In this issue

3  Considerations regarding the Kill function
10 Redirecting output from Java virtual machines with CICS TS 2.3
21 Tuning CICS TS 2.3 EJB server utilizing enhanced EJB and Java support – part 2
32 Investigating storage violations
46 CICS news

© Xephon Inc 2004
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.

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

Contributions
When Xephon is given copyright, articles published in CICS Update are paid for at the rate of $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.

© Xephon Inc 2004. All rights reserved. None of the text in this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, without the prior permission of the copyright owner. Subscribers are free to copy any code reproduced in this publication for use in their own installations, but may not sell such code or incorporate it in any commercial product. No part of this publication may be used for any form of advertising, sales promotion, or publicity without the written permission of the publisher.

Printed in England.
Considerations regarding the Kill function

The recent introduction of the Kill function in CICS, and the associated CEKL transaction, leads to a number of considerations and related behaviour that warrant further information. This article describes the background to inappropriate use of the Kill function within CICS, the CICS enhancements to messages and dump information regarding Kill, and gives some guidance on ways to approach event investigation within CICS following the use of the Kill function.

INAPPROPRIATE USE OF THE KILL FUNCTION

If a task is purged while in a CICS dispatcher wait, CICS can detect this and take appropriate action. Normally this will be to invoke the relevant clean-up routine(s) and then abend the task. An example is that of a CEDF task wait in DFHEDFD. If this task is purged, the task being stepped through under EDF will be resumed before the CEDF task is itself abended.

With the use of the Kill function, most CICS clean-up routines are bypassed, and so, in this example, the task being stepped through with EDF would remain hanging indefinitely in a wait in DFHEDFX. The use of Kill against a task suspended by EDF is an example of an inappropriate use of the Kill function. It does not allow DFHEDFX to be notified of the Kill operation. As such, this can result in a CICS Dispatcher domain DFHDS0002 message and dump (error code X'0071') for the suspended task.

It should be noted, however, that a task waiting within DFHEDFX for CEDF processing to complete is in a purgable wait, and so Kill is not intended for use against such a waiting task. Normal purge and forcepurge functionality can be utilized in such a case. Remember that purge and forcepurge commands are designed to tidy up the task and system environments as part of their operation. The important point to note is that the use of Kill is intended very much as a last resort, and does not guarantee system or data integrity. Before Kill is used, it is important to
understand the type of wait that the target task is in. This can be achieved using tools such as CEMT, the EXEC CICS SPI, CICSPlex SM, or CEKL. Also, purge and/or forcepurge should be attempted before a Kill function is made against any task in CICS anyway.

Certain situations within CICS inhibit purge or forcepurge operations to preserve integrity. An example is while a CICS task is waiting for a VSAM I/O operation to complete. If a Kill request were used against such a waiting task, the results would be unpredictable when the I/O did eventually complete. It is for this reason that all transaction dumps resulting from Kill operations should be retained, to allow investigation of the state of the task when it was killed and correlation with any subsequent CICS system problems.

RELATED CICS ENHANCEMENTS

A number of new CICS messages are associated with the Kill function. From an automated operator perspective, these may be of interest. The message numbers for the new CQ domain within CICS are as follows:

DFHCQ$001, DFHCQ$002, DFHCQ$010, DFHCQ$011, DFHCQ$012, DFHCQ$013, DFHCQ$014, DFHCQ$015, DFHCQ$020, DFHCQ$021, DFHCQ$022, DFHCQ$023, DFHCQ$024, DFHCQ$025, DFHCQ$026, DFHCQ$027, DFHCQ$028, DFHCQ$029, DFHCQ$030, DFHCQ$031, DFHCQ$032, DFHCQ$033, DFHCQ$034, DFHCQ$035, DFHCQ$036, DFHCQ$037, DFHCQ$038, DFHCQ$039, DFHCQ$040, DFHCQ$041, DFHCQ$042, DFHCQ$043, DFHCQ$044, DFHCQ$045, DFHCQ$046, DFHCQ$047, DFHCQ$048, DFHCQ$049, DFHCQ$050, DFHCQ$051, DFHCQ$052, DFHCQ$053, DFHCQ$054, DFHCQ$055, DFHCQ$056, DFHCQ$057, DFHCQ$058, DFHCQ$059, DFHCQ$060, DFHCQ$061, DFHCQ$062, DFHCQ$063, DFHCQ$064, DFHCQ$065, and DFHCQ$066.

Also, DFHDS0010, DFHDS0011, and DFHME0139 messages reflect CICS dispatcher and message domain responses to Kill events.

Of these messages, there are a number that are worth considering in more detail:

- DFHCQ$0100 – console queue initialization has started.
- DFHCQ$0101 – console queue initialization has ended.
- DFHCQ$0200 – CEKL transaction enabled.
The above three messages are informational, and relate to the CEKL console environment being initialized. They tell you that the use of the CEKL facility is available for this CICS system.

The following message is issued when a CEKL inquire command results in one or more tasks that are found matching the options specified on the command:

DFHCQ0243 CEKL INQUIRE: matched selection criteria.

CICS displays information about these tasks on the console and job log.

The following three messages relate to the use of CEKL to attempt to cancel tasks from CICS:

DFHCQ0252 CEKL SET: PURGE request issued for task number 'task num'.
DFHCQ0256 CEKL SET: FORCEPURGE request issued for task number 'task no'.
DFHCQ0259 CEKL SET: KILL request issued for task number 'task num'.

The following message is issued to confirm the acceptance of a (non-CEKL issued) Kill request against a task:

DFHDS0010 Kill request accepted for transaction id 'transid', transaction number 'task num', userid 'userid'.

The following message is issued to confirm the reacceptance of a (non-CEKL issued) Kill request against a task. In other words, a Kill attempt has been issued against a task for which a Kill request has already been issued:

DFHDS0011 Kill request reaccepted for transaction id 'transid', transaction number 'tran num', userid 'userid'.

In addition, there are a number of new CICS abend codes introduced to support the Kill function. These are of the form AKKx. The type of abend code gives details about the nature of the Kill request, and so is worth understanding in more detail.

Abend codes AKKA, AKKB, and AKKC signify (CICS SPI/CEMT/ CICSPlex SM) Kill request details for tasks in various states of purge and forcepurge protection within CICS:

• AKKA

Explanation: a Kill request has been actioned when a task was not protected from purge or force purge.
• AKKB
   Explanation: a Kill request has been actioned when a task was not protected from forcepurge but was protected from purge.

• AKKC
   Explanation: a Kill request has been actioned when a task was protected from forcepurge.

Abend codes AKKD and AKKE reflect that CEKL options have been used:

• AKKD
   Explanation: a CEKL purge has been requested.

• AKKE
   Explanation: a CEKL forcepurge has been requested.

Finally, abend codes AKKG and AKKH describe Kill requests involving task runaway:

• AKKG
   Explanation: the CICS kernel (KE) domain has detected a Kill request from the runaway exit program. The task was not protected from runaway when the Kill request was actioned.

• AKKH
   Explanation: the CICS Kernel (KE) domain has detected a Kill request from the runaway exit program. The task was protected from runaway when the request was actioned.

Understanding these abend codes can be of use when investigating any unforeseen events within the CICS system that may occur as a result of the use of the Kill function.

EVENT INVESTIGATION AFTER THE USE OF KILL
The use of Kill may result in subsequent problems within CICS,
because of the unpredictable effects that the Kill operation can have on the system and data integrity. To that end, it may be necessary to investigate subsequent problems that arise after Kill has been used.

In addition to analysis of the transaction dumps from the tasks abended by the use of Kill, there are other useful sources of information that can provide additional data.

The CEKL transaction records details of tasks that have been abended by a CEKL Kill request.

This information is displayed within the CQ VERBX when formatting a CICS system dump. It can be used in tandem with the various associated system console messages and CICS messages.

In the example data below, CEKL was used to issue a Kill request against a transaction TEST.

The command to do this was:

```
F CICSJOBN,CEKL SET TASK(00028) KILL
```

The following information appeared on the console as a result:

```
+CICSJOBN DFHCQ0259I
  CEKL SET: KILL request issued for task number 00028.
+CICSJOBN TAS(00028) KIL SUS QR TEST USE(CICSUSER) ATT(00029) CPU(*****)
  RUA(000020)
+CICSJOBN HTI(00030) HTY(ICWAIT) HVA(V111)

CICS issued the following messages:

DFHAC2206 15:35:33 CICSJOBN Transaction TEST failed with abend AKKA.

  Updates to local recoverable resources backed out.

DFHDU0203I 02/27/2004 15:35:33 CICSJOBN A transaction dump was taken for
dumpcode: AKKA, Dumpid: 1/0001.

DFHAC2236 02/27/2004 15:35:33 CICSJOBN Transaction TEST abend AKKA in
program DFHECID term V112. Updates to local recoverable resources will be backed out.

An examination of the data in the CQ VERBX from a CICS system dump revealed the following:

```
==CQ: CQ TCB TRACE ENTRIES
XM_TXN was located at: 14609030
Transaction number: 00028
CEKL purge status: KIL
TCB: QR
Dispatcher status: SUS
Transaction id: TEST
Transaction class:
User id: CICSUSER
Attach time (seconds): 00029
CPU time (seconds): *****
Run away interval (seconds): 00020
Suspend time (seconds): 00030
Suspend reason: ICWAIT
Suspend value: V111
```

Using the combination of data items shown above, it is possible to establish the environment that was in place at the time a task was abended by use of the Kill function. This information can be of use when investigating any subsequent problems that may occur after this point in time. Once again, it is important to emphasize that the use of the Kill function is intended very much as a last resort. Such problem determination should be regarded as an important technique, but one that would rarely be expected to need to be employed.
POINTS TO NOTE REGARDING THE KILL FUNCTION

When the Kill function is used to terminate a task, the subsequent abend cannot be handled by an application (via an EXEC CICS HANDLE command), nor via other recovery routines that other products may have in place.

In addition, the abend of the task will result in a dynamic transaction backout (DTB), where any changes to recoverable resources that were made within the task’s Unit Of Work (UOW) are backed out. This is also the case when a task is purged or forcepurged. As such, the task being terminated may not leave the CICS system immediately; there will be a period of time required for it to back out its modifications as part of abnormal termination. It is possible to determine the state of a task by means of the EXEC CICS INQUIRE UOW command. This returns information on the UOWs within the CICS system, such as their age (in seconds), associated transaction identifier and task number, and also the state of the UOW – whether it is inflight, in backout, indoubt, etc. If a UOW is shown to be in backout state, then this shows that CICS is backing out the recoverable changes made during its lifetime. This approach can reveal why a task that was Killed has yet to leave the system (it also applies to those tasks that were purged or forcepurged, for that matter).

As described above, the use of Kill can lead to unpredictable results. One of these is that the termination of tasks that are currently within backout processing can result in CICS being terminated as a result. Such tasks are protected from being purged or forcepurged, but Kill requests are allowed to be performed against them. As such, an initial examination of their state (by means of the CEMT INQ UOW or EXEC CICS INQ UOW SPI options) is worthwhile. This approach requires that the QR TCB be available to process such requests against the UOW.

It is not always possible for CICS to honour purge, forcepurge, or Kill requests. For example, if CICS is Short On Storage (SOS), the act of abending a task can result in the request not being able to complete for lack of available CICS storage anyway.
Redirecting output from Java virtual machines with CICS TS 2.3

Java applications running under the control of CICS Transaction Server can produce both output and error messages. These may need to be handled in specific ways for different applications. CICS TS Version 2 Release 3 provides a mechanism to intercept, modify, and redirect these messages.

This article explains how the output redirection mechanism can be customized and configured. It describes in detail the sample code provided by CICS, which illustrates how this mechanism can be tailored. This article is primarily intended for Java programmers who are designing new applications that will run within CICS, and who wish to influence the way messages are handled. It may also be of interest to systems programmers who are responsible for setting up the environment that supports Java workloads within CICS.

INPUT AND OUTPUT OPERATIONS IN JAVA

Java applications may perform both input and output operations when they execute. Output is directed to one destination for standard output and a second one for standard error output. Similarly an application can read input from a device. These
locations are referred to as STDOUT, STDERR, and STDIN, respectively, throughout the CICS documentation.

The way this is supported though the Java programming model is a consequence of the fact that all Java programs automatically import the java.lang package. This package defines, amongst other things, a class called System, which controls run-time functions including those used for input and output by a program.

In Java, the input a program receives and the output it produces are considered as streams of data. In the System class there are three predefined variables representing an input stream and two output streams (one for normal output and the other for error output). These are identified as in, out, and err respectively.

These fields are declared as public and static within the System class, permitting them to be used within a Java program without the need to refer to a specific instance of a System object. In this way the program does not have to be concerned about how it receives its input, or how it delivers output or error messages.

OUTPUT FROM JVMS RUNNING UNDER THE CONTROL OF CICS

CICS TS has provided support for Java programs for a number of releases. It makes use of information in a file known as the JVM profile to control the launching operation for the JVMs that CICS manages. A second file, the JVM properties file, defines the operational characteristics of a JVM.

CICS takes responsibility for defining the location of the input, output, and error streams when it starts up a JVM. It does so by looking for a number of different options in the JVM profile that it reads prior to launching the new JVM. The WORK_DIR option defines the directory in the local Unix Hierarchical File System (HFS), where three files are defined or reused as the input, output, and error streams. The STDIN, STDOUT, and STDERR options indicate the name of the files to be used. If any or all of these options are missing, then CICS assumes default values for those components that are not defined in the JVM Profile.
If a number of JVMs use the same profile, they will share a common set of files. This may result in messages from different JVMs becoming interleaved. To avoid this situation, there is a -generate parameter that can be included with the STDOUT and STDERR option in a JVM Profile. This causes a unique name to be generated for that file when a JVM starts, identified by a time stamp and the applid of the CICS region. In this way, output and error messages can be directed to separate files for each JVM. However when a JVM is reused by several program invocations, all messages from these programs are then written to the same output and error files.

Prior to CICS TS 2.3 no provision was made for the further separation of messages produced from within a single JVM by a series of transactions running there. There was no way to augment messages with additional information when they were being delivered, other than to do this by the Java application that issued them. This makes it difficult to distinguish those messages that originated from a single application instance running in a reusable JVM. It was also only possible to send these messages to a destination in the Unix file system.

OUTPUT REDIRECTION WITH CICS TS 2.3

Release 2.3 of CICS permits a single Java class to be named in a JVM profile, which is to be used to redirect output from applications running in any JVM using that profile. The class is defined using the USEROUTPUTCLASS option. There is no default class for this option, and, if it is not included in a JVM Profile, then those JVMs referencing that profile when they are launched have the output redirection mechanism turned off.

THE OUTPUT REDIRECTION MECHANISM

CICS starts up JVMs only in response to requests to run Java programs. When a new JVM is created, CICS identifies the name of a JVM Profile from the resource definition for the program it has been asked to run. The JVM Profile is read, and the information in it is used to launch the new JVM.
Once the JVM has been started, a special class, which CICS provides, known as the Wrapper, is invoked within the JVM, and is used to perform some state initialization functions. However, the main purpose of the Wrapper is to carry out pre- and post-processing around the execution of each Java application that is run within a JVM.

Once a JVM has fully initialized, CICS calls the main method of the Wrapper with an argument list that CICS has set up. This argument list consists of two parts. The first is the name of the initial Java class defined by the JVMCLASS property of the Java program’s CICS resource definition. The second is optional and, when included, contains the name of the Java class that is to be used for output redirection, which was found from the USEROUTPUTCLASS option when the JVM’s Profile was originally read.

The Wrapper is used to carry out operations before and after the execution of the real Java application. Prior to running the application, the Wrapper looks to see whether there is an output redirection class in its argument list, and, if found, an instance of the class object is created for output and another for errors for use by the current application. If no class is found, output and error messages will be routed to the destinations identified from the STDOUT and STDERR options defined in the JVM’s Profile, or from their default settings if these options are missing.

When a Java application finishes executing, control passes back to the Wrapper, which then discards the instances of the objects used to redirect standard output and standard error messages.

The output redirection class must be placed on the trusted middleware classpath for the Wrapper to be able to locate it. This is done by adding the class name to the TMPREFIX or TMSUFFIX option in the JVM Profile. Once successfully loaded, control passes to the writeRecord() method of this class whenever the JVM attempts to use System.out or System.err. This method can be tailored to suit the needs of a particular workload.

CICS provides a Java interface, called
OutputRedirectionPlugin.java, which is used by the Wrapper to create instances of the output redirection object. Any user-written class must implement this interface if it is to be used for the purpose described here.

THE OUTPUTREDIRECTIONPLUGIN INTERFACE

This interface is provided in the com.ibm.cics.server subdirectory of the CICS TS route directory in the HFS. This is also where the Wrapper class is located.

The interface has the following signature:

```java
import java.io.*;
public interface OutputRedirectionPlugin {
    public void initRedirect(String inDest, PrintStream inPS, PrintStream inStderr, String inApplid, String inProgramName, Integer inTaskNumber, String inTransid);
}
```

When the Wrapper uses this interface, it substitutes either the ‘stdout’ or the ‘stderr’ character string for the inDest. This is the label that the intended destination is known by.

The inPS parameter is the PrintStream, which represents the original destination for this particular category of messages – STDOUT or STDERR. The inStderr parameter is the PrintStream of the original STDERR for the JVM. It is included to permit any class implementing the interface to know where messages would be routed if it were not used, and also to provide somewhere to write error messages should the redirection mechanism encounter a problem when it is being used.

The remaining fields in the interface allow the Wrapper to hand on values that may be useful to the output redirection class. These might then be used to extend a message or to create a new one.
SAMPLES PROVIDED BY CICS

Two samples are provided with CICS TS 2.3 to illustrate different ways to use the output redirection mechanism. Both of these inherit from a common abstract superclass, com.ibm.cics.samples.SJStream. All of these can be found in the HFS directory /usr/lpp/cicsts/cicsts23/lib, where cicsts23 is a user-defined value that is selected for the CICS_DIRECTORY variable used by the DFHIJVMJ job during CICS installation.

THE SJSTREAM CLASS

This superclass extends the outputStream class and implements the OutputRedirectionPlugin interface. It defines default methods called close(), flush(), write(byte[]), write(byte[],int,int), write(int), openHFS(), and an abstract method called writeRecord(). It also contains an initRedirect() method, which is where the interface implementation is found. This method does nothing more than return the Boolean value of false. The Wrapper uses this Boolean value to indicate whether or not it has successfully called the interface.

This failure reported by the superclass forces any class that inherits from it to include an initRedirect() method, which can be used to create and initialize some objects, and which returns a Boolean value of true, indicating that it has been successful. Such a class must also include a writeRecord() method, which controls the redirection of the messages that are written through it.

THE SJMERGEDSTREAM CLASS

The com.ibm.cics.samples.SJMergeStream class demonstrates how to send output and error messages to different Transient Data queues. The queues that are named in the sample are CSJE and CSJO. Resource definitions are provided by CICS for both of these queues and need to be installed prior to using this particular class.

The initRedirect method initializes the state of the object instance
of this class. Because there are separate object instances for standard output and standard error message handling, the instance responsible for output messages makes use of the CSJO TD queue, and the one for error messages uses CSJE.

During the initialization of these object instances, a fully-qualified file name is generated, which indicates where in the local HFS messages are to be written should a failure occur while using the TD queue.

The writeRecord() method implements the code, which redirects the data being sent from the application executing in the JVM. It extends each message that is being redirected by adding information to the beginning. It uses the Date class, from the java.util package, to obtain a time stamp for each message. This is concatenated together with the applid, transid, task number, and program name, which were all determined by the Wrapper and fed in using the interface. This data is then added to the beginning of the current message, which is being processed, before it is written out to the redirected destination.

Next, this method attempts to write the message to a TD queue. There is the restriction that CICS resources are accessible from only the initial process thread of a Java application. Attempts to do so from other threads would result in unnecessary errors, which are avoided in this sample by carrying out a test to see which thread is currently being used. If the initial process thread is not the current one, the message is written to the file whose name was previously generated by the initRedirect() method.

Much of the code in the writeRecord() method is contained within a single try/catch block and, as a result, all exceptions that are encountered are handled in the same way. This is one area in the samples that could be extended to allow specific conditions – such as a failure occurring because the data is too large for the TD queue’s record size – to be dealt with differently from, for instance, the one raised if the queue is not found or is disabled.

When an error condition is encountered within the writeRecord() method, some care has to be taken if there is a requirement to
write out a message reporting this event, because this message may get dispatched into the failing method and an infinite loop might be entered. The sample deals with this problem with recursion. It looks to see whether this object represents redirection for STDERR or for STDOUT. If it is STDERR then any messages are sent directly to the original STDERR for the JVM, avoiding further attempts to redirect them; if it is, it is safe to attempt to send an error message to the redirected STDERR.

Once the error has been reported, the writeHFS() method is called to send the message to the file system. This method supersedes the one defined in the SJStream class. It deals with failures to write to the file in a similar manner to the writeRecord() method. In doing so, messages are always written somewhere. If they cannot go to the TD queue, an attempt is made to send them to a file, and, if this fails, they end up in the original STDOUT or STDERR destination.

THE SJTASKSTREAM CLASS

The com.ibm.cics.samples.SJTaskStream class has a similar initRedirect method to set up and maintain state information used by subsequent redirection operations. However, the main difference in this class is that each task that executes in a single JVM has its own unique file for output and a second one for error messages.

The Date class is again used to derive a timestamp. This is concatenated with the string ‘.task.’, which is in turn concatenated with the task number to produce a unique file name. In addition, a directory name is derived for the file. This is made up from looking up the system property user.dir and then adding to it the CICS region applid followed by either the ‘stdout’ or the ‘stderr’ string, depending on which type of message this output redirection object is intended to process. If this system property is not set, /temp is used as the root directory, which is then extended with the applid and either stdout or stderr. An example of this directory might be /usr/lpp/cics/cicsts23/IYKABC1A/stdout/.
The initRedirect() method then uses the File(String) constructor, provided by the java.io package, to create an instance of this directory and then the File(File, String) constructor to create an instance of the file itself. If either of these fails, the method returns ‘false’ to indicate to the Wrapper that output redirection is not possible for the JVM. The main cause of errors during these operations is that the security permission settings in some part of the HFS prevent the new file or its directory from being created.

Another difference in this version of the initRedirect() method is that it instantiates an object called ‘baos’, which is created using the ByteArrayOutputStream class. This class is an implementation of an output stream that uses a byte array as the destination. The constructor that is used for this has no arguments and results in a 32-byte output buffer being created for use by this object. A second constructor is available and could be used if a different-sized buffer is needed.

When a message is written by a program running in the JVM that is using this sample for output redirection, the writeRecord() method is invoked. This method attempts to open the file if it has not been used thus far and if no file open errors have been reported by any previous calls to this method. It does this by checking the content of the buffer outBuffer, which belongs to the baos object and which is set to null before it is used. If a failure occurs while attempting to open the file, a flag is set to prevent further attempts from taking place should the method be called again.

A new String is created from the concatenation of a timestamp, the applid, transid, task number, and program name, followed by the content of the record that is being written by the Java application running in the JVM. The record content is retrieved by calling the toString() method on the baos object. Once built, this new extended message, even if it cannot be redirected to the preferred destination, is the one that will appear in the original STDOUT or STDERR file.

If there was no error when opening the file, an attempt is made to write a new record to it. Three methods are called on the
outBuffer object. The first is used to write the record into the buffer. The second adds a new line character to the end of the record, and the third method is used to flush the buffer. The reset() method is then called on the baos object to clear its buffer for use by any subsequent request that may be issued from the Java application running in the JVM.

Any errors that occur during these operations are handled in a single catch block. This block determines whether or not the current record is being redirected for stderr, and, if it is, a message reporting the error is written to the original STDERR, followed by the current record. This prevents an infinite loop from being entered while attempting to write an error message to a redirected destination that is failing, and ensures that the original message is still written somewhere.

If, instead, this is a record that is being redirected for stdout, then an attempt is made to send a message to stderr reporting the problem, which message itself may then be redirected. Following this, the original message is written to the original STDOUT destination for the JVM.

The last part of the writeRecord() method deals with the situation where the file could not be opened. If there is some data in the buffer belonging to the baos object, and if the destination is STDERR, there is nothing to do at this point because this is the bottom of a recursive sequence resulting from a failure with the redirected error destination. As the recursion rolls back, the message will get written to the original STDERR for the JVM. The only remaining option is to deal with data that is being written to an output stream. The redirected version of this stream has failed and here an error message is written to the redirected STDERR before the current record is written to the original STDOUT.

MODIFYING THE SAMPLES OR WRITING ALTERNATIVE CLASSES

The samples can be used unchanged so long as any CICS resources that they depend on are installed and enabled on the region that controls JVMs that make use of them. However, they are unlikely to suit the needs of a broad variety of Java workloads.
They could form the basis of a template that customers might then tailor or extend to suit their own requirements. The general structure of these samples provides the logic needed to allow errors to be dealt with, avoiding recursive problems when error messages are issued from within the methods. The error handling is very basic in both the samples, and this is one area that might be customized to separate out the actions taken when some specific condition is encountered.

If, instead, new classes are written for this purpose, they could inherit from the SJStream class or they could simply implement the OutputRedirectionPlugin Java interface. Whichever approach is adopted, care should be taken when using CICS resources within an output redirection class to avoid inadvertently trying to use them when dealing with output from a multithreaded Java application. CICS resources can be used by such a class only when accessed from the initial process thread.

Finally it is worth pointing out that the generation of output and error messages is comparatively expensive and that this overhead becomes worse when output redirection is used. As a result, this mechanism should be used only when there is a real need and should be avoided in a production system wherever possible.

FINDING FURTHER INFORMATION ON THIS SUBJECT

The CICS Infocentre provides a wealth of information on the use of Java in CICS and describes output redirection in some detail. A good Java 2 programming reference is another place to look for information on the classes and interfaces that are provided by the java.io package. The CICS samples use only a small subset of the classes and methods that are provided by the Java language. As a result, there is considerable opportunity to enhance output redirection beyond what is demonstrated in these samples.

Mike Brooks
Senior IT Specialist
CICS Development, IBM Hursley Park (UK)
This month we conclude the article looking at tuning CICS TS 2.3 EJB server, utilizing enhanced EJB and Java support.

LE enclave allocation:

- A system heap contains system class objects that persist for the lifetime of the JVM. JVMs never perform garbage collection on this heap. Worker JVMs do not have a system heap because they use the master JVM's system heap.

- An application-class system heap contains class objects representing sharable application classes, which persist for the lifetime of the JVM and are reinitialized if the JVM is reset. JVMs never perform garbage collection on this heap. Worker JVMs do not have an application-class system heap because they use the master JVM's system heap. Continuous and single-use JVMs (those with REUSE=YES, REUSE=NO or Xresettable=NO in their JVM profile) do not have an application-class system heap because these types of JVM are not reset after use.

- A non-system heap comprises two other heaps of variable size:
  - A middleware heap, which contains objects whose lifetime is greater than a single transaction. JVMs perform garbage collection on this heap. All types of JVM have a middleware heap.
  - A transient heap, which contains objects whose lifetime is the same as the transaction using it. The segregation of this heap from the middleware heap improves the performance of garbage collection. JVMs can perform garbage collection on this heap if it runs out of space in the non-system heap. It is also completely deleted if the JVM is reset.
If you are using continuous JVMs, you need to be aware that the storage heaps in these JVMs are not automatically cleaned up after each program invocation. As a result of this, the continuous JVM might require either larger storage heap sizes or more frequent garbage collection. A resettable stand-alone JVM running the same workload can potentially have lower storage requirements, depending on the application design and the extent to which each program cleans up after itself. You also need to remember to set aside storage in each region for the shared class cache itself.

You need to pay special attention to how JVM standard class paths are defined. The standard class path is the only class path that is taken from the profile for the worker JVM itself, rather than from the profile for the master JVM. Classes on this class path are loaded into the individual worker JVMs and are not cached in the shared class cache. Do not add classes to this class path because it is detrimental to performance for a resettable worker JVM, since the classes are reloaded every time the JVM is reset. For a continuous worker JVM, these classes are kept intact from one JVM reuse to the next, so there is no need to reload them but having the classes in every JVM uses more storage than having a single copy in the master JVM. To reduce heap storage requirements, you should avoid using the standard class path for worker JVMs in a production environment.

For better performance, choose the shareable application class path for application classes that you want to be cached. For stand-alone JVMs, the classes will be cached in the JVM, kept across JVM reuses, and reinitialized if the JVM is reset. For worker JVMs, they will be obtained from the shared class cache. Adding application classes to a shareable application class path, rather than to a standard class path, produces the best performance in a resettable JVM, and it should be your normal choice for loading application classes in a production environment.

As for xmx settings, my suggestion is to set the initial value high and tune it down, based on the statistics you collected. To increase the accuracy of the results, you should purge any JVMs
with the profile that you are tuning, around the time of a statistics reset. This ensures that the statistics collected in the next statistics interval are a more accurate reflection of the storage usage for those JVMs.

Suggested initial HEAP size setting can be as follows and should be tuned based on garbage collection (GC) output:

- `Xmx=60M` (set higher than you could possibly use)
- `Xinitth=500K` (not applicable for a continuous JVM)
- `Xms=30M`
- `Xmaxf=1` (turns off heap shrinkage)
- `Xmaxe=1M` (with `Xmine` setting, sets heap expansions to 500K)
- `Xmine=1M`
- Do not specify `Xinitsh` (will default to 128KB)
- Do not specify `Xinitacsh` (will default to 128KB).

**GARBAGE COLLECTION**

For CICS TS 2.3, a full garbage collection is automatically requested by CICS for each JVM after every 101 transactions. If a JVM runs out of space in one of its storage heaps in between these automatic garbage collections, and is unable to allocate any more objects (an allocation failure), the Garbage Collector (GC) tries to expand the storage heap. If there is not enough free storage available to do this, because the middleware heap and the transient heap have already been expanded to fill the amount of storage specified by the `xmx` option in the JVM profile, a full garbage collection is triggered to free some storage. It is generally better to tune the heap settings to avoid this situation so that, if possible, the automatic garbage collections are the only full garbage collections performed.

From the output produced by GC, you can see whether the JVM profile for the JVM contains the right settings for the initial and
maximum sizes of the storage heaps and check that you have specified appropriate storage for the needs of the programs that run in the JVM. When you have optimized your use of virtual storage for each JVM, you might be able to increase the number of JVMs in your CICS region, if needed.

GC collection is set by specifying VERBOSE=gc in the JVM profile that you want to tune. By default, the messages are output to the file that is specified by the STDERR option in the JVM profile (the default name is dfhjvmerr) in the HFS directory that is specified by the WORK_DIR option in the JVM profile.

These ‘action’ fields in the GC report are in order of severity and have the following values:

- 1 – a preemptive garbage collection cycle
- 2 – full allocation failure
- 3 – a heap expansion takes place
- 4 – all known soft references are cleared
- 5 – stealing from the transient heap is done.

The example below shows output from GC, where the xmx setting is too low. Output from this garbage collection report shows an allocation failure, where there is insufficient space in one of the storage heaps for the JVM to allocate an object, causing heap expansion. In this case the middleware heap and the transient heap have already been expanded to fill the amount of storage specified by the xmx option in the JVM profile. This output shows that the value for the xmx option in the JVM profile is insufficient for the requirements of the Java programs, and it can indicate that the JVM has not been correctly tuned.

```
<AF[7]: Allocation Failure. need 524 bytes, 1894 ms since last AF>
<AF[7]: managing allocation failure, action=3 (1962672/3471872)>
<GC(53): GC cycle started Fri Mar 5 09:16:12 2004
<GC(53): freed 114688 bytes from Transient Heap 96% free
(507376/523776) and>
<GC(53): freed 247144 bytes, 62% free (2078744/3340800), in 6 ms>
```
To ensure that you do not encounter too frequent garbage collection (which might indicate that the xmx value is too low), please review JVM monitoring data, the JVMRTIME field, which shows the elapsed time spent resetting the JVM environment to its initial state, including any garbage collection that takes place in the JVM.

CHOOSING APPROPRIATE MAXJVMTCBS LIMIT

The MAXJVMTCBS system initialization parameter limits the total number of TCBs in the pool of J8- and J9-mode open TCBs that CICS uses for JVMs. CICS decides how many of them should be J8 TCBs and how many should be J9 TCBs, according to the number of requests that specify each execution key. The JM TCB, for the master JVM, which initializes the shared class cache, does not count towards the MAXJVMTCBS limit.

The default setting for MAXJVMTCBS is 5, where the minimum permitted value is 1 (meaning that CICS is always able to create at least one open TCB). You have to be careful not to over specify this value. Each JVM runs in its own LE, and uses MVS storage. For this reason, you need to choose a MAXJVMTCBS limit for your CICS region based on the amount of MVS storage available for the use of the region and the amount of MVS storage used by each of your JVMs. If you set a MAXJVMTCBS limit that is too high, CICS might attempt to create too many JVMs for the available MVS storage, resulting in an MVS storage constraint. You may want to do volume testing in your development environment, issuing frequent statistics and setting the MAXJVMTCB value relatively low (under 8). Reviewing the CICS shutdown statistics will help you determine how much of MVS storage is used by each JVM, and will enable you to calculate what the MAXJVMTCB limit should be for optimum performance, while leaving enough of a cushion in unallocated MVS storage for the CICS address space.
You can change the setting for MAXJVMTCBS without restarting CICS by using the CEMT SET DISPATCHER MAXJVMTCBS command.

To estimate how many JVMs you need to support a desired level of transaction throughput, use the following formula:

\[ \text{ETR} \times \text{Response time} = \text{Number of JVMs} \]

where:
- ETR is the desired level of transaction throughput.
- Response time is the time taken to run your transaction in a JVM.

If you have determined that your CICS region could benefit from an increase in the number of JVMs, you now need to calculate the maximum number of JVMs that your CICS region could support. To calculate the maximum number of JVMs your CICS region can allocate, perform the following calculation:

Free storage available in the CICS address space/storage needed per JVM

**JVM STORAGE MANAGEMENT**

JVMs use storage that is managed by MVS, and so they cannot make use of the storage management facilities available from the existing CICS storage manager. This was an issue under CICS TS 2.2 when testing with a high number of EJB JVMs. When MVS storage above the 16MB line was exhausted, I have seen it spilling into MVS storage below the line and causing a situation CICS could not recover from. An example of CICS shutdown statistics showing this condition is below.

**CICS TS 2.2 statistics:**

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Private Area Region size below 16Mb</td>
<td>10,216K</td>
</tr>
<tr>
<td>Max LSQA/SWA storage allocated below 16Mb (SYS)</td>
<td>556K</td>
</tr>
<tr>
<td>Max User storage allocated below 16Mb (VIRT)</td>
<td>9,668K</td>
</tr>
<tr>
<td>System Use</td>
<td>20K</td>
</tr>
<tr>
<td>RTM</td>
<td>250K</td>
</tr>
<tr>
<td>Private Area storage available below 16Mb</td>
<td>-278K</td>
</tr>
</tbody>
</table>
With CICS TS 2.3, IBM made great enhancements to this area. Now there is a new storage monitor provided for managing MVS storage. As JVMs make requests for MVS storage, the storage monitor checks whether the availability of MVS storage has dropped below pre-set thresholds and notifies CICS when this is the case, thus avoiding an outage. Once CICS has been notified that MVS storage is constrained, the actions it takes, depending on the seriousness of the situation, are as follows:

- When MVS storage is constrained, CICS deletes JVMs in the JVM pool that are not currently in use, together with their TCBs. However, new JVMs can still be created for incoming requests.

- When MVS storage is severely constrained, CICS temporarily prevents the creation of new JVMs for incoming requests, and behaves as though the MAXJVMTCBS limit has been reached and the JVM pool is full. CICS then terminates all JVMs as soon as they finish running their current Java programs. If limited MVS storage is available, and the storage monitor is still receiving requests from CICS to create JVMs, it queues any such requests that cannot obtain sufficient MVS storage.

- When CICS manages to reduce its use of MVS storage by these methods, and the availability of MVS storage has risen above the pre-set thresholds, the storage monitor informs CICS that it can return to normal operation. As it does so, operator messages inform you when MVS storage is no longer constrained.

You can now run Java programs in a JVM that executes in user key. A new type of open TCB, the J9 TCB, is used for these JVMs. Running applications in user key extends CICS storage protection, and the J9 TCB uses less storage below the line than other TCBs. When you set the EXECKEY parameter on the PROGRAM resource definition for a Java program to USER, a J9 TCB is used for the JVM, and MVS storage is obtained in user key. When the EXECKEY parameter is set to CICS, a J8 TCB is used, and MVS storage is obtained in CICS key.
The default for the EXECKEY parameter is USER. As running applications in user key extend CICS storage protection, it could be beneficial to let most of your Java programs run in a JVM in user key. However, you might need to execute a JVM in CICS key if the program that uses the JVM is part of a transaction that specifies TASKDATAKEY(CICS). If this is the case, you need to invoke the JVM using a PROGRAM resource definition that specifies EXECKEY(CICS).

For enterprise beans, CIRP (the default transaction for REQUESTMODEL definitions) specifies TASKDATAKEY(USER), and the PROGRAM resource definition for DFJIIRP (the default request processor program) specifies EXECKEY(USER), so by default enterprise beans run in user key. The master JVM that initializes the shared class cache is invoked in user key, so that worker JVMs that were invoked in user key can read and write to the shared class cache. Even if all the worker JVMs that share the class cache are invoked in CICS key, the master JVM and the shared class cache are still in user key.

Z/OS SHARED LIBRARY REGION – REDUCING THE AMOUNT OF REAL STORAGE

Another area that presents a tuning opportunity, which most CICS system programmers may not be aware of, is a shared library region z/OS feature that enables address spaces to share Dynamic Link Library (DLL) files. This feature enables CICS regions to share the DLLs that are needed for JVMs, rather than each region having to load them individually. This can greatly reduce the amount of real storage used by MVS, and the time it takes for the regions to load the files.

The amount of storage that is allocated is controlled by the SHRLIBRGNSIZE parameter in z/OS, which is in the BPXPRMxx member of SYS1.PARMLIB. The minimum is 16MB, and the z/OS default is 64MB. The storage that is reserved for the shared library region is allocated in each CICS region when the first JVM is started in the region.
You can issue the command D OMVS,L to display the library statistics and tune down the shared library allocation when appropriate. The DLLs needed for the JVM use around 10MB of the region and you can set the allocation as low as 16MB.

Please note that if you under-allocate SHRLIBRGNSIZE, you will not benefit from the sharing facility, since when the shared library region is full, the files will be loaded into private storage instead.

**USING CICS MONITORING FACILITY TO MONITOR CPU OVERHEAD**

The CICS monitoring facility can be used to monitor the CPU time used by a transaction that invokes a JVM program, including the amount of CPU time used by the JVM on a J8 or J9 TCB. The
CICS monitoring facility also includes the elapsed time spent in the JVM and the number of JCICS API requests issued by the JVM program.

When reviewing JVM reset time, note that for a continuous JVM the JVM reset time is usually zero, but not always. This is because although continuous JVMs are not reset after each use, CICS still requests garbage collection for these JVMs every 101 transactions. This process takes place after the 101st use of the JVM, and garbage collection is recorded as JVM reset time.

Watch out for unnecessary CPU time used by JVMs. In particular, look out for the use of tracing, the use of the USEROUTPUTCLASS option in JVM profiles, and resettable JVMs that are being re-initialized instead of being reset. Setting the CICS master system trace flag on and/or activating JVM trace will significantly increase the CPU cost of a JVM program execution and should not be used in production.

Figure 1 shows the monitoring data fields that can be used for performance monitoring.

![Figure 2: CICS TS V2.2 and V2.3 performance data](image)
PERFORMANCE DATA

John Burgess of IBM Hursley (a member of the CICS Performance Team) documented CICS TS V2.2 and V2.3 performance data, which is shown in Figure 2 and demonstrates a dramatic reduction in CPU, running Java and EJB applications under CICS TS 2.3.

Note: in order to improve JVM start-up performance under CICS TS 2.2, IBM delivered the PQ68286/UQ76199 fix, which made changes to the processing of JVMs within CICS to reduce the time taken to initialize a JVM.

You can reduce the start-up time for your JVMs by adding Xservice="-Xquickstart" to the JVM profiles that are used by your Java programs. At JVM start-up, the -Xquickstart option causes the Just-In-Time (JIT) compiler function of the JVM to run with a subset of optimizations. When the JVM is subsequently reused by other Java programs, the JIT-compiling process runs again to compile the code into its optimal form, and this will degrade performance for those programs. So by specifying -Xquickstart, you are reducing the initial start-up time for your JVMs, but causing a minor performance impact during later reuses of the JVMs.

So, as you can see, there is close to a three times reduction in CPU overhead for a sample EJB application, and a JVM start-up time reduction by 60%, using continuous JVM and shared class cache in comparison to CICS TS 2.2 – making CICS TS 2.3 an excellent choice for developing CICS EJB and Java applications.

SUMMARY

For shops looking to Web-enable CICS applications, using Java under CICS and CICS Connectors simplifies development of new e-business applications, reduces the overall cost of computing, and allows reuse of existing programming and operational skills without requiring a fundamental re-education of the system programmers and operators. CICS TS 2.3 not only provides functional enhancements in Java functionality under
CICS, but also addresses the issue of performance, which has been inhibiting some shops from making use of CICS EJB server, because of the very high CPU overhead.

CICS TS 2.3 provides a robust high-performance environment for enterprise applications written in Java. It exploits SDK V1.4.1 and has an architecture which ensures that Java applications have a high degree of isolation from each other. Java applications running under CICS now have reduced storage and start-up requirements for each JVM and enhanced execution brought about by improve behaviour under stress. CICS also enables construction of reusable business logic components that are binary portable between CICS and WebSphere and may be deployed in either environment using the same tools, by exploiting the WebSphere EJB container.

Elena Nanos  
IBM Certified Solution Expert in CICS Web Enablement  
Zurich NA (USA)  
© Xephon 2004

Investigating storage violations

This CICS Global User Exit will collect in an in-core table all the ‘linked to’ programs in CICS key.

The reason for writing this exit is that over the years and with different CICS releases, we have suffered a storage violation four to six times each year. As all storage protection features are ON, the program forcing the violation must be in CICS key.

The violation hits CICS domains like program, load, TS, in the chaining of resource entries, and system X'85' TIOAs.

When the violation occurs, a sysdump is taken by CICS, and in this sysdump we can investigate the in-core table and see the names of the last used programs in CICS key. There are two tables: one collects the programs as a summary, the other
collects the programs as a single entry in a wraparound table. Programs in CICS key and not linked to should get an entry saying 'Link to a CICS key program' (if the source is available...). I have never seen DFH-named programs collected by the exit.

ASSEMBLER PROGRAM CODE

*ASM CICS (SP)
EPCC TITLE 'CICS PC EXIT GLUEXPCC XPCREQ '
MACRO
&LABEL CICWTO &MSG MESSAGE
.* CHECK THE MESSAGE PARAMETER
AIF ('&MSG' EQ '').NOTE1 MISSING MESSAGE
.* THE ADDRESS OF THE MESSAGE HAS BEEN SPECIFIED
LA 1,&MSG ADDRESS THE MESSAGE
DS ØH
WTO MF=(E,(1)) ISSUE WTO SVC
MEXIT
.* PARAMETER ERROR
.NOTE1 ANOP
MNOTE 2Ø,'MESSAGE MISSING'
MEXIT ,
MEND
EJECT

***********************************************************************
* CICS GLUE EXIT: XPCREQ INVOKED BEFORE A PROGRAM LINK IS PROCESSED
* CICS RELEASE: 5.3 / 6.2
* PROGRAM NAME: GLUEXPCC
* REASON: FIND THE LAST DRIVEN CICS-KEY PROGRAMS OF A STORAGE
* VIOATION SYSDUMP
* PURPOSE: COLLECT CALLER AND CALLED PROGRAM NAMES
* OF CICS-KEY PROGRAMS IN A WRAP AROUND TABLE 'COLLTABL'
* THE LAST DATA-LINE FOLLOWS A WRAP-LINE TO FIND THE WRAP
* POINT OF THE TABLE (THE WRAP POINT CAN BEE SEEN TOO
* IN THE EIBTIME TABLE ENTRY)
* COLLECT CICS-KEY PROGRAM USAGE COUNT IN A TABLE 'OFTLTABL'
* SOLUTION: COLLECT REQUESTS IN 2 CWA CHAINED OFF IN-STORAGE TABLES
* CWAPINTER: CWAEXIPL
* CALL ENVIRONMENT: API + XPI
* START AND STOP THE EXIT
* 1. CEMT COMMAND ENABLE... DISABLE... PROGRAM (GLUEXPCC)
* EXIT (XPCREQ) GALENGTH (2Ø48) START / STOP
* SAVE THE COLLECTED DATA: A PLTSD PROGRAMS PRODUCES A TRANSACTION
* DUMP OF THE TABLE 'OFTLTABL' CONTAINING PROGRAM SUMMARY USAGE
* VIEW THE COLLECTED DATA ONLINE: AN ONLINE CORE DISPLAY TRANSACTION
* DISPLAYS THE COLLECTED DATA, BASED ON THE POINTER IN CWA FIELD
* CWAEXIPL (IF TRANSACTION IS AVAILABLE)
**PROGRAM REGISTER USAGE:**

- **RØ** - WORK REGISTER
- **R1** - POINTS TO DFHUEPAR PLIST ON ENTRY
  - WORK REGISTER
- **R2** - DFHUEPAR PARAMETER LIST AFTER ENTRY
- **R3** - CODE BASE REGISTER 1
- **R4** - CWA
- **R5** - XPI PARAMETER LIST
- **R6** - UNUSED
- **R7** - CODE BASE REGISTER 2
- **R8** - COMMAND PARAMETER LIST UEPCLPS
- **R9** - DSECT COLTABL COLLECT TABLE STORAGE PER QUEUE
- **R10** - DATAREG FOR DFHEISTG
- **R11** - EIBREG
- **R12** - DSECT TABSTORA COLLECT TABLE STORAGE HEADER AND QUEUES
- **R13** - ORIGINAL SAVE AREA
  - KERNEL STACK FOR XPI CALLS
- **R14** - WORK REGISTER
- **R15** - WORK REGISTER

**DSECTS FOR COLLECT-TABLE STORAGE**

**HEADER AND ENTRIES PER LINK REQUEST**

Always define pairs of 8: to get a good view into storage space.

**TABSTORA DSECT**

- **TABEYECA DS** D 8 EYECATCHER
- **TABJOBNA DS** CL8 8 JOBNAME
- **TABDISP DS** A DISPLCMENT TO TABTCOMA
- **TABEMPTY DS** CL12 16 EMPTY SPACE
- **TKBFIENT DS** CL4 EIBTIME FIRST ENTRY EYECATCHER
- **TABFIENT DS** CL4 8 EIBTIME FIRST ENTRY
- **TKBLAENT DS** CL4 EIBTIME LAST ENTRY EYECATCHER
- **TABLAENT DS** CL4 8 EIBTIME LAST ENTRY
- **TKBDATE DS** CL4 EIBDATE FIRST ENTRY EYECATCHER
- **TABDATE DS** CL4 8 EIBDATE FIRST ENTRY
- **PKEEXIRC DS** CL1 RC OF PREVIOUS EXIT EYECATCHER
- **PREEXIRC DS** CL2 RC OF PREVIOUS EXIT
  - DS CL5 8 EMTPY, FILLER TO DOWO SPACE
- **PROGINEY DS** CL8 8 INVOKING PROGRAM EYECATCHER
**TABLE ENTRIES**

* COLLECTED LINKS TABLE, PROGRAM SUMMARY

SPACE

TABOCOMC DS CL16 88  EYE CATCHER BEGIN OF TABLE

SPACE

*  S U M M E N T R I E S  LENGTH

LCLA &O

&O  SETA 32  LENGTH OF TABLE ENTRY

SPACE

*  S U M M E N T R I E S  PHYSICAL

TABOCOMA DS ØCL&O *COLLECTED PROG*  TABLE FOR COLLECTED PROGRAMS

QSENTRY DS ØCL32  OFTL ENTRY

QSID DS CL4  TABLE ID OFTL

QSCALLED DS CL8  CALLED PROGRAM NAME

QSCALLER DS CL8  CALLERS PROGRAM NAME

QSCOUNT DS PL4  UNUSED

QSTASKN DS CL4  TASK NUMBER

QSTIME DS CL4  TIME

SPACE

*  FURTHER TABLE ENTRIES

DS 1ØCL&O

DS 6ØCL&O

DS 2ØCL&O  FURTHER TABLE ENTRIES TEST

DS Ø6CL&O  FURTHER TABLE ENTRIES TEST

TABOCOME DS CL&O  TABLES END END MARKER

SPACE 2

* COLLECTED LINKS TABLE, SINGLE LINK
**EYE CATCHER BEGIN OF TABLE**

**LINK ENTRIES LENGTH**

LCLA &A

&A SETA 32

**LENGTH OF TABLE ENTRY**

**LINK ENTRIES PHYSICAL**

TABTcoma DS ØCL&A *COLLECTED LINKS* TABLE FOR COLLECTED LINKS

PSSysid DS CL4 SYSID OF CICS

PSCalled DS CL8 CALLED PROGRAM NAME

PSCaller DS CL8 CALLERS PROGRAMNAME

PSTaskn DS CL4 TASK NUMBER

PSDate DS CL4 DATE

PSTime DS CL4 TIME

**SPACE**

DS 4999CL&A FURTHER TABLE ENTRIES CICS5 4000

* DS 20CL&A FURTHER TABLE ENTRIES TEST

* DS Ø6CL&A FURTHER TABLE ENTRIES TEST

TABTcoma DS CL&A TABLES END END MARKER

**SPACE**

TABstore DS ØD END OF TABSTORA

**SPACE**

**ENTRIES FOR COLLECTED LINKS**

**BASED ON AND TO BE KEPT IN STEP WITH TABTcoma**

**LINK ENTRIES LOGICAL**

COLLtabl Dsect COLLECTING LINKS

DSKey DS ØCL2Ø KEY OF TABLE ) KEY

DSSysid DS CL4 SYSID OF CICS ) KEY

DSCalled DS CL8 CALLED PROGRAM NAME ) KEY

DSCaller DS CL8 CALLERS PROGRAMNAME ) KEY

DSTaskn DS CL4 TYPE OF CALL LINK OR XCTL

DSDate DS CL4 DATE

DSTime DS CL4 TIME

**SPACE**

**SUMM ENTRIES LOGICAL**

OFTLTABL Dsect COLLECTING LINKS SUMMARY

OSENTRY DS ØCL32 OFTL ENTRY

OSid DS CL4 TABLE ID OFTL

OSCalled DS CL8 CALLED PROGRAM NAME

OSCaller DS CL8 CALLERS PROGRAMNAME

OScount DS PL4 UNUSED

OSTaskn DS CL4 TYPE OF CALL Link OR XCTL

OSTime DS CL4 TIME

**SPACE**

**PREDEFINED**

* PROGRAM NAMES LOGICAL

Daboname Dsect COLLECTING LINKS SUMMARY

Dabonami DS CL8 PREGENERATED PROGRAM NAMES

**SPACE**
EJECT
*---------------------------------------------------------------------*
* CWA LAYOUT                                                      *
*---------------------------------------------------------------------*
SPACE
CWADSECT DSECT
CWAKENN DS CL18 JOBNAME FILLED IN PLTPI
CWASYSID DS CL4 SYSID FILLED IN PLTPI
CWAEXIPL DS F GETMAIN POINTER
SPACE
EJECT
*---------------------------------------------------------------------*
* THE FOLLOWING DEFINITIONS ARE FOR PROGRAM WORKING STORAGE.  *
*---------------------------------------------------------------------*
SPACE
DFHEISTG DSECT
SPACE
EISTGCA DS CL16'GLUEXPCDFHEISTG' EYECATCHER
SPACE
RETCODE DS XL4 EXIT RETURN CODE
SPACE
GETMRESP DS F GETMAIN RESP
GETMRES2 DS F GETMAIN RESP2
SPACE
SAVEINQU DS F SAVE R14 PROGINQU
SAVEINKU DS F SAVE R14 PROGINQU
SAVEMSG DS F SAVE R14 ROUTINE MSG
SAVEMSG1 DS F SAVE R1 ROUTINE MSG
SAVELIRE DS F SAVE R14 ROUTINE PC_REQUEST...
SPACE
ZSPROGIN DS CL8 INVOKING PROGRAM XPI CALL
ASZPINVO DS CL8 INVOKING PROGRAM XPI CALL
ASZPRETU DS CL8 RETURNING PROGRAM XPI CALL
SPACE
SYSICURR DS CL4 CURRENT SYSID
LINKCURR DS CL8 CURRENT LINK TO PROGRAM
INVOCURR DS CL8 CURRENT INVOKING PROGRAM
SPACE
SWTABFUL DS X TABLE IS FULL
SPACE
PCGROUP DS C COPY FROM PC_GROUP
PCFUNCT DS C COPY FROM PC_FUNCT
SPACE
RØ74RESP DS F PPT INQUIRE RESP
RØ74LANG DS F PPT INQUIRE LANG
RØ74KEYY DS X PPT INQUIRE KEY
RØ74TOKE DS F PPT INQUIRE TOKEN
RØ74PROG DS CL8 PPT INQUIRE PROGRAM CURRENT
RØ74PROK DS CL8 PPT INQUIRE PROGRAM
* AID FOR TEST, PRINT R9 VALUE AS A CONSOLE MESSAGE

SPACE

R9PRSAVE DS F UPRO REGISTER SAVE
R9PRRSF9 DS F, CL1 R9 REMEMBER
R9PRRESP DS F R9 WTO RESP
R9PRLENG DS F R9 WTO LAENGE

SPACE

R9PRMESS DS ØCL3Ø MESSAGE
R9PRMESØ DS CL1Ø ' GLUEXPCC ' EXIT PROGRAM NAME
R9PRMES1 DS C'R9=' HEADER
R9PRMES2 DS CL8' ' R9 VALUE HEX
R9PRMES3 DS C FILLER + UNPK SIGN
R9PRMES4 DS CL2 LOGICAL POINT OF CALL
R9PRMES5 DS C FILLER

ORG R9PRMESS+L'R9PRMESS

SPACE

* STANDARD MESSAGE BOX, COPIED FROM MSGBOX TO DFHEISTG AND FILLED THERE
* KEEP IN STEP WITH MSGBOX

DS ØF

MSGEISA DFHMSG (MSGEISØ1, CLØ8'GLUEXPCC', NAME OF EXIT X
               MSGEISØ2, CLØ1' ', FILLER X
               MSGEISØ3, CLØ4'SYSI', SYSID X
               MSGEISØ4, CLØ1' ', FILLER X
               MSGEISØ5, CL4Ø'MSGBOX'MSGBOX') TEXT

MSGEISE DS ØF END OF MESSAGE BOX

EJECT

*-----------------------------------------------*
* COPYBOOK AND DSECTS REQUIRED BY THE EXIT PROGRAM
*-----------------------------------------------*

SPACE

DFHUEXIT TYPE=EP,ID=(XPCREQ) EXIT SPECIFIC PARAMETERS
SPACE
DFHUEXIT TYPE=XPIENV EXIT PROGRAMING INTERFACE (XPI)
SPACE
COPY DFHXMIQY XPI INQUIRE TRANSACTION CALL
SPACE
COPY DFHSAIQY XPI INQUIRE SYSTEM CALL
SPACE
COPY DFHPGISY XPI INQUIRE CURRENT PROGRAM CALL
SPACE
COPY DFHPCEDS COMMAND LEVEL PLIST DEFINITIONS
SPACE
EJECT

*************************************************************************************
* GLUEXPCC - MAIN ROUTINE
*************************************************************************************

SPACE

GLUEXPCC DFHEIENT CODEREG=(R3,R7), DATAREG=(R1Ø), EIBREG=(R11)
SPACE
GLUEXPCC AMODE 31
GLUEXPCC RMODE ANY
SPACE
* ADDRESS INCOMING PARAMETERS
SPACE
LR R2,R1 DFHUEPAR PLIST PROVIDED BY CALLER
USING DFHUEPAR,R2 USE R2 TO ADDRESS UEPAR PLIST
SPACE
MVC EISTGCA,=CL16'GLUEXPCCDFHEISTG' EYECATCHER DFHEISTG
SPACE
LA R14,URECNORM SET OK RESPONSE
ST R14,RETCODE IN WORKING STORAGE
SPACE
SR R12,R12 DE-ADDRESS TABSTORA
SR R9,R9 DE-ADDRESS COLLTABL
B CPEXIØØØ
SPACE
DC C'***I AM GLUEXPCC***' EYECATCHER VASS
SPACE
* INIT VARIABLES IN DFHEISTG
SPACE
CPEXIØØØ DS ØH
MVC SYSICURR,BLANK BLANK TILL NOW
MVI SWTABFUL,Ø * TABLE NOT YET FULL
SPACE
* ADDRESS EIB
* MAY BE NOT NECESSARY IF NO EXEC CICS CALLS FOLLOW
* OR IF EIB FIELDS ARE NOT USED IN THE TEXT FOLLOWING
EXEC CICS ADDRESS EIB(R11)
USING DFHEIBLK,R11
SPACE
* ASSIGN PROGRAMNAMES
* MAY BE XPI DOES IT BETTER
* XPI DOES THE SAME, I USE XPI
B CPEXIØ2Ø OVERJUMP SPACE
EXEC CICS ASSIGN INVOKINGPROG (ASZPINVO) X
RETURNPROG (ASZPRETU)
CPEXIØ2Ø DS ØH
SPACE
* COPY MESSAGE BOX, CHECK LENGTH BEFORE..
SPACE
DS ØH CHECK FOR LENGTH EQUAL
CLC =AL2(MSGBOXE-MSGBOXA),=AL2(MSGEISE-MSGEISA)
BNE ERROR14 LENGTH NOT CORRECT
DS ØH
MVC MSGEISA(MSGBOXE-MSGBOXA),MSGBOXA
MVC MSGEISØ5,BLANK
* CHECK UEPGAA, UEPGAL *EXIPCC*

SPACE

L R14,UEPGAL ADDRESS OF GAA LENGTH
LTR R14,R14 EXISTENCE..
BZ ERROR47 NO, BRANCH TO ERROR ROUTINE
SPACE

LH R15,Ø(R14) GET GAA LENGTH
CH R15,=AL2(2Ø48) GET DATA LENGTH
BL ERROR48 TOO SMALL
SPACE

L R14,UEPGAA ADDRESS OF GAA
LTR R14,14 EXISTENCE
BZ ERROR49 NO, BRANCH TO ERROR ROUTINE
SPACE

* ADDRESS CWA PER XPI CALL 1. ENTRY, UEPGAA AFTER 1. ENTRY
SPACE

DS ØH
L R15,UEPGAA STORE CWA ADDRESS IN UEPGAA
CLC ØØ(8,R15),=CL8'GLUEXPCC' EXITPROGRAM NAME EXISTENT
BNE CPEXIØ3Ø NO, DO XPI CALL
ICM R4,15,24(R15) CWA ADDRESS
B CPEXII1ØØ
SPACE

CPEXIØ3Ø DS ØH GET CWA ADDRESS PER XPI CALL
DS ØH
L R5,UEPXSTOR ADDRESS XPI PARAMETER LIST
USING DFHSAIQ_ARG,R5
L R13,UEPSTACK ADDRESS KERNEL STACK
DFHSAIQX CALL,

CLEAR,
IN, IN
FUNCTION(INQUIRE_SYSTEM),
OUT, OUT
CWA((R4)),
RESPONSE(*),
REASON(*)

SPACE

USING CWADSECT,R4
SPACE

DS ØH
L R15,UEPGAA STORE CWA ADDRESS IN UEPGAA
MVC ØØ(8,R15),=CL8'GLUEXPCC' EXITPROGRAM
MVC ØØ(8,R15),=CL8'XPCREQ ' EXITNAME
MVC 16Ø(8,R15),=CL8'CWAADRES' CWA ADDRESS EYECATCHER
STCM R4,15,24(R15) CWA ADDRESS
SPACE

L R13,UEPEPSA ADDRESS ORIGINAL SAVE AREA
SPACE
CLI   SAIQ_RESPONSE,SAIQ_OK   CHECK RESPONSE
BE   CPXIIØØ
* *** CLI   SAIQ_REASON,SAIQ_.....  CHECK RESPONSE REASON
* *** DS   ØH   REASON IS NOT CHECKED
* *** BE   *
DROP R5
B     ERROR1Ø   SEND ERROR MESSAGE
SPACE
* CHECK FOR PROPER CWA LAYOUT
SPACE
CPXIIØØ DS   ØH
CLC   CWAKENN+10(2),KTPXYCIC   CWA OK   JOBNAME
BNE   ERROR11   NO
CLC   CWAKENN+14(3),KTPXYCIC+4   CWA OK   JOBNAME
BNE   ERROR12   NO
SPACE
MVC   SYSICURR,CWASYSID   SYSID IN CURRENT KEY
MVC   MSGEISØ3,CWASYSID   SYSID IN MESSAGE
SPACE
* CHECK FOR POSSIBLE RECURSION
L     R14,UEPRECUR   ADDRESS OF RECURSIVE COUNT
LH    R14,Ø(R14)   FETCH COUNT
LTR   R14,R14   HAS EXIT BEEN INVOKED RECURSIV
BNZ   ERROR5   YES, BRANCH TO ERROR ROUTINE
SPACE
* CHECK FOR PROPER EXIT ID
SPACE
L     R14,UEPEXN   ADDRESS OF 1 BYTE EXIT ID
CLI   Ø(R14),XPCREQ   IS THIS XPCREQ  EXIT?
BNE   ERROR1   NO, BRANCH TO ERROR ROUTINE
SPACE
* GETMAIN FOR COLLECT-TABLE PER XPI CALL REGISTER R12
* ONLY ONE TIME, THEN SAVE ADDRESS IN CWAEXIPL
* JOB LIFETIME STORAGE
* GETMAIN DONE BY EXEC CICS API TO AVOID FETCH PROTECTIONS
SPACE
USING TABSTORA,R12
CLC   CWAEXIPL(2),=CL2'**'   LAST TIME AN ERROR
BE   CPXI21Ø   YES
OC   CWAEXIPL,CWAEXIPL   ALREADY GETMAINED
BZ   CPXI21Ø   NO
L     R12,CWAEXIPL   YES, LOAD USING REGISTER
B     CPXI295
SPACE
CPXI21Ø DS   ØH
XC   CWAEXIPL,CWAEXIPL   CLEAR, IF LAST TIME AN ERROR
SPACE
EXEC   CICS GETMAIN   *
LENGTH   (TABSTORL)   *
INITIMG   (X'ØØ')   *

SET (R12) *
NOSUSPEND *
SHARED *
RESP (GETMRESP) *
RESP2 (GETMRES2)

CLC GETMRESP,DFHRESP(NORMAL)
BE CPEXI22Ø

SPACE

MVC CWAEXIPL(2),=CL2'**' REMEMBER ERROR CODE
MVC CWAEXIPL+2(1),GETMRESP+L'GETMRESP-1
MVC CWAEXIPL+3(1),GETMRES2+L'GETMRES2-1
B ERROR13 SEND ERROR MESSAGE

SPACE

TABSTORL DC AL4(TABSTORE-TABSTORA) GETMAIN LENGTH
TABENDMA DC &A.X'FF' END MARKER OF STORAGE

* SAVE STORAGE ADDRESS IN CWA, INIT STORAGE

SPACE

CPEXI22Ø DS ØH

SPACE

MVC MSGEISØ5(L'MSGTEØ1),MSGTEØ1 TELL ABOUT THIS ACTION
BAS R14,MSG ONE TIME MESSAGE...
B CPEXI23Ø GO TO GETMAIN INIT TAB STORAGE

MSGTEØ1 DC C'CWA ADDRESSED, GETMAIN DONE'

SPACE

CPEXI23Ø DS ØH

ST R12,CWAEXIPL REMEMBER STORAGE ADDRESS

SPACE

SPACE

DS ØH INIT HEADER
DS ØH EYECATCHER, VALUES
DS ØH

ZAP RFENR,=P'Ø' INIT ENTRY COUNTER
ZAP RFTABFU,=P'Ø' INIT TABLE FULL COUNTER
ZAP RFTABVO,=P'Ø' INIT TABLE FULL COUNTER
ZAP RFERROR,=P'Ø' INIT ERROR COUNTER
ZAP RFERROR1,=P'Ø' INIT ERROR COUNTER, UNUSED
ZAP RFENTRY,=P'Ø' INIT USED ENTRIES COUNTER
ZAP RFEMPT1,=P'Ø' INIT RFEMPT1
ZAP RFEMPT2,=P'Ø' INIT RFEMPT2
ZAP RFPROKLI,=P'Ø' INIT RFPROKLI

SPACE

MVC TABEYECA,=CL8'GLUEXPCC' EYECATCHER
MVC TABJOBNA,CWAKENN+1Ø JOBNAME
MVC TABTCOMC,=CL16'**BEGIN OF TABLE**'
MVC TABOCOMC,=CL16'#$BEGIN OF TABLE#'

SPACE

OC TABDATE,TABDATE DATE FIRST ENTRY
BNZ  *+14
MVC  TBKDATE,=CL4'DATE'       DATE FIRST ENTRY
MVC  TABDATE,EIBDATE         DATE FIRST ENTRY
SPACE
OC  TABFIENT,TABFIENT       TIME FIRST ENTRY
BNZ  *+14
MVC  TBKFIENT,=CL4'TIMF'     TIME FIRST ENTRY
MVC  TABFIENT,EIBTIME        TIME FIRST ENTRY
SPACE
MVC  PROGINEY,=CL8'PROGINEY' PROG NAME INVOKING
MVC  PROGLIEY,=CL8'PROGLIEY' PROG NAME LINK TO
MVC  PROKLIEY,=CL8'PROKLIEY' PROG NAME LINK TO INKU ERROR
SPACE
MVC  PROGINVO,=CL8'INVOKING' INIT VALUE PROG NAMES INVOKING
MVC  PROGLINK,=CL8'LINKTOPR' INIT VALUE PROG NAMES LINK TO
MVC  PROKLINK,=CL8'INKUPROK' INIT VALUE PROG NAMES LINK TO
SPACE
MVC  LASTEYEC,=CL8'LASTCALL' INIT LAST CALL VARIABLES
MVC  LASTPOIN,=CL4'POIN'     LAST TABLE POINTER
MVC  LASTSYSI,=CL4'LSYS'     LAST SYSID
MVC  LASTCALD,=CL8'LASTCALD' LAST CALLED PROGRAM NAME
MVC  LASTCALR,=CL8'LASTCALR' LAST CALLER PROGRAM NAME
SPACE
DS  ØH                       INIT ENDMARKER SUMM TABLE
LA  R14,TABOCOMA             BEGIN OF TABLE  SUMM TABLE
A  R14,=AL4(TABOCOME-TABOCOMA) ADDRESS TABLE END
MVC  Ø(L'TABOCOME,R14),TABENDMA INIT ENDMARKER
SPACE
DS  ØH                       INIT ENDMARKER LINK TABLE
LA  R14,TABTCOMA             BEGIN OF TABLE  LINK TABLE
A  R14,=AL4(TABTCOME-TABTCOMA) ADDRESS TABLE END
MVC  Ø(L'TABTCOME,R14),TABENDMA INIT ENDMARKER
SPACE
MVC  TABDISP,=A(TABTCOME-TABEYEC) DISPLCMENT ZU TABTCOMA
SPACE
DS  ØH                       INIT ENDMARKER COLLECT-SUMM-TABLE
DS  ØH                       INIT ENDMARKER COLLECT-TABLE
LA  R1,TABOCOMA              ADDRESS OF COLLECT-TABLE
USING OFTLTABL,R1
SPACE
CPEXI243 DS  ØH                       INIT COLLECT-TABLE LOOP
MVC  OSID,=CL4'OFTL'         TABLE ID
MVC  OSCALLED,KEMPTY        STILL EMPTY CALLED PROGRAM
MVC  OSCALLER,KEMPTY        STILL EMPTY CALLING PROGRAM
ZAP  OSCOUNT,=P'Ø'          USAGE COUNT
ZAP  OSTASKN,=P'Ø'          TASKNUMBER
ZAP  OSTIME,=P'Ø'           EIBTIME
AH  R1,=AL2(L'TABOCOMA)     ADDRESS NEXT ENTRY

LA R14,=AL4(TABOCOMA) BEGIN OF TABLE
A R14,=AL4(TABOCOMA-TABOCOMA) ADDRESS TABLES END
CLC Ø(L'TABOCOMA,R1),Ø(R14) END OF TABLE
BNE CPExi243 NO
DROP R1
SPACE

CPExi245 DS ØH INIT COLLECT-SUMM-TABLE
DS ØH WITH PREDEFINED PROGRAM NAMES
DS ØH
USING OFTTABL,R1 COLLECT-SUMM-TABLE
USING DABONAME,R15 PROGRAM NAME-TABLE
LA R1,TABOCOMA ADDRESS OF COLLECT-TABLE
L R15,=A(TABONAMØ) ADDRESS OF PROGRAMNAMES
SPACE

CPExi247 DS ØH INIT COLLECT-SUMM-TABLE
MVC OSCALLED,DABONAM1 PREDEFINED NAME
SPACE
AH R1,=AL2(L'TABOCOMA) ADDRESS NEXT ENTRY
AH R15,=AL2(L'DABONAM1) ADDRESS NEXT ENTRY
SPACE
LA R14,TABONAMØ BEGIN OF TABLE NAME-TABLE
A R14,=AL4(TABONAMZ-TABONAMØ) ADDRESS TABLES END
BE CPExi249 YES
SPACE

CPExi248 DS ØH INIT COLLECT-SUMM-TABLE
MVC MSGEISØ5(L'MSGTEØ2S),MSGTEØ2S
BAS R14,MSG
SPACE

CPExi249 DS ØH INIT COLLECT-SUMM-TABLE
SPACE
DROP R1
DROP R15
SPACE
DS ØH INIT COLLECT-LINK-TABLE
DS ØH
LA R9,TABTCOMA ADDRESS OF COLLECT-TABLE
USING COLLTABL,R9
SPACE

CPExi293 DS ØH INIT COLLECT-TABLE LOOP
MVC DSSYSID,CWASYSID SYSID
MVC DSCALLED,KEMPTY STILL EMPTY CALLED PROGRAM
MVC DSCALLER,KEMPTY STILL EMPTY CALLING PROGRAM
Editor's note: the code for this article will be concluded next issue.

Hannes Bojan
CICS Systems Programmer (Germany) © Xephon 2004

Articles, or ideas for an article, should be e-mailed to Trevor Eddolls at trevore@xephon.com. A copy of our Notes for Contributors is available at www.xephon.com/nfc.
Micro Focus has formed a partnership with Unilog to port mainframe applications, such as CICS, to the Microsoft Windows platform. This, they claim, will enable existing COBOL applications to run at higher performance levels on low-cost platforms. Micro Focus sells COBOL application development and deployment software, and Unilog is an IT systems integrator.

To prove that the technology works, they have migrated the time management system for Eurocopter, a helicopter manufacturer based in Germany. Eurocopter were using a COBOL-based ZINA time management system on the mainframe. They now run ZINA and CICS on PCs.

For further information contact:
Micro Focus, 9420 Key West Avenue, Rockville, MD 20850, USA.
Tel: (301) 838 5000.

* * *

NEON Enterprise Software, which develops CICS and IMS products allowing the management of enterprise data, has announced additional ways customers can benefit from its simplified pricing model.

Streamlined pricing, based on broad levels of CPU capacity, is one method. Most mainframe software companies price their products so there are large price increases for each minor increase in MIPS capacity.

The new simplified pricing structure allows for growth across broad spans of machine capacity. Now, a business can increase hardware MIPS without incurring software upgrade fees common to the mainframe software industry.

For further information contact:
NEON Systems, 14100 Southwest Freeway, Suite 500, Sugarland, TX 77478, USA.
Tel: (281) 491 4200.

* * *

Wily Technology, which supplies J2EE portal management solutions and claims to be first to market with software to manage integration connections to CICS and Tuxedo, has turned its attention to business workflows.

The company is looking to fill the gap for developers and architects looking for ways to get more visibility out of their J2EE-based integrated legacy networks.

To begin to address the problem, Wily is offering Wily 5, an upgraded application management platform that includes enhancements to its Introscope line of PowerPack probes and components. The key Introscope’s upgrade is the ability to monitor dataflows inside Java adapters/connectors. In this latest release, Wily 5 supports IBM’s Websphere Business Integration adapters. Other features include: customizable dashboards to view availability and data/workflow information down to granular level in real time; additional systems management, component-based visibility, a native reporting system, fast-path user interface navigation, and integration with enterprise security solutions.

For further information contact:
Wily Technology, 8000 Marina Boulevard, Suite 700, Brisbane, CA 94005, USA.
Tel: (415) 562 2000.

* * *