220

CICS update

March 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.

Preventing problems with mixed-case fields in CEDA in CICS Transaction Server Version 2.3

Prior to CICS Transaction Server Version 2.1, when CICS resources were defined using Resource Definition Online (RDO) or the CEDA transaction, the case of the attributes of the various keywords was not considered an issue. All input would simply be folded into upper case by the 3270 terminal and everything would work. However, with the introduction of the use of Enterprise JavaBeans (EJBs) with CICS in CICS TS 2.1, things changed. Here the use of lower-case input is not only desirable but necessary.

Now, in order to use EJBs with CICS, the customer needs to define CICS resources such as CORBASERVERs or DJARs containing fields representing HFS file names, and things can easily go wrong. Having defined a CORBASERVER, for instance, one then installs it. On attempting to use the CORBASERVER, difficulties may be encountered because something has gone wrong – such as the shelf directory cannot be found even though it is known to exist in HFS. What happens then is the realization that upper-case translation should have been turned off for the terminal. The input of potentially 255 characters for CORBASERVER's SHELF name has to be entered again on the CEDA definition panel so that this time it is saved in mixed case!

The transaction CEOT (EXEC Operator Terminal) was extended to allow users to alter the upper-case translation setting of their terminal. By setting UCTRAN (upper-case translation) to NO, case folding of input from the 3270 terminal could be inhibited (or one could specify that only transaction names be folded by setting UCTRAN status to TRANIDONLY). This helps, after the event, but is a rather indirect way of solving the problem.

Nearly every CICS user falls over the RDO case folding issue when they first start playing with EJBs. How long they take to conclude that upper-case folding is the root of the problem they are experiencing varies according to the skill of the user. Having spent an hour trying to work out what has gone wrong, they figure out that the resource definition needs changing, they make the necessary changes in CEDA, and then continue. The next time this problem is encountered, they resolve it that much faster. Unfortunately, the more experienced the user is with CICS the longer they take to realize what the cause of the problem is. This is because experienced users know that case folding occurs and do not expect it to be an issue. That's how CICS has always worked.

Given that no warning occurs when defining such a resource during RDO, the user has to analyse the side effects of having an incorrect resource definition defined later on. This may be a CORBASERVER failing to resolve, a DJAR becoming UNUSABLE, or a REQUESTMODEL not being honoured at run time. The problem can manifest itself in a lot of undesirable ways.

Documentation has stressed that you must enter these various fields in CEDA very carefully while considering that the input could be in mixed case. It also points out that the CEOT transaction can be used to alter the case folding property of the current terminal. In addition, many of the problems that occur if the resource definition is wrong result in messages being issued which, when looked up in the CICS *Messages and Codes Manual*, do have explanations suggesting that case folding may be at the root of the problem. However, these documented warnings were considered as mere compensatory measures. During usability testing of both CICS TS 2.1 and 2.2 it was recognized as a major issue because real customers were observed struggling with trying to solve these problems for themselves.

WHAT IS THE SOLUTION?

The solution is to tackle the problem at source and prevent the automatic folding of the input through the CEDA panel without the user having to think about using CEOT to inhibit upper-case translation on the terminal he is using for RDO. In order to do this we had to consider the following.

There are three different field types to consider for case folding:

1 Those that must always be in upper case.

2 Those fields that can be mixed case, but will be folded if UCTRAN(YES) is specified for the terminal.

3 Those fields that must not be folded, but are case-sensitive.

Prior to CICS TS 2.3, only field types 1 and 2 were distinguished by CEDA. Now, it also takes into account type 3 fields. CEDA has been altered to examine what the UCTRAN setting for the terminal is and to force UCTRAN(NO) for the duration of the session that CEDA is in use. It then decides on an individual field basis whether to fold the field to upper case or not, depending on what the UCTRAN setting was. That is, if UCTRAN(YES) had been specified, then type 2 fields such as TRANSID (see example below) are folded into upper case. Type 3 fields such as the long fields in question (listed below) would not be folded, but left unchanged. CEDA can determine which fields are to be left alone because a new internal identifier for type 3 fields has been created. The relevant fields are indicated on the CEDA definition panel by '(Mixed Case)'.

Type 1 fields are always folded regardless of the UCTRAN value.

The UCTRAN setting will be restored to its original value on exiting CEDA either normally or via a new HANDLE ABEND routine.

The fields that are case-sensitive and in which folding into upper case causes a major problem are as follows:

- PROGRAM – JVMCLASS and JVMPROFILE.
- REQUESTMODEL – MODULE, INTERFACE, OPERATION, and BEANNAME.
- CORBASERVER – JNDIPREFIX, HOST, SHELF, DJARDIR, and CERTIFICATE.
- DJAR – HFSFILE.

An example of a DJAR definition follows containing a mixed-case HFS file name.

Example 1: CEDA define DJAR panel:

```
OVERTYPE TO MODIFY                                CICS RELEASE = Ø63Ø
  CEDA  DEFine DJar( JARFILE  )
   DJar           : JARFILE
   Group          : TEST
   Description  ==>
   Corbaserver  ==> COR1
   Hfsfile      ==> winvms2c.jar.file
   (Mixed Case) ==>
                ==>
                ==>
                ==>




                                   SYSID=CIAT APPLID=IYK2Z4P2
  DEFINE SUCCESSFUL           TIME:  16.Ø2.21  DATE: Ø3.3Ø1
PF 1 HELP 2 COM 3 END   6 CRSR 7 SBH 8 SFH 9 MSG 1Ø SB 11 SF 12 CNCL
```

An example of a section of a PROGRAM definition follows, containing JVMClass and JVMProfile, which are in mixed case.

Example 2: CEDA alter PROGRAM panel:

```
  OVERTYPE TO MODIFY                              CICS RELEASE = Ø63Ø
   CEDA  ALter PROGram( ARD1PROG )
 +  REMOTEName   ==>
    Transid      ==> J1AS
    EXECUtionset ==> Fullapi          Fullapi | Dplsubset
   JVM ATTRIBUTES
    JVM          ==> Yes              No | Yes
    JVMClass     ==> jclass
   (Mixed Case)  ==>
                 ==>
                 ==>
                 ==>
    JVMProfile   ==> dfhjvmpr        (Mixed Case)
   JAVA PROGRAM OBJECT ATTRIBUTES
    Hotpool      ==> No               No | Yes
```

```
                                               SYSID=CIAT APPLID=IYK2Z4P2
ALTER SUCCESSFUL                    TIME:  14.59.49  DATE: Ø3.279
 PF 1 HELP 2 COM 3 END   CRSR 7 SBH 8 SFH 9 MSG 1Ø SB 11 SF 12 CNCL
```

On the other hand, entering mixed-case fields on the command line along with the CEDA transaction when the terminal is defined as having UCTRAN(YES) would not have the desired results. This is because terminal control applies the UCTRAN setting before CEDA gets control to process the keywords on the command.

Example 3: CEDA command:

```
ceda define program(jprog) group(test) jvm(yes) jvmclass(jclass)
jvmprofile(dfhjvmpr)
```

You need to actually enter the CEDA define or alter panel first in order for the new mixed-case logic to take effect. CEDA program logic determines the current UCTRAN setting, saves the value, and switches it to NO before reading the keyword values to be defined or altered. So, entering keywords to be set or altered via CEDA at the same time as entering the CEDA transaction, will not work, but will result in the UCTRAN setting being altered too late – after the keywords have been folded while being read on input. Note that in the example the jvmclass field would be folded to JCLASS and the jvmprofile field would be folded to DFHJVMPR. This may not have been the intention of the system programmer.

For further information about entering mixed-case input refer to the *Entering mixed case attributes* section of the *CICS Resource Definition Guide*.

CONCLUSION

CICS's CEDA transaction has been altered for CICS TS 2.3 to handle mixed-case input for fields that need to match the HFS file names or Unix-style input as required for the new EJB type resources. These fields are indicated on the CEDA panel by '(Mixed Case)' so that the user knows which fields will not automatically be folded to upper case by the terminal default folding mechanism. You no longer need to remember to first use the CEOT transaction to set UCTRAN(NO) for the terminal. The only way to take advantage of the new implementation is to enter the CEDA panel before making the changes to the mixed-case fields and not by entering the field alterations along with the transaction all in one command.

Arden Stewart (arden_stewart@uk.ibm.com)
CICS Development
IBM (UK)

Debugging CICS TS 2.2 Java applications using Java Platform Debugger Architecture

CICS TS 2.2 provided Enterprise Java Bean (EJB) support, which allows CICS to participate in the strategic solution for e-business as a full J2EE participant or as a back-end procedural application server. The IBM Strategy for e-business provides a complete architecture for developing and deploying e-business applications. New CICS TS Java applications can now take advantage of the Java standard debugging mechanism.

CICS TS 2.2 EJB server supports the Java Platform Debugging Architecture (JDPA), which is the standard debugging mechanism provided in the Java 2 platform. This architecture provides a set of APIs that allow the attachment of a remote debugger to a JVM.

A number of third-party debug tools are available that exploit JPDA and can be used to attach to and debug a JVM that is running an enterprise bean, CORBA object, or CICS Java program. The list of products using Java Platform Debugger Architecture can be found at http://java.sun.com/products/jpda/using.html.

Typically the debug tool provides a graphical user interface that runs on a workstation and allows you to follow the application flow, setting breakpoints and stepping through the application source code, as well as examining the values of variables.

END-TO-END DEBUGGING JAVA PLATFORM

The facilities that CICS TS V2.2 provides, in association with a client debugging tool, allow you to test a CICS enterprise bean as if it were running in the application development environment on your workstation.

JPDA enables a debugger client program on a remote workstation to control the CICS JVM execution. You can set breakpoints at individual method calls or lines within the code, and display or alter variables. This all happens in real time because the CICS JVM is actually executing the application classes' Java byte code.

Persistent reusable JVM supports the Java Debug Interface (JDI), the Java Debug Wire Protocol (JDWP), and the Java Virtual Machine Debugging Interface (JVMDI), which are covered in detail later in this article.

Any remote debugger built to exploit those architected interfaces can be used to debug EJBs running under CICS and any other Java program running under CICS in JVM mode.

Figure 1 shows how end-to-end debugging works.

An incoming request starts a request processor transaction. That request processor transaction specifies a request processor PROGRAM with a JVMPROFILE attribute that has special debug options causing a debug JVM to start.

The JVM starts debug threads to manage the debugging session. These threads use JVMDI to communicate with the JVM and control the execution of the application threads.

A remote client uses JDI to communicate with the debugging threads within the JVM. The actual communication between the debugger client and the remote JVM uses JDWP.

JPDA overview

JPDA consists of the following layered APIs:

- Java Debug Interface (JDI) – a Java programming language interface providing support for remote debugging. This is the highest-level interface in the architecture, and can be used to implement a remote debugger user interface without

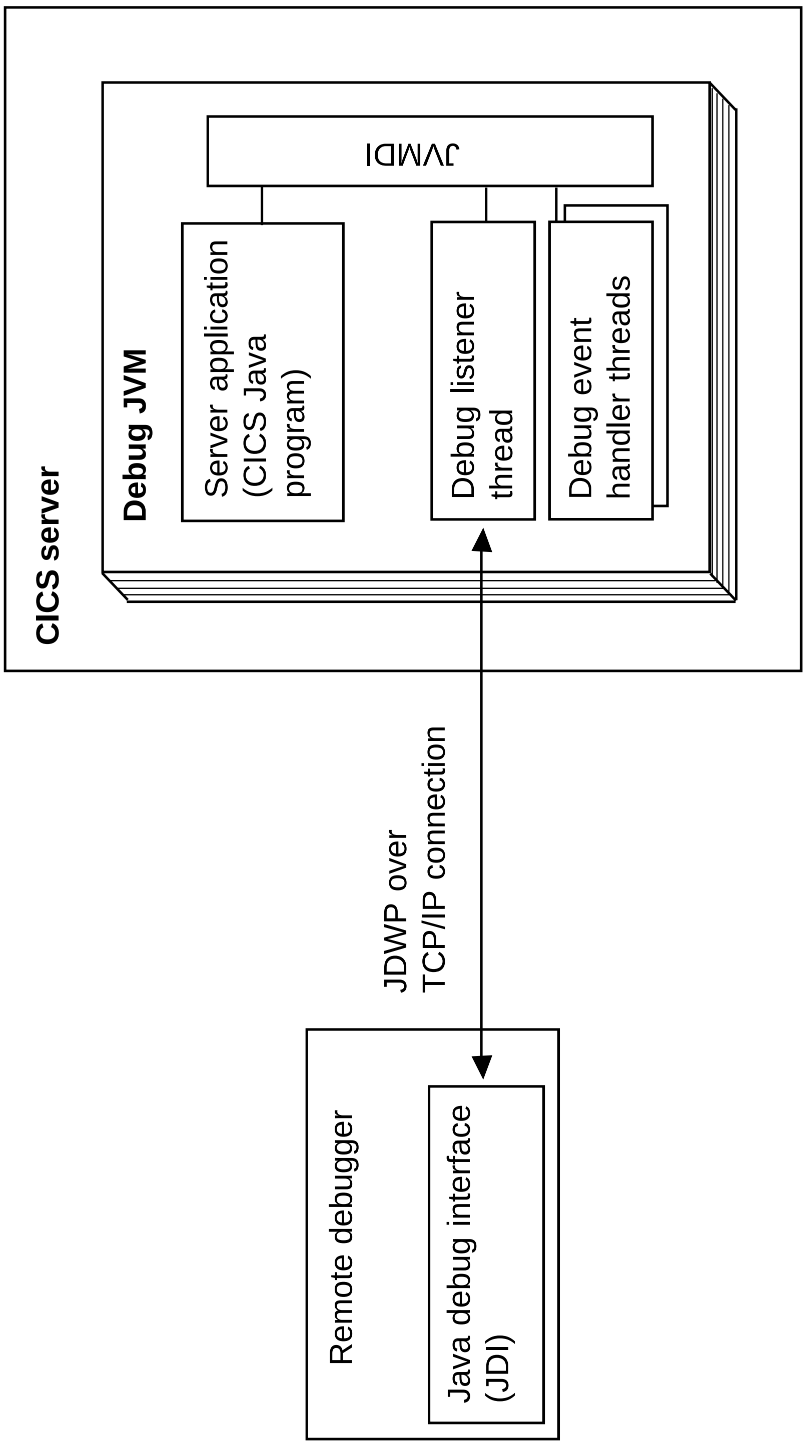
CICS server
Debug JVM
JVMDI
Server application (CICS Java program)
Debug listener thread
Debug event handler threads
JDWP over TCP/IP connection
Remote debugger
Java debug interface (JDI)

Figure 1: End-to-end debugging

having to write any code that runs in the application JVM or understand the protocol between the debugger and the JVM. Most third-party debuggers that support JPDA currently use this API.

- Java Debug Wire Protocol (JDWP) – this API defines the format of the flows that run between the application JVM and the debugger user interface. This protocol is for use by debuggers that need to exploit the communication at a lower level than the JDI, and for JVM suppliers or more advanced debugger developers who need to support the standard connection architecture from the application JVM side.
- Java Virtual Machine Debug Interface (JVMDI) – a low-level native interface within the JVM. It defines the services a Java virtual machine must provide for debugging, and can be used by advanced debugger developers who wish to implement debugger code that runs inside the application JVM.

When you start the JVM in debug mode, the JVMDI interface is activated and additional threads are started in the JVM. One of these threads handles communication with the remote debugger, the others monitor the application that is running in the JVM.

For more information on Sun Java Platform Debugger Architecture go to http://java.sun.com/products/jpda.

Why do we need three different interface layers in JPDA?

We need three different interface layers in JPDA to make the debugging process easier.

It is possible to use JPDA to do cross-platform remote debugging using only JDWP, but writing directly to JDWP is very difficult and undesirable for most developers.

JDI is provided to make interfacing to JDWP easier. Not only does JDI format data to be sent across the wire and parse incoming data, but it provides queueing, cacheing, connection initiation, and many other services. And all this functionality is available from an easier-to-use Java programming language interface.

JVMDI insulates Java virtual machine implementers from the intricacies of the ‘debuggee’ side of JDWP and also simplifies the process, making the developer’s testing less ‘painful’, without worrying about sending and receiving the data in precise format.

DEBUGGING AN ENTERPRISE BEAN USING JPDA

The steps necessary to debug a Java program are:

1. Compile the Java program to be debugged with the debug option.
2. Configure the enterprise bean JAR file in CICS to use a request processor program that has a JVM profile with the special debug options.
3. Make the Java source code of the class to be debugged available to the debugger.
4. Use a JPDA debugger client to connect to CICS and debug the program.

ATTACHING A DEBUGGER TO CICS JVM

To run a JVM in debug mode and allow a JPDA remote debugger to be attached, some JVM initialization options must be set. The required options are specified for a CICS JVM via a JVM profile in the DFHJVM data set. The specific options required for debugging are as follows:

- Xdebug=YES – this is needed to start the JVM in debug mode.
- Xnoagent=YES – this option disables an emulation mode that the JVM provides for compatibility with older (Java 1.1.*x*) debuggers. This emulation mode is not supported.
- Xrunjdwp=(suboption=...,suboption=...) – this option specifies the details of the connection between the debugger and the

CICS JVM. These details include the TCP/IP address to be used for the connection, and the sequence in which connection occurs. Different debuggers have different connection requirements and capabilities; refer to the documentation provided with the debugger.

- Xresettable=NO – a JVM that has been run in debug mode is not a candidate for reuse. Set this option to NO to ensure that the JVM is discarded after the debug session.

CREATE A SEPARATE JVM PROFILE SPECIFICALLY FOR DEBUG USE

Before you start setting up the required options for the JVM to use JPDA, it is important to realize that you do not want to make these changes to the default profiles DFHJVMPR and DFHJVMPS, which are used by internal CICS Java programs.

You should create a separate JVM profile specifically for debug use and ensure that it applies only to programs that you wish to debug. The CICS JVM program that will use this profile will run in debug mode and can go into a wait state, attaching to a debugger. You should then set the CICS program definition for DFJIIRP to use this debug JVM profile. The definitions required are outlined below.

AN EXAMPLE OF SETTING UP THE REQUIRED CICS DEFINITIONS

Figure 2 shows what CICS parameters are used by JPDA and how this fits into the overall picture.

Figure 2 illustrates an incoming request from the Java trader client application that starts a request processor transaction (DBUG) because of the request model DBUG001 matching all methods. The transaction (DBUG) specifies a request processor PROGRAM (DFJIIDB) with a JVMPROFILE (DFHJVMDB) that has special debug options, causing a debug JVM to start. The JVM starts debug threads to manage the debugging session. These threads use the JVMDI to communicate with the JVM and control the execution of the application threads. A remote client uses JDI to communicate with the debugging threads within the JVM. The actual communication between the remote debugger and the remote JVM uses JDWP on port 08099.

You would need to set up the following CICS definitions to follow the set-up above:

1 Define CORBASERVER with CEDA:

```
     CORbaserver :   DBUG
     Group :         TESTDBUG
     DEscription :   CORBASERVER for EJB debugging
     Jndiprefix :    itbocicb/PJA1/DEBUG/
     Autopublish :   Yes
     SHelf :         /usr/lpp/cicsts22/shelf
     DJardir :       /usr/lpp/cicsts22/pkupdbug
SERVER ORB ATTRIBUTES
     Host :          your.hostname.com
 TCPIP SERVICES
     Unauth :        DEBUGTCP
```

Notes:

- Place the Jar file in the pick-up directory /usr/lpp/cicsts22/pkupdbug:

```
Type Perm      Changed  (GMT) Owner    Size    Fil          Row
_    File 775 7/28/2ØØ3 18:46 OMVSKERN 135435Ø test-dbug.jar
```

- Use **CEMT I DJAR** to check the status of the CORBASERVER you have defined:

```
STATUS:                RESULTS
Djar(test-dbug )       Corba(DBUG)       Inser
Dates(2ØØ3Ø2Ø9)        Times(18:46:11)   Hfsfi(/usr/lpp/cicsts22)
```

2 Define REQUESTMODEL definition with CREA transaction:

```
CREA       REQUESTMODEL Definition Transaction
Please enter the name of a DJAR you want to work with.
DJAR name: test-dbug
```

If you want to restrict the display of beans and methods within the DJAR to those associated with a particular transaction then you can use the tranID filter. The * character can be used as a wildcard:

```
TranID filter:
CREA       REQUESTMODEL Definition Transaction       applid
```

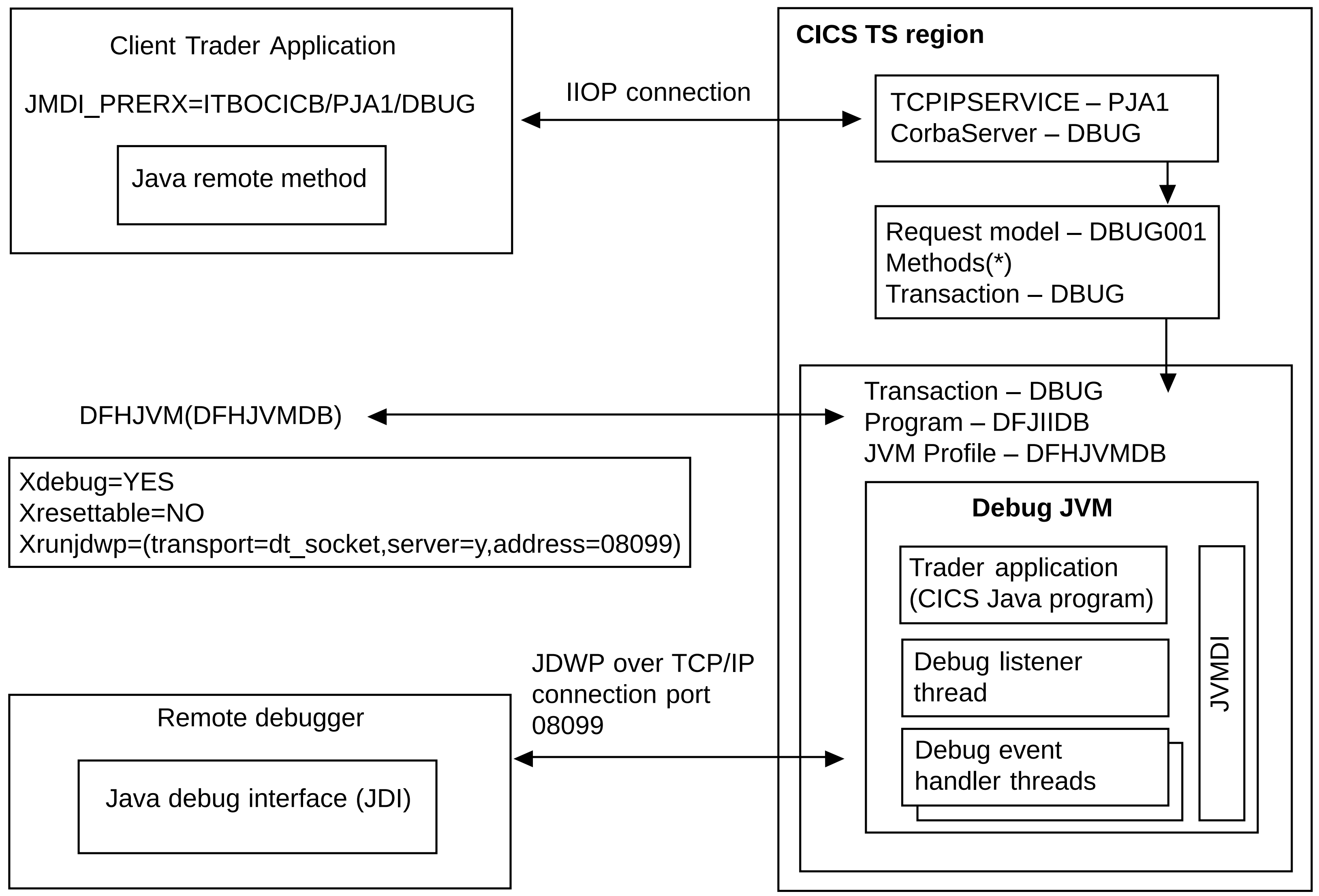

Figure 2: CICS parameters and JPDA

Associate a transaction ID with a bean or bean method:

```
DJAR name:        test-dbug              TranID filter:      DBUG
TranID            Bean/Method/Parameters
CREA              TestProgram
CREA        >     TestProgram
                   java.lang.String
                   java.util.Date
CREA        >     getEJBHome (inherited from EJBObject)
CREA        >     getHandle (inherited from EJBObject)
CREA        >     getPrimaryKey (inherited from EJBObject)
CREA        >     isIdentical (inherited from EJBObject)
                  javax.ejb.EJBObject
CREA        >     remove (inherited from EJBObject)

PF1=Help       3=Quit       5=Create ReqMods         9=Hide Methods
```

- Select option 5 to **Create REQUESTMODEL**:

```
CREA        REQUESTMODEL Definition Transaction       applid
```

Create a REQUESTMODEL definition DBUG001:

```
Group:             TESTDBUG  Define to CSD: Y  Install to CICS: Y
Replace Dups:       N
REQUESTMODEL name: DBUGØØ1   CORBASERVER:       DBUG

TranID:       DBUG

Bean:       TestProgram

Operation:       *

PF1=Help       3=Quit       8=Skip ReqMod       ENTER=Create ReqMod
```

- Select the **Create REQUESTMODEL** option, which will create the following RDO definition:

```
      CEDA  View  Requestmodel(  DBUGØØ1  )
     Requestmodel : DBUGØØ1
     Group :        TESTDBUG
     Description :  Request model for DBUG Corbaserver with DBUG tran-id
     Corbaserver :  DBUG
     TYpe :         Ejb
 EJB PARAMETERS
     Beanname :     TestProgram
TRANSACTION ATTRIBUTES
     TRansid :      DBUG
```

- Check its status with the **CEMT I REQUESTMODEL** command:

```
    Req(DBUGØØ1 )      Corbas(DBUG)      Ejb      Tra(DBUG)      Bot
    Bea(TestProgram )
```

3 Define the DBUG transaction using CIRP as a model:

```
CEDA COPY TRAN(CIRP) G(DFHIIOP) TO(TESTDBUG) AS(DBUG)

 TRANSaction :      DBUG    <- new transaction, RACF needs to be updated
       Group :      TESTDBUG
       DEscription   ==>    Debug IIOP Req Processor copied from CIRP ->
prog DFJIIRP
       PROGram       ==>    DFJIIDB
```

4 Define the DFHJIIDB program, using DFJIIRP as a model:

```
CEDA COPY PROG(DFJIIRP) G(DFHIIOP) TO(TESTDBUG) AS(DFJIIDB)

    PROGram :     DFJIIDB    <- new JVM program definition, was DFJIIRP
    Group :       TESTDBUG
    DEscription     ==>       CICS IIOP Request Processor for debugging
    Language        ==>       Le37Ø
 JVM ATTRIBUTES
    JVM             ==>       Yes
    JVMClass        ==>       com.ibm.cics.iiop.RequestProcessor
    JVMProfile      ==>       DFHJVMDB
   ¯ (step5 creates it)
```

Check its status with the **CEMT I PROG(DFJIIDB)** command:

```
    Prog(DFJIIDB )      Le3       Pro       Ena       Res(ØØØ)
            Any         Uex       Ful       Thr       Jvm
```

5 Create a new JVM profile, DFHJVMDB, which the DBUG transaction points to:

```
    CICSTS22.XDFHENV(DFHJVMDB)

    #
    # DO NOT SET THESE OPTIONS IN PRODUCTION
    #
    Xdebug=YES
    Xnoagent=YES
    Xresettable=NO
    Xrunjdwp=(transport=dt_socket,server=y,address=8Ø99)
    ¯ (step 6 defines it)
```

6 Define the IIOP port that is defined on the Xrunjdwp JVM parameter:

```
    TCpipservice :  DEBUGTCP
    GROup :         TESTDBUG
    DEscription :   TCPIPSERVICE FOR DEBUGGING JVM WITH JPDA
    POrtnumber :     Ø8Ø99
    STatus :         Open
    PRotocol :       Iiop
    TRansaction :    CIRR
```

Check its status with the **CEMT I TCPIPSERVICE** command:

```
Tcpips(DEBUGTCP)  Bac(  ØØØØ5 )    Con(ØØØØ)   Por(Ø8Ø99)  Iiop    Nos
      Ope         Tra(CIRR)        Ipa(xxx.xxx.xxx.xxx )       Wai
```

DEBUGGER ATTACH AND BREAKPOINTS SETTING

Once you have all the required CICS definitions in place and installed, the debugger should give an indication that it has successfully attached to the CICS JVM. The initial state of the JVM (such as the identity of threads that have started, and system classes that are loaded) is visible in the debugger user interface. The JVM will have suspended execution, and the Java application in CICS (enterprise bean, CORBA object, or CICS Java program) will not yet have started.

Your next action would be to set a breakpoint at a suitable point in the Java application by specifying the full Java class name and source code line number. As the application class will not usually have been loaded at this point, the debugger indicates that activation of this breakpoint is deferred until the class is loaded. You should then let the JVM run through the CICS middleware code to the application breakpoint, at which point it suspends execution again. You can then examine loaded classes and variables, set further breakpoints, and step through code as required.

To terminate the debug session you can let the application run to completion, at which point the connection between the debugger and the CICS JVM closes. Some debuggers support forced termination of the JVM, but it should not be used and will result in an abend and error messages on the CICS system console.

To fully enable the capabilities of a Java source code debugger, the Java code to be debugged must be compiled using the **-g** option on the Java compiler (javac command). Additional symbolic information is then preserved in the .class file, which is used when the debugger is attached at run-time. IDEs usually support this compiler option through a user setting (for example VisualAge for Java provides an option that can be set when a compiled application is exported from the IDE).

THE CICS JVM PLUG-IN MECHANISM

In addition to the standard JPDA debug interfaces in the JVM, CICS provides a set of interception points in the CICS Java middleware, which can be of value to the developers of debugging applications. These interception points allow additional Java programs to be inserted immediately before and after the application Java code is run. Information about the application is made available to the plug-in programs. The plug-in programs can also use the JCICS API to obtain information about the application. These interception points can be used in conjunction with the standard JPDA interfaces to provide additional CICS-specific debug facilities.

There are three Java exit points:

- A CICS EJB container debug plug-in providing methods that are called immediately before and after an EJB method is invoked.
- A CICS CORBA debug plug-in providing methods that are called before and after a CORBA method is invoked.
- A CICS Java wrapper plug-in providing methods that are called immediately before and after a CICS Java program is invoked.

In order to activate a debug plug-in implementation, you need to set one or more of the following system properties:

- **com.ibm.cics.server.debug.EJBPlugin**=<classname>

  This is the EJB container debug plug-in. If this is set, the supplied plug-in is registered by Java code in the CICS EJB server layer when the EJB container is initialized.

- **com.ibm.cics.server.debug.CORBAPlugin** =<classname>

  This is the CORBA debug plug-in. If this is set, the supplied plug-in is registered by Java code in the CICS ORB when the ORB is initialized.

- **com.ibm.cics.server.debug.WrapperPlugin** =<classname>

  This is the CICS Java debug plug-in. If this is set, the supplied plug-in is registered by additional Java code in the JCICS wrapper when the CICS Java program is run.

IBM DISTRIBUTED DEBUGGER

The CICS support for JPDA has been tested by IBM using the IBM Distributed Debugger V9.

The IBM Distributed Debugger consists of a graphical user interface that communicates with a remote debug engine, which interacts with the application being debugged. The user interface

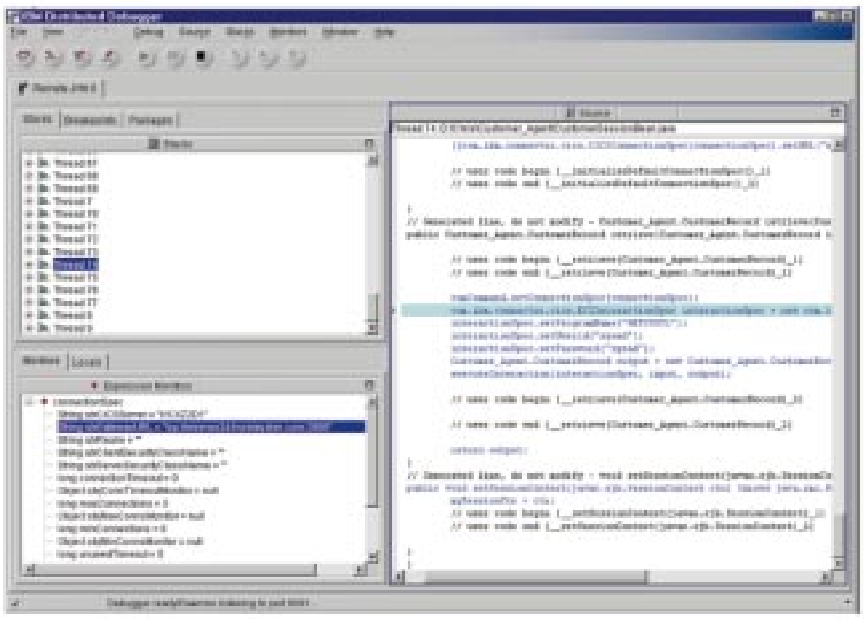

Figure 3: The debugger debugging a session bean

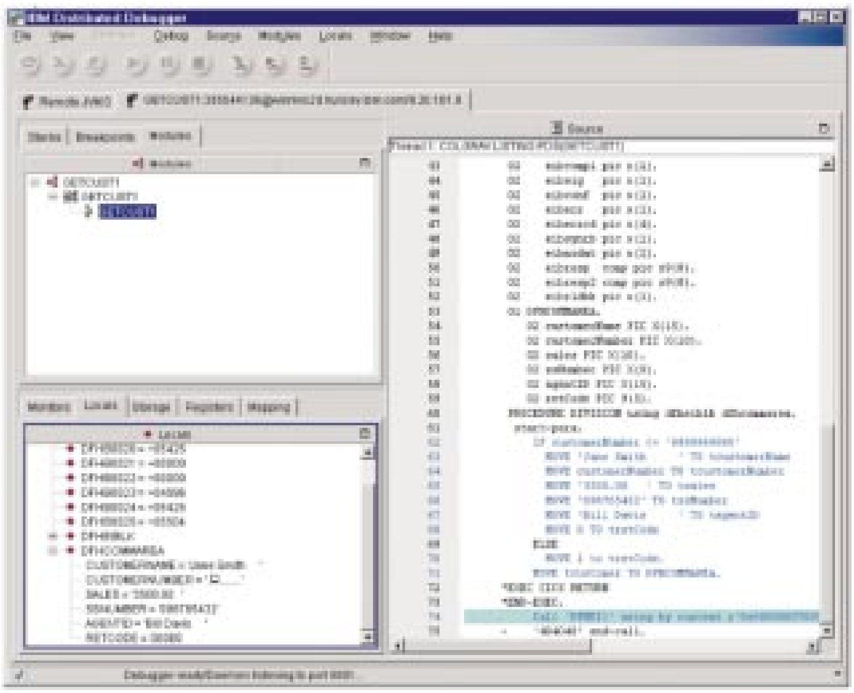

Figure 4: Debugging a CICS COBOL program

can display multiple windows, each of which corresponds to a particular debug session with a particular system. This facility can be used to debug a servlet or JSP file in addition to an enterprise bean invoked by the servlet or JSP. The debugger is not limited to debugging Java, it supports debugging code written in C, C++, and COBOL.

Figure 3 shows the debugger debugging a session bean.

Figure 4 shows the debugger running on Windows NT debugging a CICS COBOL program. Note that the COBOL program is shown in a second window, and the first window showing the session bean is still available so that the developer can switch between the Java code and the COBOL code.

OBJECT-LEVEL TRACE

Another tool that can be used for debugging is the Object-Level Trace (OLT) tool, which allows a developer to follow the flow of an application from the client to a server and from one server to

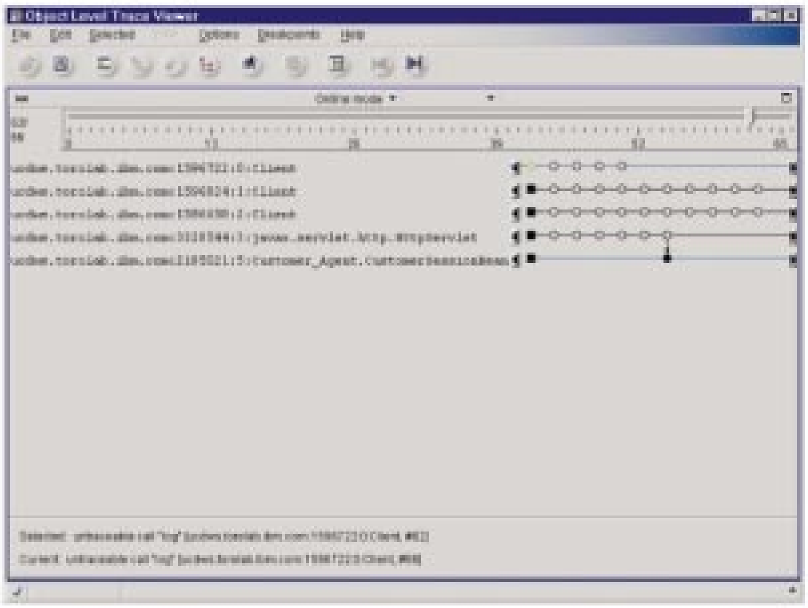

Figure 5: A sample of the OLT user interface

another server. The client and the servers send information about the inbound and outbound method calls to an OLT server, and a graphical tool presents a view of the distributed thread of control made up of all of the (remote) method calls. In addition to visualizing the flow of a distributed application, the OLT tool can be used to control the points in the distributed application at which the IBM Distributed Debugger should be invoked to remotely debug a particular method invocation. This provides much better control of the debugger than the typical server-oriented configuration schemes such as DTCN, a CICS transaction that configures the z/OS debug tool.

Figure 5 shows a sample of the OLT user interface.

The information controlling the debugger flows together with an object invocation over IIOP as an extra service context. This information is read and written by request interceptors running in the ORB. The interfaces to allow such request interceptors to be registered with the ORB are documented so ISVs can register their own request interceptors to flow their own additional information on method calls and support tools similar to OLT.

OTHER DIAGNOSTIC AIDS

Last but not least, when debugging CICS Java programs, the following diagnostic aids should not be overlooked:

1 CICS job log and console messages.

2 Java stdout and stderr files, written to the CICS region's work directory in the HFS. Any uncaught exceptions that are thrown while executing a Java program cause a stack trace to be written to the Java stderr file. This is one of the first places that needs to be checked when you experience unexpected problems with a Java program.

   Notes:

   - The stdin/stdout/stderr files are standard Java files that are located in the directory specified by the WORK_DIR parameter of the JVM profile.
   - In order to keep a permanent log of JVM messages you can use the **-generate** option on the STDOUT and STDERR parameters in the JVM profile as follows: **STDOUT=dfhjvmout -generate STDERR=dfhjvmerr -generate** This will cause the CICS APPLID and a unique time stamp to be appended to each file created.

3 The system properties event logging file contains useful information about the parameters being used for the JVM initialization, such as the Java build level, the class and lib paths, and the trusted middleware class path that is generated by CICS. You can use this information to verify that the JVM was using the profiles and system properties file expected and that the various directory paths were being built correctly. It can be set as follows: **ibm.jvm.events.output=user.log**.

4 The JVM profile VERBOSE parameter indicates whether or not the JVM should issue messages containing information about its activities. The information gets written to the Java stderr file. You can also use VERBOSE=class to aid with diagnosing some problems. This causes messages to be written as each class is loaded and initialized. It can

sometimes help with locating precisely where a problem is occurring.

5 The system properties JVM trace. The ibm.dg.trc.external parameter of the system property file enables the internal JVM trace facility to aid in the diagnosis of problems within the Java Virtual Machine itself.

6 CICS auxiliary trace.

7 EDF of the request processor (by default CIRP) for an enterprise bean, using the CEDX transaction.

CONCLUSION

IBM has come a long way with CICS Java and EJB support and has provided sufficient diagnostic aids to assist developers in the problem determination process. By supporting EJBs (and providing tools to manage it), CICS enables the seamless inclusion of an enterprise's investment in CICS into the IBM Application Framework for e-business. As a result of this, CICS continues to be a highly scalable application server for running mission-critical business logic, running 30 billion transactions daily.

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

Newcopy in a CICSPlex environment – part 2

This month we conclude the code that will help to newcopy the same program in all AORs.

```
     IF EIBAID = DFHPF3          OR
                 DFHCLEAR
        MOVE     END-MESSAGE      TO   PAUSE-MESSAGE
        PERFORM END-PGM          THRU EX-END-PGM.
     PERFORM SEND-MAP1        THRU EX-SEND-MAP1.
 EX-RECEIVE-HLP.
      EXIT.
***************************************************************
**     E N D    P R O G R A M                              ***
***************************************************************
 ASSIGN-CICS.
     EXEC CICS ASSIGN APPLID(W-APPLID)
                      USERID(W-USERID)
     END-EXEC.
***************************************************************
* move context value of CICSPLEX
***************************************************************
     MOVE  'CXPLPROD'     TO   MP1CTXO  MP2CTXO.
     MOVE  'CXPLPROD'     TO   W-CONTEXT.
     IF  EIBRESP NOT EQUAL Ø
         MOVE  W-PROGRAM      TO   WTO-PGM
         MOVE  ' - ESITO KO ' TO   WTO-ESITO
         MOVE KO-MESSAGE      TO   PAUSE-MESSAGE
         PERFORM  WRITE-QUE  THRU EX-WRITE-QUE
         PERFORM  END-PGM    THRU EX-END-PGM.
     EXEC  CICS  ASKTIME  ABSTIME(ABS-TIME)  END-EXEC.
     EXEC  CICS  FORMATTIME  ABSTIME(ABS-TIME)
                             YYMMDD(DATA-OGGI)
                             TIME(ORA-OGGI)
     END-EXEC.
     IF  EIBRESP NOT EQUAL Ø
         MOVE  W-PROGRAM      TO   WTO-PGM
         MOVE  ' - ESITO KO ' TO   WTO-ESITO
         MOVE KO-MESSAGE      TO   PAUSE-MESSAGE
         PERFORM  WRITE-QUE  THRU EX-WRITE-QUE
         PERFORM  END-PGM    THRU EX-END-PGM.
     MOVE     HH-OGGI     TO    WTO-HH.
     MOVE     MN-OGGI     TO    WTO-MN.
     MOVE     SS-OGGI     TO    WTO-SS.
     MOVE     AA-OGGI     TO    WTO-AA.
     MOVE     MM-OGGI     TO    WTO-MM.
     MOVE     GG-OGGI     TO    WTO-GG.
 EX-ASSIGN-CICS.
     EXIT.
 WRITE-QUE.
     MOVE  W-PROGRAM      TO    WTO-PGM
     MOVE     W-USERID    TO    WTO-USER.
     MOVE     W-PROGRAM   TO    WTO-PGM.
     EXEC CICS WRITEQ TD QUEUE('CSMT') FROM(MSG-WTO)
          LENGTH(LENG-WTO)
     END-EXEC.
 EX-WRITE-QUE.
        EXIT.
```

MPSCPSM

```
MPSFILE  DFHMSD TYPE=&SYSPARM,MODE=INOUT,                              *
               LANG=COBOL,TIOAPFX=YES,STORAGE=AUTO,EXTATT=YES
MPSØ1    DFHMDI SIZE=(24,8Ø)
         DFHMDF POS=(Ø1,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),               *
               INITIAL='************************************************
               *******************************'
         DFHMDF POS=(Ø2,Ø1),LENGTH=71,ATTRB=(ASKIP,BRT),               *
               INITIAL='*              Utility for a NEWCOPY of program*
               s in a CICSPlex         '
         DFHMDF POS=(Ø2,73),LENGTH=Ø2,ATTRB=ASKIP
         DFHMDF POS=(Ø2,76),LENGTH=Ø4,ATTRB=(ASKIP,BRT),INITIAL='   *'
MP1TES   DFHMDF POS=(Ø3,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),               *
               INITIAL='*----------------------------------------------*
               ------------------------------*'
         DFHMDF POS=(Ø4,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø4,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                           '
         DFHMDF POS=(Ø4,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø5,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø5,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '

         DFHMDF POS=(Ø5,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø6,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø6,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                   Context ....: '
MP1CTX   DFHMDF POS=(Ø6,38),LENGTH=Ø8,ATTRB=(UNPROT,BRT,FSET),         *
               INITIAL='CXPLPROD'
         DFHMDF POS=(Ø6,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø7,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø7,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '

         DFHMDF POS=(Ø7,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø8,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø8,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                   Scope ......: '
MP1SCO   DFHMDF POS=(Ø8,38),LENGTH=Ø8,ATTRB=(UNPROT,BRT,FSET),         *
               INITIAL='ALLAORS '
         DFHMDF POS=(Ø8,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø9,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø9,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '

         DFHMDF POS=(Ø9,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(1Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(1Ø,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                   Function ...: '
```

```
MP1FUN   DFHMDF POS=(1Ø,38),LENGTH=Ø8,ATTRB=(UNPROT,BRT,FSET),         *
               INITIAL='PHASEIN '
         DFHMDF POS=(1Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(11,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(11,Ø3),LENGTH=35,ATTRB=(ASKIP,BRT),               *
               INITIAL='                  (PHASEIN/NEWCOPY)'
         DFHMDF POS=(11,39),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='        '
         DFHMDF POS=(11,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(12,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(12,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                             '
         DFHMDF POS=(12,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(13,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(13,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    Program ....: '
MP1PGM   DFHMDF POS=(13,38),LENGTH=Ø8,ATTRB=(UNPROT,BRT,FSET,IC),      *
               INITIAL='________'
         DFHMDF POS=(13,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(14,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(14,Ø3),LENGTH=35,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    (PGMNAME/AABB*)'
         DFHMDF POS=(14,39),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='        '
         DFHMDF POS=(14,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(15,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(15,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                            '
         DFHMDF POS=(15,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(16,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(16,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                            '
         DFHMDF POS=(16,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(17,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(17,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                            '
         DFHMDF POS=(17,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(18,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(18,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                            '
         DFHMDF POS=(18,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(19,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(19,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                            '
         DFHMDF POS=(19,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(2Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(2Ø,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),              *
               INITIAL='                                              *
                           '
         DFHMDF POS=(2Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(21,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(21,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),              *
               INITIAL='                                              *
                           '
         DFHMDF POS=(21,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(22,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),              *
               INITIAL='*---------------------------------------------*
               ------------------------------*'
MP1MSG   DFHMDF POS=(23,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT)
         DFHMDF POS=(24,Ø1),LENGTH=79,INITIAL='Pf1 ==> Help, Pf3/Clear *
               End work, Enter ==> Select      ',ATTRB=(ASKIP,BRT)
MP1SEN   DFHMDI SIZE=(1Ø,8Ø)
         DFHMDF POS=(Ø1,Ø1),LENGTH=79,INITIAL='       PRESS  ENTER    *
                                              ',ATTRB=(ASKIP,BRT)
MPSØ2    DFHMDI SIZE=(24,8Ø)
         DFHMDF POS=(Ø1,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),              *
               INITIAL='***********************************************
               *********************************'
         DFHMDF POS=(Ø2,Ø1),LENGTH=71,ATTRB=(ASKIP,BRT),              *
               INITIAL='*            Utility for a NEWCOPY of program*
               s in a CICSPlex         '
         DFHMDF POS=(Ø2,73),LENGTH=Ø2,ATTRB=ASKIP
         DFHMDF POS=(Ø2,76),LENGTH=Ø4,ATTRB=(ASKIP,BRT),INITIAL='   *'
MP2TES   DFHMDF POS=(Ø3,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),              *
               INITIAL='*---------------------------------------------*
               ------------------------------*'
         DFHMDF POS=(Ø4,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø4,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),              *
               INITIAL='                                        '
         DFHMDF POS=(Ø4,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø5,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø5,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),              *
               INITIAL='                                              *
                           '
         DFHMDF POS=(Ø5,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø6,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø6,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),              *
               INITIAL='                    Context ....: '
MP2CTX   DFHMDF POS=(Ø6,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),              *
               INITIAL='CXPLPROD'
         DFHMDF POS=(Ø6,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø7,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø7,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),              *
               INITIAL='                                              *
```

```
                           '
         DFHMDF POS=(Ø7,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø8,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø8,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    Scope ......: '
MP2SCO   DFHMDF POS=(Ø8,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='ALLAORS '
         DFHMDF POS=(Ø8,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø9,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø9,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(Ø9,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(1Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(1Ø,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    Function ...: '
MP2FUN   DFHMDF POS=(1Ø,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='PHASEIN '
         DFHMDF POS=(1Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(11,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(11,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(11,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(12,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(12,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    Program ....: '
MP2PGM   DFHMDF POS=(12,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='        '
         DFHMDF POS=(12,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(13,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(13,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT)
         DFHMDF POS=(13,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='        '
         DFHMDF POS=(13,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(14,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(14,Ø3),LENGTH=34,ATTRB=(ASKIP,BRT),               *
               INITIAL='                    Number/Entry: '
MP2ENT   DFHMDF POS=(14,38),LENGTH=Ø8,ATTRB=(ASKIP,BRT),               *
               INITIAL='        '
         DFHMDF POS=(14,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(15,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(15,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(15,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(16,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(16,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(16,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(17,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(17,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(17,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(18,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(18,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(18,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(19,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(19,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(19,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(2Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(2Ø,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(2Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(21,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(21,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(21,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(22,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),               *
               INITIAL='*----------------------------------------------*
               ------------------------------*'
MP2MSG   DFHMDF POS=(23,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT)
MP2PFK   DFHMDF POS=(24,Ø1),LENGTH=79,INITIAL='Pf3 ==> End Session, Oth*
               er Pf Keys to Return Main Menu''  '
MPSHLP   DFHMDI SIZE=(24,8Ø)
         DFHMDF POS=(Ø1,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),               *
               INITIAL='************************************************
               *******************************'
         DFHMDF POS=(Ø2,Ø1),LENGTH=71,ATTRB=(ASKIP,BRT),               *
               INITIAL='*            Utility for a NEWCOPY of program*
               s in a CICSPlex          '
         DFHMDF POS=(Ø2,73),LENGTH=Ø2,ATTRB=ASKIP
         DFHMDF POS=(Ø2,76),LENGTH=Ø4,ATTRB=(ASKIP,BRT),INITIAL='   *'
         DFHMDF POS=(Ø3,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),               *
               INITIAL='*----------------------------------------------*
               ------------------------------*'
         DFHMDF POS=(Ø4,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIØ   DFHMDF POS=(Ø4,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='                                               *
                           '
         DFHMDF POS=(Ø4,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø5,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
```

```
MPHRI1   DFHMDF POS=(Ø5,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='This utility executes the newcopy command of on*
               e or more   '
         DFHMDF POS=(Ø5,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø6,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI2   DFHMDF POS=(Ø6,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='program in a CICSPlex envirnment. The newcopy  *
               is done     '
         DFHMDF POS=(Ø6,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø7,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI3   DFHMDF POS=(Ø7,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='in the same instant on the AORs defined to CICS*
               Plex        '
         DFHMDF POS=(Ø7,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø8,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI4   DFHMDF POS=(Ø8,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='In this way all mismatch problems about CICS re*
               ources      '
         DFHMDF POS=(Ø8,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(Ø9,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI5   DFHMDF POS=(Ø9,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='are avoided, in particular -818 sqlcode in DB2 *
               transaction '
         DFHMDF POS=(Ø9,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(1Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI6   DFHMDF POS=(1Ø,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='The Context and Scope are CICSPlex parameter an*
               d usually   '
         DFHMDF POS=(1Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(11,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI7   DFHMDF POS=(11,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='they haven't to be modified, instead the Action*
               and the     '
         DFHMDF POS=(11,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(12,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI8   DFHMDF POS=(12,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='Program Name have to be written in the map     *
                            '
         DFHMDF POS=(12,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(13,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRI9   DFHMDF POS=(13,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='We can ask a single program or all programs wit*
               h name start'
         DFHMDF POS=(13,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(14,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIA   DFHMDF POS=(14,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),               *
               INITIAL='by AAAA followed by * as wildcard (for example *
                PGMØ1*).    '
         DFHMDF POS=(14,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(15,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
```

```
MPHRIB   DFHMDF POS=(15,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(15,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(16,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIC   DFHMDF POS=(16,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(16,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(17,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRID   DFHMDF POS=(17,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(17,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(18,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIE   DFHMDF POS=(18,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(18,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(19,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIF   DFHMDF POS=(19,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(19,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(2Ø,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHRIG   DFHMDF POS=(2Ø,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(2Ø,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(21,Ø1),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
MPHSYH   DFHMDF POS=(21,Ø3),LENGTH=75,ATTRB=(ASKIP,BRT),                   *
               INITIAL='                                                   *
                   '
         DFHMDF POS=(21,79),LENGTH=Ø1,ATTRB=ASKIP,INITIAL='*'
         DFHMDF POS=(22,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT),                   *
               INITIAL='*----------------------------------------------*
               ------------------------------*'
MPHMSG   DFHMDF POS=(23,Ø1),LENGTH=79,ATTRB=(ASKIP,BRT)
MPHPFK   DFHMDF POS=(24,Ø1),LENGTH=79,INITIAL='Pf3 ==> End Session, Oth*
               er Pf Keys to Return Main Menu''  '
         DFHMSD TYPE=FINAL
         END
```

Marco Busichella
Systems Programmer (Italy)   © Xephon 2004

Build a homegrown adapter solution to enable CICS applications for eBusiness

BUSINESS SUMMARY

According to statistics from IBM and others, 30 years and more than $1 trillion has been invested in CICS applications, there are 14,000+ CICS customers worldwide, and there are more than 30 million end users of CICS applications. Considering these facts, all companies with large investments in CICS applications are building an ability to integrate with middle-tier servers to enable CICS for eBusiness. We are playing an important role in building the US Navy Marine Corps Intranet (NMCI) – this also requires building an adapter to integrate the Shipyard CICS applications.

SOLUTION

We have built a home-grown adapter that allows secure access to CICS transactions and delivers their output as a standard XML document with no screen-scraping or host application reengineering. This adapter can circumvent the use of 3270 data streams and, as a result, provides the functionality to leverage, enable, integrate, and scale the existing CICS applications for use with eBusiness applications.

The solution we have come up with solves the problem of data recognition by converting returned data from CICS applications into XML for use by the Web applications. Using XML preserves the business logic of the CICS applications and provides an industry-standard means of exchanging data between CICS and any other XML-enabled applications.

HOW DOES THE ADAPTER WORK?

The adapter runs on the mainframe under OS/390. It is built on the foundation of two features that IBM has added to CICS Transaction Server – CICS Web Support and 3270 Bridge.

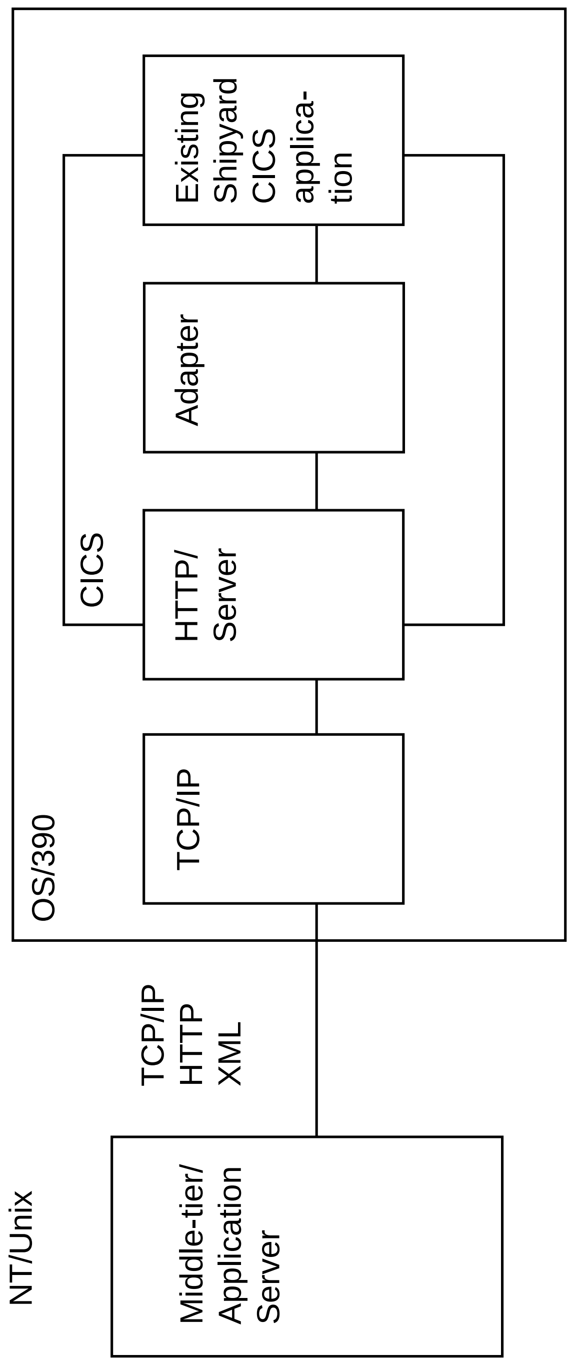
Existing Shipyard CICS applica- tion
Adapter
CICS
HTTP/ Server
OS/390
TCP/IP
TCP/IP HTTP XML
NT/Unix
Middle-tier/ Application Server

Figure 1: Basic system architecture

- CICS Web Support (CWS) enables an HTTP client (eg a Web browser) to communicate directly with mainframe CICS application programs without an intermediate gateway or a separate Web server (you can also communicate through the OS/390 HTTP Server or WebSphere 390).

- 3270 Bridge makes it possible to intercept the flow of data into and out of a CICS transaction before a 3270 data stream is generated as output or expected as input. 3270 Bridge works by intercepting the flow of control between the user transaction and BMS, thereby allowing another software component, such as the adapter we built, to handle input/output operations for the transaction. The vast majority of CICS transactions rely on a component within CICS called BMS (Basic Mapping Support) to interact with a 3270 terminal. When a transaction calls BMS, it specifies the name of a 3270 screen 'map' and a set of fields and values to be used with it. The map determines where on the screen the fields are to be placed and BMS generates the resulting 3270 data stream. BMS also accepts the 3270 terminal input and returns it to the program as values in the corresponding fields. Thus, an often-overlooked fact is that CICS transactions that use BMS to interact with a 3270 terminal 'think' in field name/value pairs and not in 3270 data streams.

ARCHITECTURE

Figure 1 shows the basic architecture of the system.

The basic components include a client application that resides on Unix or NT/2000. A transaction sequence using the adapter consists of the following steps:

1. The client application sends an HTTP request via the TCP/IP layer on the host.
2. An HTTP Listener within CICS Web Support (CWS) monitors TCP/IP and receives the request from TCP/IP.
3. If the Listener receives the HTTP request on a specified port, it passes the request to the adapter.
4. The adapter authenticates the request from the client. If the request from the client contains the correct authorization header, the adapter assigns a userid and begins a transaction with the host application under that userid.

5 The user transaction returns the requested data to the adapter.

6 The adapter converts the data to XML and passes it to CICS Web Support (CWS).

7 CWS then returns an HTTP response to TCP/IP.

8 TCP/IP then sends the HTTP response with the XML data back to the client application.

External applications can also pass requests over ECI and EXCI to invoke the adapter using a LINK interface. This article will use examples showing the HTTP interface, but the concepts presented are the same for both interfaces.

ADAPTER XML CONVERSION

```
                    Inventory Management Demonstration            SHP.SØØ4


    User name: WELCH

    Vessel Name: Supship_Ingleside_n68958
Item:                         Storage Unit:                Level Indicator:

 334553                       A445                         AVAILABLE
 334939                       A8893                        NOTAVAILABLE
 93ØØØ3                       B8839                        LOWERLIMIT


Request Completed : OK

PF3=Return                                                          PF12=Exit
```

In the screen above, a port engineer named Welch is retrieving the inventory for certain items in Vessel SupShip_Ingleside. The adapter identifies each screen and field in a CICS application. Notice the *User Name* field with the Value *WELCH*. Take a look below at how the adapter displays the CICS data in XML after an external application sends the following HTTP request to the mainframe Web server:

```
http://cdms.navsea.navy.mil:[port]/ams_adapter?BH_TRANID=[CICSID]
```

Below is the adapter output sent back to the requesting application:

```
<?xml version="1.Ø" encoding="ISO-8859-1" ?>
<!— AMS Copyright 2ØØØ, 2ØØ3 American Management Systems,Inc, U.S.
Patent Pending —>
<AMS>
<token>73722b93</token>
<timestamp>2ØØ3Ø7Ø2154224</timestamp>
<status>
<response>Ø</response>
<desc>ok</desc>
</status>
<transaction facility="}AAB" next_tranid="TRAD">
<status>
<cics_resp>Ø</cics_resp>
<cics_resp2>Ø</cics_resp2>
<cics_desc>ok</cics_desc>
<task_end>endtask</task_end>
<abend_code />
</status>
<parameters>
<tranid>TRAD</tranid>
<userid>DSI1</userid>
</parameters>
<command>
<send_map erase="y" erase_unp="n" unlock_kb="y" alarm="n" reset_mdt="n">
<mapset>TRADBMS</mapset>
<map>TØØ4</map>
<data_indicator>map_and_data</data_indicator>
<fields count="15">
<field name="USER41" index="Ø">
<name len="6">USER41</name>
<value maxlen="2Ø" len="4">WELCH</value>
<attr byte="ØØ" justify="l" disp="n" prot="n" num="n"
int="n" mdt="n" />
</field>
.
.
.
<field name="MESS4" index="Ø">
<name len="5">MESS4</name>
<value maxlen="79" len="2Ø">Request Completed OK</value>
<attr byte="ØØ" justify="l" disp="n" prot="n" num="n" int="n" mdt="n" />
</field>
</fields>
</send_map>
</command>
</transaction>
</AMS>
```

The entire contents of a transaction appears between the <transaction></transaction> tags. Notice that the tags

<name>USER41</name> and <value>WELCH</value> identify the *User Name* field indicated in the screen above. This information, generated in XML format by the adapter, is not only descriptive enough for humans to understand but the important point is that B2B and EAI tools can easily find and extract the data they need.

ACCESSING CICS APPLICATIONS WITH THE ADAPTER

To interact with the CICS application, the HTTP request needs to include several pieces of information. Let's look at a URL that could be sent to the CICS application and change the contents of the USER41 field from *WELCH* to *MICHAEL*.

```
http://company.com:4Ø41/ams?AMS_TOKEN=73722b93&USER41=MICHAEL
```

The different parts of the URL perform the following:

- *http://company.com: 4041/* identifies the address and port on which the adapter runs. The HTTP listener passes URLs on this port to the adapter.
- *AMS?* identifies this as an adapter session. Everything that follows the '?' in the URL is the command string that controls the adapter's behaviour.
- *AMS_TOKEN=73722b93* after an initial connection to the host application, the adapter returns a session ID called a state token, seen in the XML above as:

  ```
  <token>73722b93</token>
  ```

  Subsequent transactions with the host must include this token in the URL so CICS will recognize the transaction as part of an existing session.
- *USER41=MICHAEL* sends the field name and the data value we want to enter in the transaction. The adapter interprets any name/value pair sent along the URL as input to a field unless the name starts with **AMS_** (eg AMS_TOKEN).

The adapter can also receive requests in the form of XML documents that conform to a fixed schema. The following XML document submits the same transaction data as the URL above:

```
<?xml version="1.0" ?>
<AMS>
<transaction>
<parameters>
<ams_token>73722b93</ams_token>
</parameters>
<fields>
<user41>MICHAEL</user41>
</fields>
</transaction>
</AMS>
```

We are currently working on this adapter to support SOAP messages as input.

FLEXIBILITY AND EASE OF USE

Using XML to allow external applications to access host data using tags provides several key benefits in a production environment.

Screen-scrapers create dependencies between CICS applications and the client applications that access them. If a change in the host application shown above shifts the position of the field labelled USER41, the CICS developers must document the changes for the Web application developers so that they can modify the screen-scraping tool to grab data from the new field location. If the Web developers do not change their applications, the changes in the host application will create errors when the Web application scrapes the screens looking for the fields in the old locations. This type of adapter can remove this dependency by converting identifying elements for host fields – in this case <name>,USER41</name> – regardless of their physical location. Therefore, Web applications simply find the information for the fields they need in the XML files.

SCALABILITY

Because it runs on the mainframe, there is no need to incur overhead by passing data through SNA stacks and HLLAPI running on an NT/2000 or Unix box.

SECURITY OVERVIEW

Running a CICS transaction using CICS Web Support and the XML application connector is as secure as running the transaction from a 3270 terminal. The XML application connector works with standard mainframe and Internet security methods, so it will work within whatever security model you have in place at your site to protect your applications and data.

This adapter combines standard mainframe and Internet security methods to create a security model that ensures that your applications and data are protected from end-to-end transactions:

- Username/password protection through RACF, ACF2, and Top Secret maintains your existing host security that limits user access to resources based on user ID authorizations.
- Client authentication through CICS Web Support or OS/390 Unix System Services ensures that clients and external applications are authorized to connect to the host applications.
- Data encryption with Secure Sockets Layer (SSL) protects data as it passes between the Web server and the client application using 56-bit or 128-bit encryption.

The information below provides a more detailed overview of the technical aspects of the adapter security model.

CICS WEB SUPPORT VERSUS OS/390 WEB SERVER

CICS Web Support is part of CICS TS 1.3. It enables HTTP clients, such as Web browsers, to communicate directly with mainframe CICS application programs without an intermediate gateway or a separate Web server. CWS includes an HTTP server that determines how to process HTTP requests based on the port number on which the request connects. Rather than entering the CICS Web Support environment directly through its integrated HTTP server, you can also enter through an OS/390

Web Server, such as Domino Go. The OS/390 Web Server runs on the Unix System Services (USS) platform and can also provide secure access to CICS services.

The OS/390 Web Server includes some functionality that CICS Web Support does not. For example, OS/390 Web Server supports logging, filtering, and redirection. However, beginning with CICS Transaction Server 1.3, CICS Web Support provides the same secure access possibilities as when going through an OS/390 server. Both provide better access control than when using real or emulated 3270 terminals.

CREATING A SECURE TRANSACTION ENVIRONMENT

Effective authorization and access control requires that every transaction runs under a user ID. The user ID then becomes the basis for performing all kinds of security checking. Most common is resource security, where the user ID is used for checking that a user is authorized to use CICS resources that the currently executing transaction uses. Examples of CICS controlled resources are files, transient data and temporary storage queues, referenced journals, and other transactions that an application wants to start. To create and enforce a secure environment, CICS relies on an External Security Manager (ESM) such as RACF.

SECURITY IN THE 3270 TERMINAL ENVIRONMENT

In a terminal environment, if a user is not signed on, the transactions started from that terminal run under the user ID of the 'default user'. This user ID is specified in the System Initialization Table (SIT) using parameter DFLTUSER. Typically, this user has limited (or no) authority and cannot start transactions. Thus, in a secure environment, users must sign on with the CESN transaction, which allows them to enter a user ID and a password. Another way to sign on is to execute an installation-written transaction containing an EXEC CICS SIGNON command. All transactions started from the terminal by the signed-on user will run under this user ID.

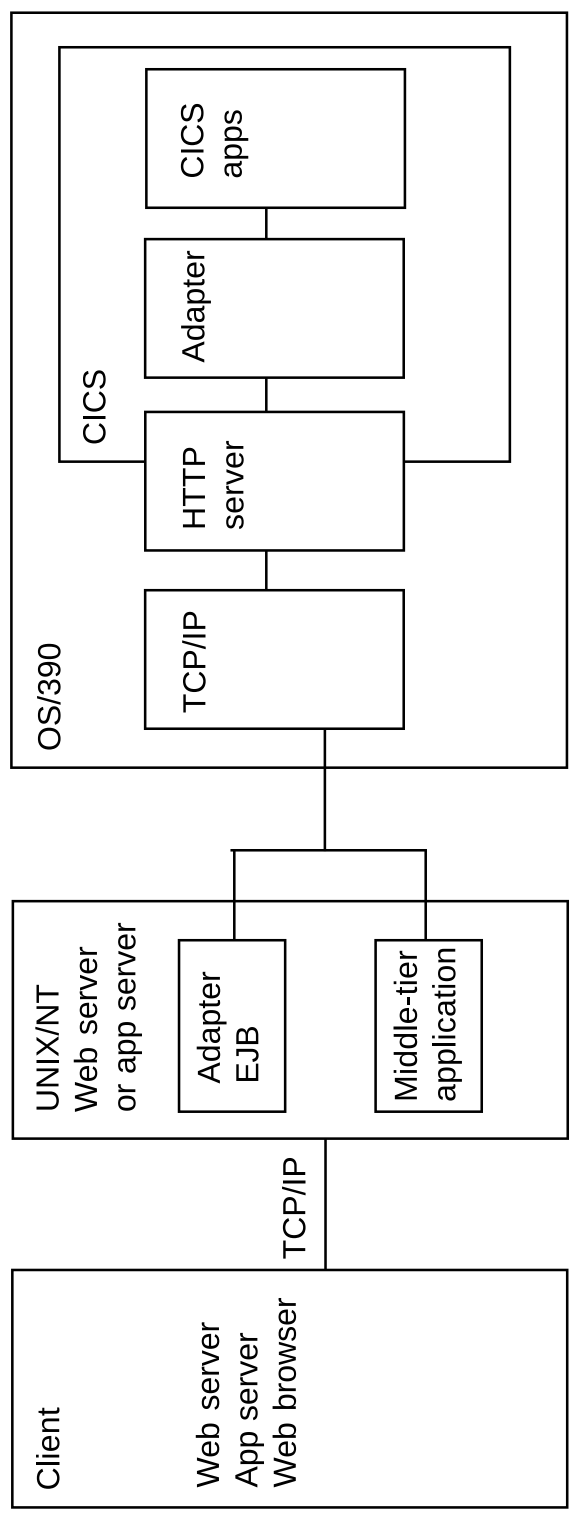

Figure 2: Typical configuration set-up

The above description of security in a 3270 terminal environment is predicated on the existence of a 'principal facility'. Certain

CICS commands, such as EXEC CICS SIGNON, require that a principal facility exists. It is easiest to think of a principal facility as CICS's abstract representation of a physical terminal.

SECURITY IN THE CICS WEB SUPPORT ENVIRONMENT

In the CICS Web Support environment, there is no principal facility. Instead, CWS defines the notion of a 'bridge facility'. Whereas a principal facility represents a physical terminal, a bridge facility represents a virtual terminal. Essentially, a bridge facility is a control block that the existing 3270-based transaction sees as its principal facility. A bridge facility is modelled on an existing installed terminal. By default, CICS uses the CICS-supplied terminal definition CBRF (essentially a basic 3270 LU2 terminal definition). Most existing CICS transactions expect to be invoked by unsolicited input from a 3270 terminal, and then issue RECEIVEs and SENDs to that terminal, its principal facility. This is exactly the emulation that the bridge creates to the existing code. However, even though the task thinks it has a principal facility and can issue RECEIVEs and SENDs, all other tasks, and CICS itself, see the task as a non-terminal task, that is, one running without a principal facility.

SYSTEM CONFIGURATIONS

We built this adapter for NMCI, which had Windows 2000 on the application server; however, we saw this adapter as an opportunity in various military commands running CICS applications. This adapter is being enhanced to connect to middle-tier HTTP servers, OS/390 HTTP servers, and MQSeries.

There are two types of HTTP server found in most networks – basic Web servers and application servers. Web servers from Apache, Microsoft, and Netscape power most of the Unix and NT/2000 servers that comprise the middle-tier between client applications and CICS. The OS/390 HTTP server and CICS Web Support power most of the mainframe Web servers.

Application servers such as WebSphere, SilverStream, and WebMethods, and others, power much of the eBusiness world.

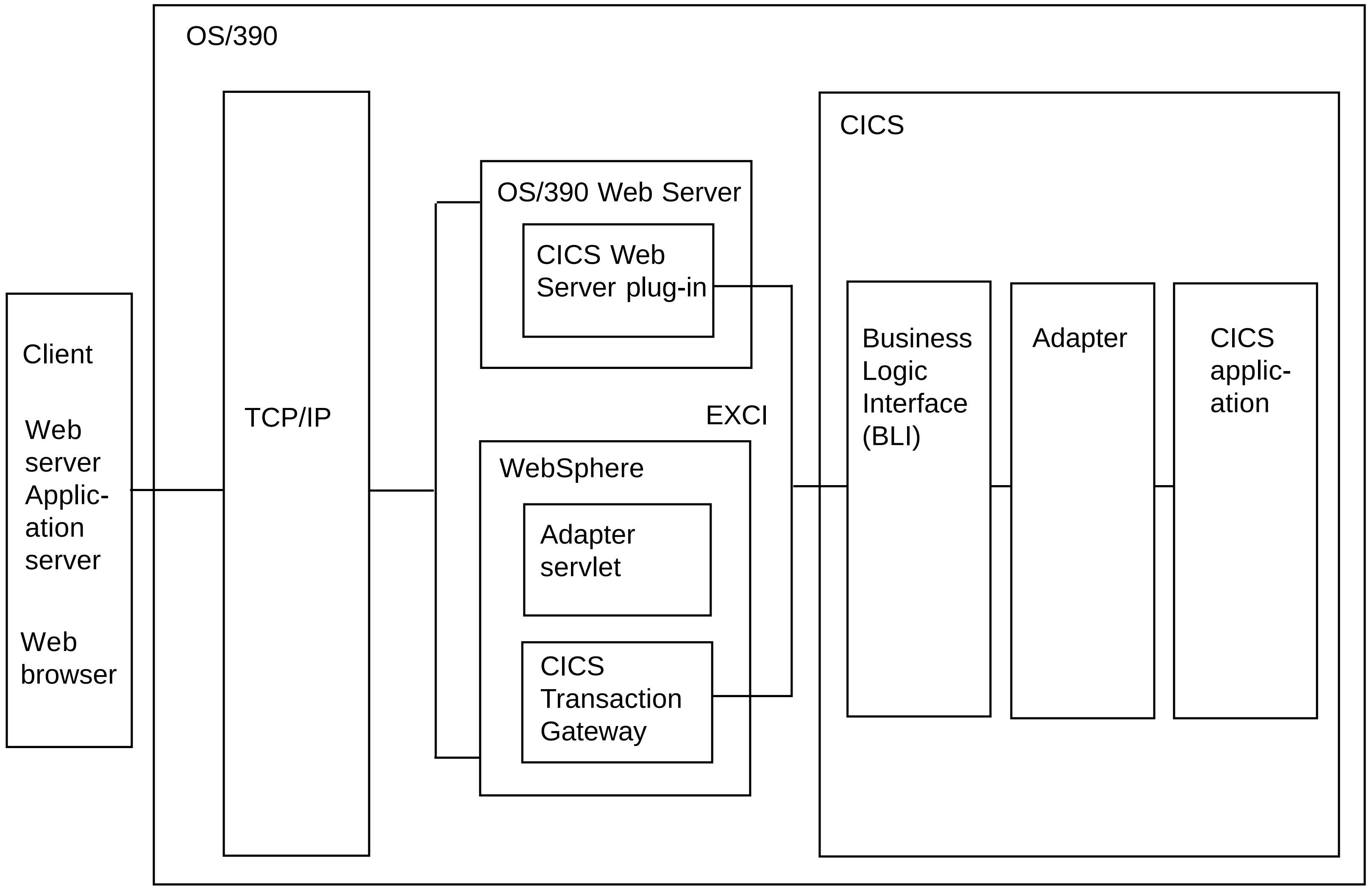

Figure 3: Eliminating the middle tier

These servers collect data from disparate sources and combine them into a single datastream to be viewed in a browser or passed to another application. Below we take a look at each of the adapter system configurations.

MIDDLE-TIER HTTP SERVERS

For middle-tier HTTP server implementation, the adapter has an EJB Component that resides on the Web server. This middle-tier enterprise Java Bean generates and sends a request to the HTTP server running on OS/390. The middle-tier connection to CICS will take place using TCP/IP rather than SNA. Figure 2 shows a typical configuration set-up.

In Figure 2, the following sequence of events results in the retrieval of XML data from CICS.

1 A client application requests data from the middle-tier application by sending a URL that identifies the CICS host and the adapter as the application to execute.

2 The middle-tier application calls an EJB, which sends a request to the HTTP server running on OS/390.

3 The HTTP server passes the request to the adapter.

4 The adapter initiates a CICS transaction and returns the transaction data as XML to the HTTP server.

5 The HTTP server passes the XML back to the middle-tier application, which then passes the data on to the client application.

This is the preferred method for enabling CICS applications to return XML data. It does not require development on the host and there is no need for resource intensive use of an SNA stack on the middle-tier.

OS/390 HTTP SERVERS

By using Web servers or application servers that run on the host, we can eliminate the middle-tier, as shown in Figure 3.

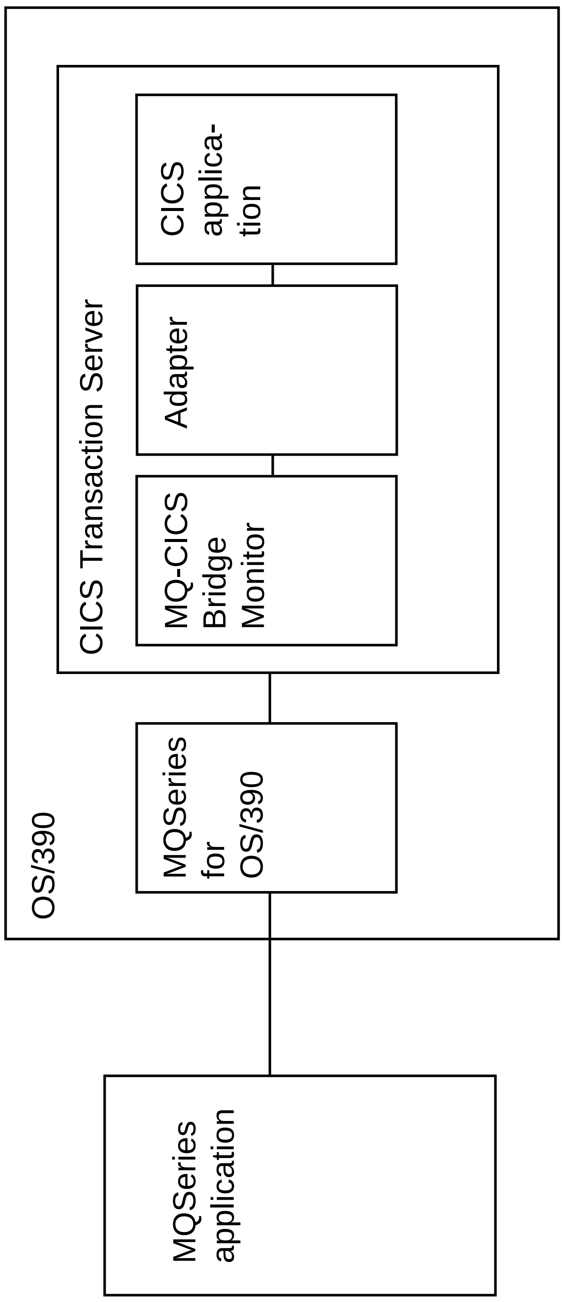

Figure 4: Client communicating using MQ messages

This configuration requires TCP/IP and an HTTP server on the host. The method for retrieving XML from CICS applications differs slightly depending on whether your HTTP server is one of the OS/390 Web Servers or the WebSphere Application Server.

Below are the steps that retrieve XML from CICS using an OS/390 Web Server:

1. A client application connects to the host using TCP/IP and a URL that identifies the CICS host and the adapter as the application to execute.
2. The Web server passes the URL to the CICS Web server plug-in.
3. The plug-in uses the external CICS Interface (EXCI) to connect to the business logic interface of CICS Web Support (CWS).
4. The BLI passes the URL to the adapter.
5. The analyser executes the CICS transaction and returns the transaction data as XML.

Below are the steps to retrieve XML from CICS using WebSphere:

1. A client application connects to the host using TCP/IP and a URL that identifies the CICS host and the adapter as the application to execute.
2. WebSphere passes the URL to the CICS Transaction Gateway (CTG).
3. The CTG uses the external CICS interface (EXCI) to connect to the Business Logic Interface of CICS Web Support (CWS).
4. The BLI passes the URL to the adapter.
5. The adapter executes the CICS transaction and returns the transaction data as XML.

MQSERIES FOR OS/390

If MQ is present on both the client application and the host, then the client application can communicate with the adapter using MQ messages as shown in Figure 4.

This configuration requires the client application to have either

MQSeries server or client installed and MQSeries for OS/390 Version 2.1 or greater running on the CICS host with the MQ-CICS Bridge monitor installed. In Figure 4, the following sequence of events results in the retrieval of XML data from a CICS application.

1 The client application issues an MQPut to the MQSeries Queue manager running on OS/390 with a request to run a CICS transaction.
2 The MQ-CICS Bridge Monitor browses the queue and starts the adapter transaction.
3 The adapter removes the message from the queue and executes the CICS transaction.
4 The adapter builds the XML into a single message and places it in the queue.
5 The client application browses the queue and gets the message containing the XML from the queue.

Sites with existing installations of MQSeries can use this method to XML-enable their CICS applications without having to write their own application-specific bridge code or make any changes to their MQSeries applications.

CONCLUSION

The adapter we built for NMCI provides a flexible, scalable, secure, and easy-to-use solution that makes CICS applications usable by converting application data to XML. Unlike screen-scrapers, this solution does not rely on field locations on a screen; hence, if CICS developers make changes to their applications by adding or removing fields on a screen, Web applications that use this adapter to access the CICS applications will not be affected.

This solution combines mainframe and Internet security technologies into a single solution, so that using it to access CICS applications is actually more secure than using a TN3270 client. This means that we can ensure safety of corporate data while simultaneously making that data available for end-user and eBusiness transactions.

This solution allows other applications to access legacy data and use it in innovative ways to reduce training costs associated with teaching employees how to use complicated host applications, improve business efficiency, and unlock the valuable data businesses spend years accumulating in their mainframes.

Because of the security intensive nature of this customer and the length of the article I have not included the adapter program details that build the XML output. If you wish to get more information on the programming details or specific type of architecture e-mail me at vikas.baruah@ams.com.

Vikas Baruah
Senior Technical Specialist
American Management Systems (USA)

CICS questions and answers

Q We have recently upgraded to CICS TS 2.2 and gone threadsafe with our programs. However, to our surprise the CPU has increased when we look at the CICS and DB2 SMF records. Should this be the case, or have we missed something?

A It sounds as if you are counting the DB2 processor time twice. This is documented in the *CICS DB2 Guide*. When connecting to DB2 V6 or higher and exploiting the open transaction environment (threadsafe in this case) the DB2 CPU time is recorded in both the CICS SMF 110s and the DB2 SMF 101s. So, do not add together the CPU time from the CICS records (SMF 110 records) and the DB2 accounting records (SMF 101 records) when calculating the time for a single transaction, because the DB2 processor time would then be included twice. The total time is simply the time from the CICS records. Also be aware that any DB2 thread creation and termination CPU time (say while using an unprotected thread) will also be included in the CICS SMF 110 record. Previously, with DB2 V5 and lower, this time was unaccounted for.

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.

CICS news

Cole Software has announced z/XDC, its Assembler extended debugging controller. The program can be used for source-level debugging of Assembler programs that execute under z/OS.

Specifically, z/XDC can be used to debug programs running in CICS, TSO, or batch. It can also debug APF-authorized programs, multi-tasking programs, programs running as system exits or product exits (eg SMF exits, DADSM exits, JES2 exits), SVC routines, PC routines, and programs running as subsystems

For further information contact:
Cole Software, 736 Fox Hollow Road, Afton, VA 22920, USA.
Tel: (540) 456 8536.
Fax: (540) 456 6658.
URL: http://www.colesoft.com/about.html.

* * *

MacKinney Systems has announced Version 4.5 of CICS/SWAP. The product allows a single CICS terminal to function as up to eight logical terminals with each one running the same or different transactions.

The new release addresses several issues ranging from dynamic table entry reuse for better 24-7-365 support, 31-bit VSE loadlibs, and new sample programs to automate swap session initiation, and control the initial transaction for sessions based on userid, opid, or other profile identifier.

Enhancements include: added support to reclaim TYPE=AUTO terminals freed in SWPPLNK via the Autoinstall delete exit or ZNEP; password expiration warning message now goes to the console as well as the hardcopy log; included new userid and opid SWAP profile automation to transparently control sessions, keys, and transactions; and VSE release now shipped with 31-bit and 24-bit loadlibs.

For further information contact:
MacKinney Systems, 4411 E State Hwy D, Suite F, Springfield, MO 65809, USA.
Tel: (417) 882 8012.
Fax: (417) 882 7569.
URL: http://www.mackinney.com/products/cics/cics_swap.htm.

* * *

NEON Systems has announced Shadow Event Publisher, which provides a single interface for the real-time capture and publishing of critical mainframe business events occurring within CICS, DB2, and IMS environments. Without touching the application code, events are captured in real time and 'pushed' asynchronously via HTTP and WebSphere MQ messaging protocols to drive heterogeneous business processes.

Event listeners require no changes to the mainframe application code and are tailored to the mainframe subsystem where it resides, optimizing performance and minimizing resource utilization. The product operates in real time, eliminating the lag and expense associated with traditional batch processing and CPU-intensive polling for state changes.

For further information contact:
NEON Systems, 14100 Southwest Freeway, Suite 500, Sugarland, TX 77478, USA.
Tel: 281-491-4200.
Fax: 281-242-3880.
URL: http://www.neonsys.com/Shadow/shadow_event_publisher.asp.