDO DEFINITIONS

Let's now discuss the following RDO resource definitions that support Enterprise beans under CICS TS 2.2.

The CORBASERVER resource definition is used to define an execution environment for Enterprise beans and stateless CORBA objects.

You can use the CORBASERVER resource definition to define the attributes of an execution environment for Enterprise beans and stateless CORBA objects.

A CICS EJB server may contain more than one CORBASERVER. This feature allows us to deploy Enterprise beans with different characteristics, such as timeout values and different JNDI

locations in the same CICS region. The shelf is in an HFS directory associated with the CORBASERVER, which holds copies of the deployed Java Archive (JAR) files. The DJARdir is an HFS directory containing the deployed JAR files that are to be automatically installed into the CORBASERVER.

The Server ORB Attributes provide the hostname for the CORBASERVER.

The following is a brief description of the CORBASERVER resource definition parameters:

- JNDIPREFIX – the prefix added to a bean name to specify a fully-qualified JNDI name.
- AUTOPUBLISH – specifies that its Enterprise beans should be published automatically when a DJAR is installed.
- SESSIONBEANTIME – specifies, in days, hours, and minutes, the period of inactivity after which a session bean may be discarded by CICS.
- SHELF – the fully-qualified name of a directory containing the Shelf. A Shelf is used to store copies of the installed deployed JAR files. This HFS directory is also used as a work area for the AORs.
- DJARDIR – the fully-qualified name of a directory that can contain deployed JAR files. If specified, CICS will, periodically or on command, scan this 'pickup' directory and automatically install any new or changed deployed JAR files.
- HOST – specifies the TCP/IP host name (domain name) or IP dotted-decimal address of this Logical EJB/CORBA server. It must match the IP address defined in the corresponding TCPIPSERVICE definition. The hostname is included in the reference to the objects in the CORBASERVER.
- UNAUTH – the TCP/IP service attributes name the TCPIPSERVICEs, with the appropriate security attributes, that are listening for IIOP requests on behalf of the CORBASERVER.

- The DJAR resource definition is used to define a deployed JAR file. The deployed JAR file contains Enterprise beans and is a Hierarchical File System (HFS) file.

You use the DJAR resource definition to specify a deployed JAR file. This file contains the Enterprise bean classes and the code generated by the deployment tool. It is a Hierarchical File System (HFS) file. A DJAR definition can be created automatically by using the 'pickup' directory mechanism.

The DJAR resource definition parameters are:

- DJAR – name of DJAR resource definition.
- CORBASERVER – name of the server in which this DJAR is to be installed.
- HFSFILE – specifies the 1-255 character fully-qualified file name of the deployed JAR file on HFS. The name is case sensitive and may not contain blanks.
- The TCPIPSERVICE resource definition is used to identify the port number the listener task will open.

You use the TCPIPSERVICE resource definition to define which TCP/IP services are to be supported. The services that can be defined are IIOP and HTTP (for CICS Web support).

TCPIPSERVICE resource definition parameters are:

- TCPIPSERVICE – specifies the eight-character name of this TCP/IP service.
- URM – specifies the name of the User-Replaceable Module (URM) to be invoked by this service, to derive the MVS userid to be assigned to the IIOP service request. DFHXOPUS is a CICS-supplied sample program that provides a default mechanism to derive the userid.
- PORTNUMBER – specifies the decimal number of the port on which CICS is to listen for incoming client requests.
- PROTOCOL – identifies the type of service provided by this port. IIOP connections support EJB, while HTTP connections support the CICS Web support.

- TRANSACTION – specifies the CICS transaction attached to receive and analyse the request. CIRR (Request Receiver task) is the default for IIOP support.

The REQUESTMODEL resource definition is used to map the Internet Inter-ORB Protocol (IIOP) inbound request to a CICS transaction that will execute the method request.

A REQUESTMODEL resource definition specifies the mapping between an Internet Inter-ORB Protocol (IIOP) inbound request and the CICS transaction that is to be initiated.

REQUESTMODEL resource definition parameters are:

- REQUESTMODEL – specifies the eight-character name of this request model definition.
- CORBASERVER – specifies the name of the CORBASERVER for this REQUESTMODEL.
- TYPE ({GENERIC | CORBA | EJB }) – indicates the type of REQUESTMODEL. GENERIC for both CORBA and EJB support.

MANAGING EJB BEANS

Here are the most useful CEMT commands you can use to manage CICS EJB server:

- The CEDA transaction supports the specification and the installation of TCPIPSERVICE, CORBASERVER, DJAR, and REQUESTMODEL definitions.
- CEMT INQUIRE BEAN transaction determines whether a specified bean is installed in a specified CORBASERVER and, if so, which DJAR contains the bean.
- CEMT PERFORM DJAR() or CORBASERVER() PUBLISH. To publish a bean means to install the Enterprise bean's home reference into the JNDI namespace.

  We can now execute methods on the Enterprise bean.

- CEMT PERFORM DJAR() or CORBASERVER() RETRACT. To retract a bean means to remove an Enterprise bean's home reference from the JNDI namespace.

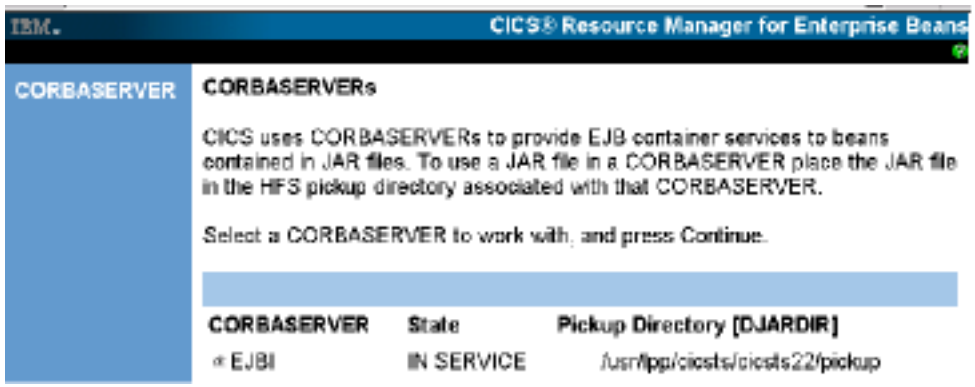

Figure 1: CORBASERVER display

IVP TESTING USING HELLOWORLD

Here is an example of an RDO definition for IVP testing using

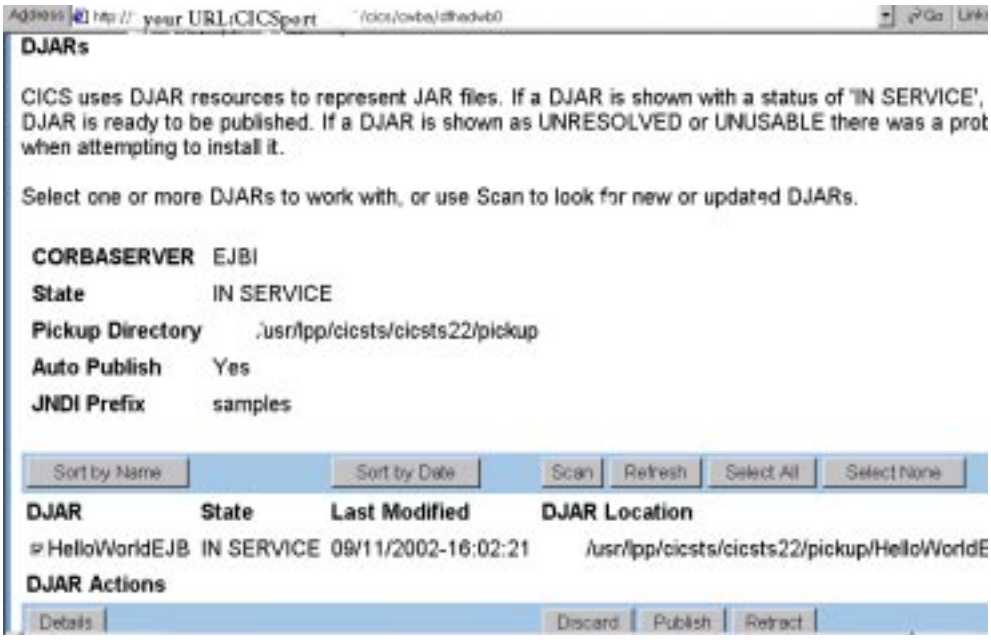

Figure 2: CORBASERVER(EJBI) detailed information

Figure 3: JNDI prefix 'samples' in front of 'HelloWorld'

HelloWorld (under CICS TS 2.2):

```
CORbaserver    :   EJBI
Group          :   EJBIVP
DEscription    :   CORBASERVER for EJB IVP program
Jndiprefix     :   samples
Autopublish    :   Yes
SHelf          :   /usr/lpp/cicsts/cicsts22/shelf
Host           :   MVSSYSID
```

Figure 1 is a display of this CORBASERVER using RMEB (CICS Resource Manager for EJBs).

Figure 2 shows detailed information for CORBASERVER(EJBI). Please notice JNDI prefix information, which is the key for JNDI configuration.

Figure 3 shows that JNDI prefix 'samples' was attached in front of bean 'HelloWorld' when CORBASERVER was published. So, Application will reference this bean as 'samples/HelloWorld'. When there are multiple beans within the JAR file, they will all have the same JNDI prefix, defined on the CORBASERVER definition.

IVP SCRIPT – RUNEJBIVP

Here is a sample IVP script that can be used to test your set-up. It is located in /usr/lpp/cicsts22/samples/ejb/helloworld.

```
# CICS EJB IVP run script

# ENVIRONMENT PROPERTIES
# ======================
#
# Modify the following to match your IBM SDK 1.3.1 installation director
# (as set in your CICS JVM profile):
JAVA_HOME=/usr/lpp/java/IBM/J1.3/
#
# Modify the following to match your CICS TS 2.2 installation directory
# (as set in your CICS JVM profile):
CICS_DIRECTORY=/usr/lpp/cicsts/cicsts22/
 # COMMON PROPERTIES
# =================
#
# Modify the following to match your CICS region's JNDI provider URL. Th
# is set in CICS using the 'com.myCompany.cics.ejs.nameserver' property.
# For example
#    CosNaming->JNDI_PROVIDER_URL=iiop://nameserver.location.company.com
#         LDAP->JNDI_PROVIDER_URL=ldap://nameserver.location.company.com
JNDI_PROVIDER_URL=ldap://your URL : LDAP port
#
# Modify the following to match your CORBASERVER's Jndiprefix. The
# default value for the CICS samples is already supplied and shouldn't
# need to be changed.
CORBASERVER_JNDI_PREFIX=samples
# Modify the following to match the Bean Name as defined in the
# deployment descriptor of the sample jar file. By default this
# should be 'HelloWorld'
BEAN_NAME=HelloWorld
#
# LDAP NAMING PROPERTIES
# ======================
#
# If you are using the LDAP naming service, modify the following
# properties to match your CICS region's LDAP naming properties. If you
# are not using LDAP as your JNDI naming service then these properties
# will be ignored. The LDAP properties will normally be set by the LDAP
# administrator and can be found in the CICS JVM properties file.
#
#
LDAP_CONTAINERDN=myCompany-wsnTree=t1,o=YOUR_PLEX
#   (as specified in the JVM properties file using the following name:
#    'com.myCompany.ws.naming.ldap.containerdn')
#
```

```
LDAP_NODEROOTDN=myCompany-wsnName=legacyRoot,myCompany-
wsnName=YOUR_PLEX, ,myCompany-wsnName=domainRoots
#   (as specified in the JVM properties file using the following name:
#    'com.myCompany.ws.naming.ldap.noderootrdn'.
#   Note, this property is option
#
# EXPERT PROPERTIES
# =================
#
# You may switch from using the default JNDI initial context factory if
# wish. If using WebSphere v 4.Ø as your JNDI provider then you should
# change this property to
#                  'com.myCompany.websphere.naming.WsnInitialContextFac
# otherwise it is usually best to leave this value as 'default'.
#
INTIAL_CONTEXT_FACTORY=default
#
# Other example naming factories include:
#
#INTIAL_CONTEXT_FACTORY=
                  com.myCompany.websphere.naming.WsnInitialContextFactor
```

IVP testing from Unix Services:

```
TEST:Userid:/usr/lpp/cicsts/cicsts22/samples/ejb/helloworld: $ ./
runEJBIVP
CICS EJB IVP: Querying the Java SDK level
java version "1.3.1"
Java(TM) 2 Run-time Environment, Standard Edition (build 1.3.1)
Classic VM (build 1.3.1, J2RE 1.3.1 MYCOMPANY OS/39Ø Persistent Reusable
VM build hm13
1s-2ØØ2Ø2Ø7 (JIT enabled: jitc))

CICS EJB IVP: Starting the EJB client program
HelloWorld client program started
Performing JNDI lookup using LDAP
Testing the following location: samples/HelloWorld
Located home interface for HelloWorld bean
You said: Hello from CICS EJB IVP client
HelloWorld client program ended
CICS EJB IVP: Completed successfully

Testing CICS IVP from WebSphere - http://.us.yourcompany.com:9Ø91/
cicshello/
```

CICS TS 2.3 AND WEBSPHERE V5.0 ENHANCEMENTS AND UPDATES

The COS Naming Directory Server supplied with WebSphere

Application Server Version 5 differs from that supplied with WebSphere Application Server Version 4. From Version 5 onwards:

- The default TCP/IP port used by the COS Naming Directory Server is 2809 (rather than 900, as in WebSphere Version 4).
- Java objects must be published to a specially-architected location called domain/legacyRoot. (CICS publishes Java objects to a context defined by the JNDIPREFIX option of the CORBASERVER definition, where the JNDI prefix is a relative path.) If you do not specify the /domain/legacyRoot path from the root node of the name space, CICS tries to publish Java objects to a JNDI prefix location relative to the root node itself. This works for the COS Naming Directory Server supplied with WebSphere Application Server Version 4, but fails with that supplied with later versions of WebSphere Application Server.
- The recommended way to specify the location of your name server is on the com.myCompany.cics.ejs.nameserver property in the JVM system properties file. If you use the COS Naming Directory Server supplied with WebSphere Application Server Version 5, you should specify the location like this:

```
com.myCompany.cics.ejs.nameserver=iiop://mycsserv.com:28Ø9/domain/
legacyRoot
```

- CICS objects must be published to a specially-architected location (in the WebSphere naming structure) called domain/legacyRoot. CICS publishes objects to a context defined by the JNDIPREFIX option of the CORBASERVER definition, where the JNDI prefix is a relative path. If you do not specify the /domain/legacyRoot path from the root node of the name space, CICS tries to publish objects to the JNDI prefix location relative to the root node itself. This works for the COS Naming Directory Server supplied with WebSphere Application Server Version 4, but fails with that supplied with later versions of WebSphere Application Server.

In CICS TS 2.3 there is a new profile called dfjjvmcd.props, where you specify the system properties necessary to configure your JNDI nameserver com.myCompany.cics.ejs.nameserver system property, including the URL and TCP/IP port number of the name server that you use for JNDI references.

For example:

```
com.myCompany.cics.ejs.nameserver=ldap://myLDAPserv.com:389
```

For a standard COS Naming Directory Server you specify:

```
com.myCompany.cics.ejs.nameserver=iiop://mycsserv.com:900
```

An example of this statement is included in the CICS-supplied sample JVM properties file, dfjjvmpr.props.

Note: if you are using a COSNaming service, and you have chosen to specify it in java.naming.provider.url, do not specify it again here.

You can set up a file called jndi.properties to contain JNDI nameserver configuration properties that are common across a set of CICS regions. By default, CICS does not attempt to locate a jndi.properties file. Include the following system property to cause CICS to load jndi.properties for this JVM:

```
com.myCompany.cics.ejs.loadjndiproperties=true
```

Place the directory containing the jndi.properties file on either the sharable application class path (in the JVM properties file) or the trusted middleware class path (in the JVM profile), in all the relevant JVM profiles or JVM properties files, for all the regions that you want to share the same nameserver settings.

You can turn the JNDI cache on or off:

```
com.myCompany.websphere.naming.jndicache.cacheobject={ populated |none}
```

The JNDI cache stores the results of JNDI look-ups in local storage, so that, if an application does the same look-up twice (perhaps in different tasks), the results are already available. Note that the cache is JVM-specific. That is, there is a separate cache for each JVM. This only works with an IBM JNDI name server:

- Populated – the JNDI cache is active.
- None – the JNDI cache is not used.

You can specify, in minutes, the 'time to live' of the JNDI cache:

```
com.myCompany.websphere.naming.jndicache.maxcachelife={20 |mins}
```

If the cache is accessed after this time is exceeded the entire cache is flushed of its contents.

You can specify the Container Distinguished Name for the LDAP name server:

```
com.myCompany.ws.naming.ldap.containerdn=
```

You can specify the Noderoot Relative Distinguished Name for the LDAP name server:

```
com.myCompany.ws.naming.ldap.noderootrdn=
```

J2EE components use the framework classes to acquire a connection to an EIS and to send and receive data. First, a J2EE component obtains a ConnectionFactory object for the particular EIS that is to be accessed – for example, CICS. (The component may manufacture the ConnectionFactory programmatically or, more likely, look it up in a JNDI namespace.) It uses the ConnectionFactory to get a Connection object. Then it uses the Connection object to create one or more Interaction objects. It executes commands on the EIS through these Interaction objects.

The example below shows the CCI framework classes being used to connect to an EIS and execute a command.

```
ConnectionFactory cf = <Lookup from JNDI namespace>
Connection conn = cf.getConnection();
Interaction int = conn.createInteraction();
int.execute(<Input output data>);
int.close();
conn.close();
```

PERFORMANCE

The performance of Enterprise beans has been improved in

CICS TS 2.3. In particular, when a JVM is reset, some objects are cached for reuse (com.myCompany.websphere.naming.jndicache.cacheobject= populated), which greatly improves performance of JNDI look-ups.

In previous releases of CICS, an EJB deployed JAR file was considered to contain only application code. In CICS TS V2.3 the CICS-generated code within the deployed JAR file (the implementation of the beans' home and component interfaces) is treated as middleware. This means that some objects that were previously held in the application heap and discarded when the JVM was reset have become long-lived middleware heap objects. Now, when a JVM is reused to process requests against the same Enterprise beans, the cached objects are reused, which greatly improves performance.

It should be noted that after the EJB server has been migrated to CICS TS 2.3, some clients may have stale, cached, IORs that point to the old server. This is because some application servers cache the results of JNDI look-ups locally to increase performance. You may find that these caches have to be purged before the new IORs are used.

Z/OS LDAP SERVER START-UP JCL

This example shows a sample start-up JCL that we used for our LDAP server started task, SCSLDAP1:

```
//SCSLDAP1 PROC PARMS=''
//GO EXEC PGM=GLDSLAPD,REGION=0M,TIME=NOLIMIT,
// PARM=('/&PARMS >DD:SLAPDOUT 2>&1')
//STEPLIB DD DSN=GLD.SGLDLNK,DISP=SHR LDAP library
// DD DSN=DB7Q7.SDSNLOAD,DISP=SHR DB2 library
//CONFIG DD PATH='/usr/cicsts22/ldap/slapd.conf' Configuration file
//ENVVAR DD PATH='/usr/cicsts22/ldap/slapd.envvars' Environment vars
//DSNAOINI DD PATH='/usr/cicsts22/ldap/dsnaoini' DB2 CLI parms
//SLAPDOUT DD SYSOUT=* //SYSOUT DD SYSOUT=* stdout/stderr msgs
//SYSUDUMP DD SYSOUT=*
//CEEDUMP DD SYSOUT=* LE dumps
```

REFERENCES

- IBM Redbook *Enterprise JavaBeans for z/OS and OS/390 CICS Transaction Server V2.2* (SG24-6284-01) contains a good chapter on *Configuring the JNDI server*. This includes a detailed description of how to use the scripts provided to set up an LDAP server, including details of how the process differs if your LDAP server is already configured for use by WebSphere.
- APAR PQ61492 outlines PROBLEMS WITH CICS LDAP CONFIGURATION FILES AND DOCUMENTATION. PQ61492 was raised to address various problems with the CICS LDAP configuration files – WebSphereNamingSchema.ldif and dfhsns.ldif. Also there are inconsistencies between these files and the documentation which describes their use.
- IBM Redbook *WebSphere Application Server for z/OS Form-Based Authentication with LDAP*, REDP-3664-00.

Elena Nanos
IBM Certified Solution Expert in CICS Web Enablement
Zurich NA (USA)

CICS questions and answers

Q Is there a way to browse TSQs in CICS that have hex characters in the name? CEBR doesn't allow me to enter hex as the Qname.

A A great question because it allows me to highlight a little known, but documented, feature of CEMT. After using CEMT to display TSQs, it's possible to put a 'B' to the left of the Qname, then CEMT will start CEBR for you and show you the contents of the queue.

Q I seem to be missing some user journal information on occasion. I've heard that I need to FLUSH user journals in CICS TS if CICSshutdown is not normal; is this true and how can I achieve this?

A Yes, this is true. During Immediate shutdown CICS does not flush user journal buffers (DFHLOG and DFHSHUNT don't need to be flushed). A common way to fix this is to add a PLTSD program that flushes user journals (START BROWSE, NEXT looking for user journals) and add this program to the PLTSD before the Shutdown Assist program (which is normally the program that issues the Immediate shutdown).

If your PLTSD program will not be invoked, because for example, the Immediate Shutdown is requested by the operator, then the application program needs to issue an EXEC CICS WAIT JOURNALNAME command just before returning to CICS, or specify the WAIT option on each WRITE to the journal.

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.

Why not share your expertise and earn money at the same time? Articles for *CICS Update* 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.

Seagull Software and InnerAccess Technologies have entered into a strategic partnership in which InnerAccess' z/Services connector technology for CICS, IMS, and Advantage CA-IDMS will be embedded into Seagull's LegaSuite solution for host integration.

The host integration module of LegaSuite supports rapid development of custom legacy connections to IBM mainframe, ICL, iSeries, Unix/VT, and other character-based applications, integrating legacy business functions with other enterprise applications, composite applications, and service-oriented architectures.

LegaSuite provides connectors for 3270 screen-based integration as well as various methods for CICS, IMS, and IDMS transaction server integration. The addition of InnerAccess' z/Services expands LegaSuite to include a high-performance XML-based gateway into CICS, IMS, and IDMS transactions that bypasses the 3270 data stream, run outside the transaction monitor for zero impact to production regions, and require no changes to the host application.

For further information contact:
Seagull Software, 3340 Peachtree Road NE, Atlanta, GA 30326, USA.
Tel: (404) 760 1560.
URL: http://www.seagullsoftware.com/about/media/pressreleases/253.html.

* * *

Compuware has announced Release 4.5 of Abend-AID for CICS.

The product, formerly CICS Abend-AID/FX, analyses the causes of application and region faults, enabling programmers to resolve critical problems for customers and users, regardless of the system's complexity or technology. Abend-AID for CICS gives programmers online access to vital information about a transaction abend, storage violation, application deadlock, or system outage.

For further information contact:
Compuware, One Campus Martius, Detroit, Michigan 48226, USA.
Tel: (313) 227 7300.
URL: http://www.compuware.com/products/abendaid/cics.htm.

* * *

Attachmate has released myEXTRA! Smart Connector Mainframe Edition Version 4.0.2, bringing direct access to mainframe data sources such as VSAM, IMS/DB, DB2, and Adabas, plus native transactional access to CICS and IMS/TM applications.

Based on a Service-Oriented Architecture (SOA), the product makes legacy assets reusable as services for new application development. Smart Connector Mainframe Edition provides real-time access to mainframe data sources and uses standard SQL queries to join data across disparate systems.

Mainframe-resident connectors provide transactional access to CICS and IMS programs. XML-based event generation and routing enable legacy applications to interact bi-directionally with other applications.

For further information contact:
Attachmate, 3617 131st Ave SE, Bellevue, WA 98006, USA.
Tel: (425) 644 4010.
URL: http://www.attachmate.com/products/profile/0,1016,4176_1,00.html.