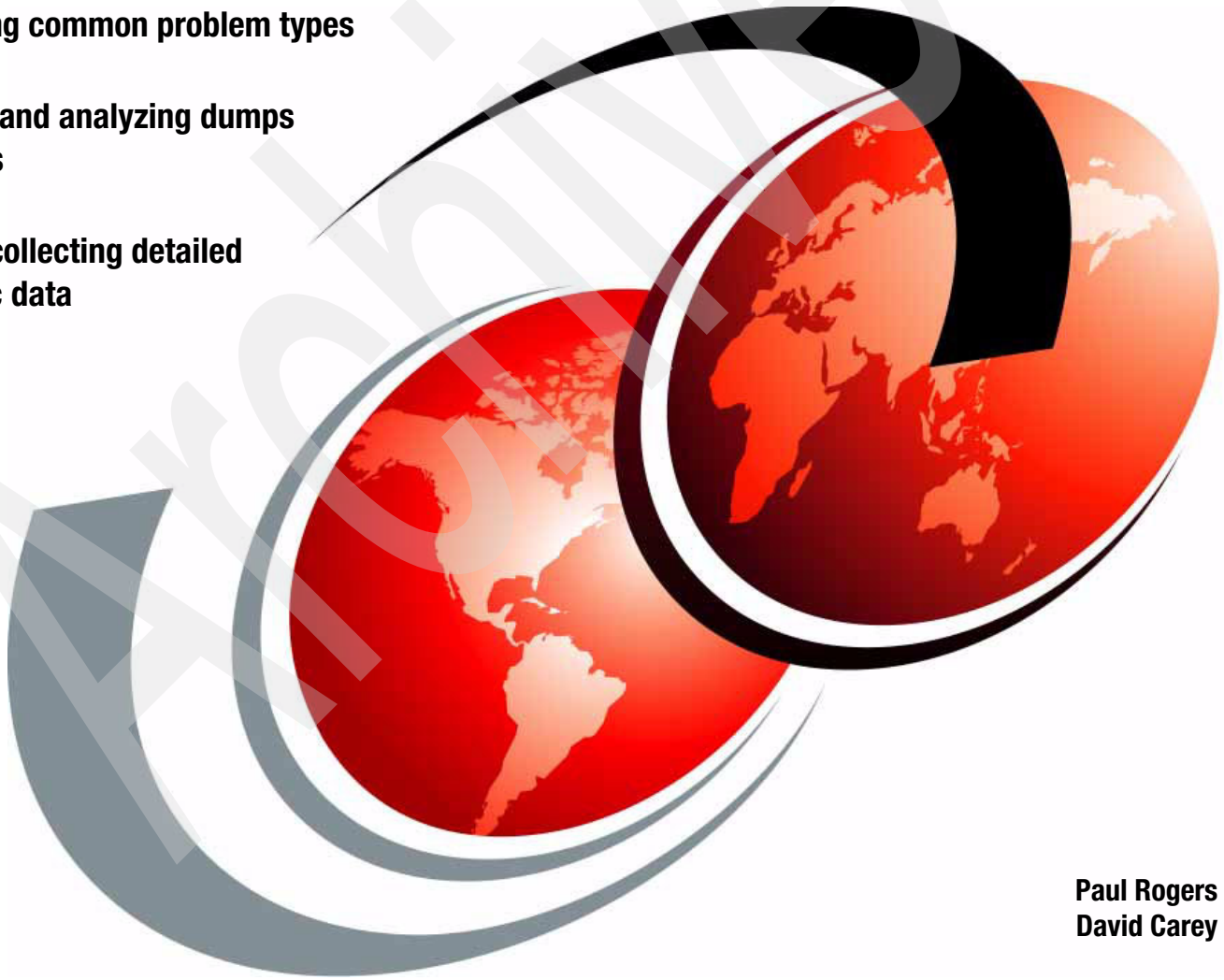
IBM

z/OS Diagnostic Data Collection and Analysis

- Diagnostic fundamentals and recognizing common problem types
- Obtaining and analyzing dumps and traces
- Tools for collecting detailed diagnostic data



Paul Rogers
David Carey

Redbooks

ibm.com/redbooks

International Technical Support Organization

z/OS Diagnostic Data Collection and Analysis

August 2005

SG24-7110-00

Note: Before using this information and the product it supports, read the information in “Notices” on page xi.

First Edition (August 2005)

This edition applies to Version 1, Release 6, of z/OS™ (5694-A01) to Version 1, Release 6, of z/OS.e™ (5655-G52), and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing, IBM Corporation, North Castle Drive Armonk, NY 10504-1785 U.S.A.

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:
This information contains sample application programs in source language, which illustrates programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of the International Business Machines Corporation in the United States, other countries, or both:

@server®
@server®
Redbooks (logo) ™
z/Architecture™
z/OS®
AIX®
CICS/ESA®
CICS®
CICSPlex®
CUA®
DB2®
DFSMS/MVS®
DFSMSdfp™
Encina®
Enterprise Systems Architecture/390®
FFST™
Infoprint®
IBM®
IMS™
Language Environment®
MQSeries®
MVS™
MVS/ESA™
MVS/XA™
OS/390®
Redbooks™
RACF®
RETAIN®
RMF™
SecureWay®
System/360™
System/370™
TCS®
TXSeries®
VTAM®
WebSphere®

The following terms are trademarks of other companies:

Java, Java Naming and Directory Interface, JavaScript, JDBC, JDK, JVM, J2EE, RSM, Solaris, Sun, Sun Microsystems, SNM, and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows NT, Windows, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

MCS, Intel logo, Intel Inside logo, and Intel Centrino logo are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Preface

This IBM® Redbook describes problem diagnosis fundamentals and analysis methodologies for the z/OS® system. It provides guidelines for the collection of relevant diagnostic data, tips for analyzing the data, and techniques to assist in identifying and resolving of Language Environment®, CICS®, CICSPlex/SM, MQSeries®, VTAM®, and DB2® problems. Also described are some diagnostic procedures that are not purely z/OS, but that are related to the various platforms (UNIX® and Windows®) where IBM software executes and interacts with z/OS in a Client/Server or distributed framework topology.

This document shows you how to:

- Adopt a systematic and thorough approach to dealing with problems
- Identify the different types of problems
- Determine where to look for diagnostic information and how to obtain it
- Interpret and analyze the diagnostic data collected
- Escalate problems to the IBM Support Center when necessary

Diagnostic data collection and analysis is a dynamic and complex process. This redbook shows you how to identify and document problems, collect and analyze pertinent diagnostic data and obtain help as needed, to speed you on your way to problem resolution.

The team that wrote this redbook

This redbook was produced by a team of specialists from around the world working at the International Technical Support Organization, Poughkeepsie Center.

Paul Rogers is a Consulting IT Specialist at the International Technical Support Organization, Poughkeepsie Center. He writes extensively and teaches IBM classes worldwide on various aspects of z/OS JES3, Infoprint® Server, zFS, and z/OS UNIX. Before joining the ITSO 18 years ago, Paul worked in the IBM Installation Support Center (ISC) in Greenford, England, providing OS/390® and JES support for IBM EMEA and the Washington Systems Center in Gaithersburg, Maryland.

David Carey is a Senior IT Advisory Specialist with the IBM Support Center in Sydney, Australia, where he provides defect and non-defect support for CICS, CICSPlex/SM, the WebSphere® MQ family of products, and z/OS. David has been working in the IT industry for 25 years and has written extensively about diagnostic processes for the ITSO.

Become a published author

Join us for a two- to six-week residency program! Help write an IBM Redbook dealing with specific products or solutions, while getting hands-on experience with leading-edge technologies. You'll team with IBM technical professionals, Business Partners and/or customers.

Your efforts will help increase product acceptance and customer satisfaction. As a bonus, you'll develop a network of contacts in IBM development labs, and increase your productivity and marketability.

Find out more about the residency program, browse the residency index, and apply online at:

ibm.com/redbooks/residencies.html

Comments welcome

Your comments are important to us!

We want our Redbooks™ to be as helpful as possible. Send us your comments about this or other Redbooks in one of the following ways:

- Use the online **Contact us** review redbook form found at:

  ibm.com/redbooks

- Send your comments in an email to:

  redbook@us.ibm.com

- Mail your comments to:

  IBM Corporation, International Technical Support Organization
  Dept. HYJF Mail Station P099
  2455 South Road
  Poughkeepsie, NY 12601-5400

1

z/OS problem diagnosis fundamentals

This chapter describes, at a high level, the steps to identify and resolve problems in a z/OS environment.

Most businesses that make significant use of computers have a staff of people who diagnose software problems that occur while running the operating system. These people are usually systems programmers for the installation.

If an installation does not wish to debug the problem or does not have the source code involved in the problem, a diagnostic procedure is used to collect the problem data needed for reporting the problem to IBM. IBM will debug the problem and provide a fix.

If an installation wishes to debug the problem and has the source code, a diagnostic procedure is used to collect problem data. The installation's diagnostician can use this data to debug the problem. If the problem is in IBM code, the diagnostician should report the problem to IBM.

To perform problem determination in a z/OS system address space, it may be necessary to search problem databases. It may also be necessary to report the problem to the IBM support center.

1.1 Problem resolution steps

A system problem can be described as any problem on your system that causes work to be stopped or degraded. The steps involved in diagnosing these problems are different for each type of problem. The steps needed to investigate and resolve problems are described in this section.

1.1.1 Identify the problem

Before you can begin to diagnose a system problem, you have to know what kind of problem you have. Problem identification is often not a straight-forward process, but an investigative exercise that requires a structured method that will enable the correct initial assessment to be made. This initial phase is important because decisions you make now relating to diagnostic data collection will influence the speed of the resolution.

The most important questions you must ask include:

- Is the process that is causing the problem a new procedure, or has it worked successfully before?
- If it was an existing procedure that was previously successful, *what has changed*?
- What messages are being generated that could indicate what the problem is? These could be presented on the terminal if the process is conversational, or in the batch or subsystem job log, or in the system log (SYSLOG).

  Note: Review the *z/OS MVS System Messages*, SA22-763x and *z/OS MVS Systems Codes*, SA22-7626 manuals for more details.

- Can the failure be reproduced, and if so, what steps are being performed?
- Has the failing process generated a dump?

All of these questions will enable you to develop an appropriate plan of action to aid with the resolution. You can never be criticized for providing too much diagnostic data, but too little information only delays the solving or escalation of the problem.

1.1.2 Document the problem

Documentation of the problem and the analysis steps taken can assist with not only initial resolution, but will also assist if the problem occurs again. For larger and more complex problems regular documentation during the analysis process can highlight areas that will become more crucial as the investigation progresses. This will enable you to develop a flow chart and reference point listing that can be referred to throughout your analysis. Document the final resolution for future reference.

1.1.3 Prioritize problem resolution

Your prime objective as a system programmer is to ensure system availability, and in the event of a major subsystem failure, for example, a Customer Information Control System (CICS) failure, or worse still the whole z/OS system, your focus will be on the speedy restoration of the system.

Subsystem failures often generate their own diagnostic data, and the recovery process is often fairly straightforward. These systems generally perform cleanup processes during

recovery and system availability is resumed. If the subsystem fails during the recovery, immediate problem analysis and resolution is required.

The worst-case scenario is that your complete z/OS system is down. Swift system recovery is required, but a decision must be made to determine whether the currently preserved central storage should be dumped via a stand-alone dump routine prior to the recovery Initial Program Load (IPL). The IPL process clears central storage; therefore, any failure information will be lost. The stand-alone dump process will take some time, but could be extremely valuable should the problem re-occur.

1.1.4 Analyze the problem

Before you start the process of complex analysis procedures, review all of the data you currently have that may solve your problem. Have you:

1. Looked in the system log for any relevant messages or abend information?
2. Looked in the job log for any relevant messages or abend information?
3. Reviewed the meanings of any messages or codes in the relevant manuals?
4. Reviewed the system error log, SYS1.LOGREC, which contains information about hardware and software failures?

Problem analysis is, like any process, a skill that develops the more you use it. Unfortunately, problems vary in their complexity and frequency, and it is possible that tasks requiring this type of review may be infrequent in your environment. Of course the ultimate aim is to have little need to perform complex problem diagnosis. This is why a sound methodology is necessary to assist with your analysis.

Solving a problem is a combination of:

1. Your ability to understand the problem
2. The quality of the diagnostic data you can review
3. Your ability to use the diagnostic tools at your disposal

1.1.5 Ask for assistance

To ask for assistance with a problem is not a sign of failure, but an indication you are aware that another person's views could speed up the resolution. A fresh idea can often stimulate and enhance your current thought processes, and there are often alternative ways to approach a problem.

You will hopefully be aware, if you are making little progress with your diagnosis, that some assistance may be required. What you and your manager are seeking is a speedy resolution, and it is far more professional to use all the facilities and technical expertise available to assist you. The IBM Support Center is there to assist you with your problems and the diagnostic data you have collected, and the analysis steps you have already performed will be of great help to the Support Center when they review the problem.

1.1.6 Implement the resolution

Successful diagnosis of the problem will result in a number of possible resolutions:

- *User Error* - This will require the user to correct their procedure to ensure a satisfactory resolution is implemented. If the procedure is impacting other users, then it is imperative that prompt action be encouraged.

- *Software implementation error* - You must ensure that all installation procedures have been correctly executed and any required customization has been performed correctly. Until you can be sure of a successful implementation it is advisable to remove this software, or regress to a previous level of the software until more extensive testing can be done in an environment that will not impact production workloads.
- *Software product fault* - If the fault is identified as a failure in software a fix might already have been developed to solve this problem. This fix is identified as a Program Temporary Fix (PTF) and it will need to be installed into your system. If the problem is causing a major impact, it is suggested that you expedite your normal migration process and promote the fix to the problem system to hopefully stabilize that environment.

  If the problem has not been previously reported, an authorized program analysis report (APAR) will be created and a PTF will be generated.
- *Hardware fault* - This is a resolution that will be controlled by the hardware service representative, but it may require some reconfiguration tasks depending on the nature of the problem. Consultation with the hardware vendor's service representative will clarify the requirements.

1.1.7 Close the problem

When you have tested and implemented the problem resolution, ensure that all parties involved with this problem are informed of the closure of this issue.

It should be noted that during your career you will experience some problems that occur only once, and even with the best diagnostic data, the problems cannot be recreated or solved, by anyone. When this happens there is a point where you must accept the fact that this was, in fact, just an anomaly.

1.2 Problem severity

When you report a problem to the IBM Support Center, you will be asked what the severity of the problem is. We set severity from SEV-1 (highest severity, meaning the worst problems) to SEV-4 (lowest severity, meaning the least serious problems). It is important that you be realistic when reporting the severity of an issue so that support center staff can prioritize it properly.

Severity 1 (SEV 1)

Production system down, critical business impact, unable to use the product in a production environment, no workaround is available.

Severity 2 (SEV 2)

Serious problem that has a significant business impact; use of the product is severely limited, but no production system is continuously down. SEV-2 problems include situations where customers are forced to restart processes frequently, and performance problems that cause significant degradation of service but do not render the product totally unusable. In general, a very serious problem for which there is an unattractive but functional workaround would be SEV-2, not SEV-1.

Severity 3 (SEV 3)

Problems that cause some business impact but that can be reasonably circumvented; situations where there is a problem but the product is still usable. For example, short-lived problems or problems with components that have failed and then recovered and are back in

normal operation at the time the problem is being reported. The default severity of new problem reports should be SEV-3.

Severity 4 (SEV 4)

This severity is for minor problems that have minimal business impact. While we are all aware of the pressure that customers and management place on the speedy resolution of their problems, the correct problem severity enables all involved support teams to react and manage the problems with respect to the “real” severity of each problem. While a “customer is unhappy SEV1” is in many cases valid for business reasons, it does not preclude the fact that a customer with a “production system down SEV1” is more important.

2

What version/release am I running?

Different platforms use different commands to show you product information. Many environments now comprise combinations of different platforms, operating systems, and products that all interact with the z/OS operating system in a distributed topology.

This information is vital to ensure that during problem analysis, we know exactly what system and product level we are dealing with and what maintenance has been applied to the product or module that is failing.

The sources of this information vary from the most obvious source, the system and job logs, to far more detailed interrogation using SMP/E and dump interrogation via IPCS.

In this chapter we discuss how to locate this important information.

Tip: Do not overlook the most obvious source of release information: that which is often recorded in the console or job log messages generated during startup of the operating system or product.

2.1 Source of version and release information

In z/OS, the job log will often show release information generated during the start sequence for a product. Figure 2-1 shows an example of the CICS startup message written to the CICS job log.

```
DFHSI1500 SCSCPAA1 CICS startup is in progress for CICS Transaction Server Version 3.1.0
```

Figure 2-1 CICS startup message

The abend symptom string that is written to the master console and system log shows us relevant release and maintenance information. Figure 2-2 shows an example of a CICS abend message in the MVS™ Syslog.

```
15.12.09 STC05964  +DFHME0116 CBNZPF00
(Module:DFHMEME) CICS symptom string for message
DFHFC0002 is PIDS/565514700 LVLS/530 MS/DFHFC0002 RIDS/DFHFCDN
PTFS/UQ56477 PRCS/00000445
```

Figure 2-2 CICS Abend message in MVS Syslog

This indicates that the CICS release in this case is R530 (known as CICS/TS 1.3) and the module where the failure was detected, DFHFCDN, has PTF UQ56477 applied. The PIDS field identifies the product compid.

Figure 2-3 shows how WebSphere MQSeries for z/OS displays the release level in the MQ MSTR joblog.

```
CSQY000I +MQT1 IBM WebSphere MQ for z/OS V5.3.1
```

Figure 2-3 WebSphere MQ for z/OS version information

Figure 2-4 displays the IMS™ release information that is written to the IMS CTL joblog.

```
DFSAOE00 - IMSID  IMS VERSION  SMPLEVEL  GEN DATE  GEN TYPE  869
           IMST      610        34C       031112    MODBLK
```

Figure 2-4 IMS Version information written to the joblog

In z/OS we have SMP/E to verify product and PTF levels. SMP/E is used to manage and maintain information related to system and product installation and maintenance. With SMP/E we can interrogate what has been installed into the product libraries, but this does not necessarily reflect what has been migrated to a production environment. Therefore, do not assume that the maintenance that is supposed to have fixed a problem has actually been moved into the production data sets. SMP/E does not manage the migration of upgrades.

Figure 2-5 shows the result of a SMP/E CSI GZONE query. This displays the FMIDs (Function Modification Identifiers), or more specifically, product components, that have been received into the global zone data sets. This is the first installation level. The next is to APPLY the product/maintenance into the TARGET libraries, then finally ACCEPT the product/maintenance into the DLIB, or distribution libraries.

```
 Entry Type:  GZONE                                      Zone Name: GLOBAL
 Entry Name:  GLOBAL                                     Zone Type: GLOBAL

 Default OPTIONS: CICSOPT     Related Zone:


         -------- -------- -------- -------- -------- -------- --------
ZONES    CIC22DZ  CIC22TZ
SRELS    C150
FMIDS    DELCIPM  HBDD110  HCCV320  HCI6200  HCMZ100  HCMZ110  HCMZ200
         HCP2200  HOB5110  HOB7110  HOZ2110  H24D120  H24D130  JCCV32B
         JCI620D  JCI6201  JCI6202  JCI6203  JCMZ111  JCMZ130  JCMZ201
         JCMZ230  JCP2202
```

Figure 2-5 SMP/E SMPCSI query for the GLOBAL zone

The SMP/E Cross-Zone Query lets you interrogate the maintenance level of a specific module or load module. Figure 2-6 shows an example of a cross-zone query request against the DFHSMGF module. This shows us that in the target library this module has an RMID level of UQ68396, which means that a PTF (UQ68396) have been applied to this module.

```
Entry Type:  MOD
Entry Name:  DFHSMGF

To return to the previous panel, enter END .

To select an entry from a zone, enter S next to the zone.

     * - Entry not found in zone.
    ** - Zone could not be allocated or is not initialized.

           -------------------- Status -----------------------------------
   ZONE     FMID     RMID     LASTUPD  DISTLIB  UMID(S)
   -------- -------- -------- -------- -------- -------- -------- --------
   CIC22DZ  HCI6200  HCI6200  HCI6200  ADFHMOD
   CIC22TZ  HCI6200  UQ68396  HCI6200  ADFHMOD
   GLOBAL   *
```

Figure 2-6 SMP/E Cross-Zone Query for a MODule

Note: What is reflected in the SMP/E environment does not necessarily reflect what is running in your problem system environment. It shows what maintenance has been received, applied, and accepted, but it does not show what libraries or data sets have been migrated to higher level systems.

IPCS, the Interactive Problem Control System, which we discuss later, can also be used to verify the operating system or product release, as well as abend symptom data as follows: Using **IPCS**, we can format the Communication Vector Table (**CVT**) to determine the release of z/OS that is running. The IPCS command that can be used is the **CBFORMAT** command, which means Control Block Format, and is usually abbreviated as CBF. Figure 2-7 shows the result of an IPCS CVT format.

```
CBF CVT
CVT: 00FCD2C8
-0028 PRODN... SP7.0.2 PRODI... HBB7705 VERID... MDL..... 2064
```

Figure 2-7 IPCS Communication Vector Table format

This is the first line of the formatted CVT control block and this tells us that we are running z/OS V1R2, as indicated by the PRODN value, SP7.0.2 and the FMID for this release of z/OS is HBB7705, as indicated in the PRODI field. The MDL field indicates that this version of z/OS is running on a 2064 processor.

In CICS**,** if we format the dump using `IPCS VERBX 'CSA=2'` we can review the data at offset **x'9F'** which displays the CICS release level, for example, **53** or **62**.

We can also interrogate the maintenance that has been applied to modules using IPCS as follows:

- In CICS, for example, issue the IPCS command `VERBX DFHPD530 'LD=1'` and locate the PROGRAM STORAGE MAP. Figure 2-8 shows an example of an IPCS format of the CICS Loader domain.

```
DFHCSA   8004CE20 DFHKELCL 0004C000 530  ESA530   02/20/99 18.36
                  DFHKELRT 0004C380 530  ESA530   02/20/99 18.36
                  DFHCSAOF 0004C600 0530 UQ43786  I  01/06 13.31
                  DFHCSA   0004CBD8 0530 UQ43786  I  01/06 13.31
                  DFHKESFM 0004D0C8 530  UQ39652  01/27/00 15.20
                  DFHKESGM 0004D4A0 530  UQ39652  01/27/00 15.20
                  DFHKERCD 0004DCC8 530  ESA530   02/20/99 18.36
                  DFHKERER 0004DEA0 530  ESA530   02/20/99 18.36
```

Figure 2-8 CICS IPCS format of the Loader Domain

- In **DB2** you can run the `DIAGNOSE DISPLAY MEPL` utility to format the module information. Figure 2-9 shows an example of the DB2 Diagnose Display MEPL process.

```
*....DSNAA   10/22/9811.44   ....*
*....DSNAPRH 07/10/9813.28   ....*
*....DSNFMNFM07/10/9814.38   ....*
*... DSNFPMSG07/10/9814.42   ....*
*....DSNFSAMG07/10/9814.42   ....*
*....DSNUBBCD09/30/9814.29   ....*
*....DSNUBBCM06/11/02UQ66957 ....*
*....DSNUBBCR08/20/02UQ69047 ....*
*... DSNUBBID08/29/02UQ69311 ....*
*....DSNUBBOP12/02/01UQ60569 ....*
*....DSNUBBRD04/27/99UQ29552 ....*
*....DSNUBBUM01/17/02UQ61891 ....*
```

Figure 2-9 DB2 Diagnose Display MEPL output

3

Fundamental sources of diagnostic data

Often, the most readily available source of data identifies the key piece of information that will resolve the problem, and often, this source of data is overlooked. The *console log*, *system log*, or *error log*, related to a specific product or the whole system, is the first place to look when reviewing a problem.

While a system dump or a trace is often required, the logs provide enough detail, in many cases, to solve the problem. The location of the relevant logs varies from product to product, and system to system.

3.1 Diagnostic data sources

The main sources of diagnostic data are contained in the messages provided by the system in the following logs:

- Console log

  Messages sent to a console with master authority are intended for the operators. The system writes in the hard-copy log all messages sent to a console, regardless of whether the message is displayed.

- SYSLOG

  The SYSLOG is a SYSOUT data set provided by the job entry subsystem (either JES2 or JES3). SYSOUT data sets are output spool data sets on direct access storage devices (DASD). An installation should print the SYSLOG periodically to check for problems. The SYSLOG consists of the following:

  - All messages issued through WTL macros
  - All messages entered by LOG operator commands
  - Usually, the hard-copy log
  - Any messages routed to the SYSLOG from any system component or program

- Job log

  Messages sent to the job log are intended for the programmer who submitted a job. Specify the system output class for the job log in the MSGCLASS parameter of the JCL JOB statement.

- OPERLOG

  Operations log (OPERLOG) is an MVS system logger application that records and merges messages about programs and system functions (the hardcopy message set) from each system in a sysplex that activates OPERLOG. Use OPERLOG rather than the system log as your hardcopy medium when you need a permanent log about operating conditions and maintenance for all systems in a sysplex.

- Hard-copy log

  The hard-copy log is a record of all system message traffic:

  - Messages to and from all consoles
  - Commands and replies entered by the operator

  In a dump, these messages appear in the master trace. With JES3, the hard-copy log is always written to the SYSLOG. With JES2, the hard-copy log is usually written to the SYSLOG but can be written to a console printer, if the installation chooses.

- Logrec

  Logrec log stream is an MVS System Logger application that records hardware failures, selected software errors, and selected system conditions across the sysplex. Using a logrec log stream rather than a logrec data set for each system can streamline logrec error recording.

3.2 SYSLOG

On z/OS, the SYSLOG can be viewed via the Spool Display and Search Facility (SDSF) using the LOG option. A small amount of the SYSLOG is also stored in memory and is included when an address space is dumped. This is referred to as master trace (MTRACE) data and can be accessed via the IPCS using the `VERBX MTRACE` command.

Figure 3-1 shows an example of the MVS SYSLOG. The timestamps that would normally be seen to the left of the data have been removed for presentation, but are valuable when comparing problem data from different sources.

```
STC18213 00000090  $HASP100 BPXAS    ON STCINRDR
STC18213 00000090  $HASP373 BPXAS    STARTED
STC18213 80000010  IEF403I BPXAS - STARTED - TIME=13.36.36 - ASID=001F - SC53
STC16316 00000291  IST663I IPS SRQ REQUEST FROM ISTAPNCP FAILED, SENSE=08570002
     111 00000291  IST664I REAL  OLU=USIBMSC.S52TOS48    REAL  DLU=USIBMSC.S48TO
     111 00000291  IST889I SID = ED0385CAAEEAAF28
     111 00000291  IST264I REQUIRED RESOURCE S48TOS52 NOT ACTIVE
     111 00000291  IST314I END
STC16352 00000291  IST663I IPS SRQ REQUEST FROM ISTAPNCP FAILED, SENSE=087D0001
     883 00000291  IST664I REAL  OLU=USIBMSC.S52TOS48    ALIAS DLU=USIBMSC.S48TO
     883 00000291  IST889I SID = ED0385CAAEEAAF28
     883 00000291  IST314I END
STC28215 00000291  IST663I IPS SRQ REQUEST TO ISTAPNCP FAILED, SENSE=08570002 86
     864 00000291  IST664I REAL  OLU=USIBMSC.S52TOS48    ALIAS DLU=USIBMSC.S48TO
     864 00000291  IST889I SID = ED0385CAAEEAAF28
     864 00000291  IST264I REQUIRED RESOURCE S48TOS52 NOT ACTIVE
     864 00000291  IST891I USIBMSC.SC48M GENERATED FAILURE NOTIFICATION
     864 00000291  IST314I END
```

Figure 3-1 Sample MVS SYSLOG data

3.3 OPERLOG

In SDSF the OPERLOG panel displays the merged, sysplex-wide system message log. Parameters of the LOG command allow users to choose the OPERLOG panel or the single-system SYSLOG panel.

The OPERLOG panel displays the data from a log stream, a collection of log data used by the MVS System Logger to provide the merged, sysplex-wide log.

Each individual product has its own log file on the z/OS platform. These log files may contain data that may be valuable when diagnosing a problem. It is particularly important to look for events that precede an actual abend or failure because the problem, in many cases, will have been caused by a previous action. Figure 3-2 shows the SYSOUT data sets that might be associated with a CICS address space.

```
NP   DDNAME   StepName ProcStep DSID Owner
     JESJCLIN                    1 CICSTS
     JESMSGLG JES2               2 CICSTS
     JESJCL   JES2               3 CICSTS
     JESYSMSG JES2               4 CICSTS
     $INTTEXT JES2               5 CICSTS
     CAFF     SCSCPAA1         101 CICSTS
     CINT     SCSCPAA1         103 CICSTS
     DFHCXRF  SCSCPAA1         104 CICSTS
     COUT     SCSCPAA1         105 CICSTS
     CEEMSG   SCSCPAA1         106 CICSTS
     CEEOUT   SCSCPAA1         107 CICSTS
     PLIMSG   SCSCPAA1         108 CICSTS
     CRPO     SCSCPAA1         109 CICSTS
     MSGUSR   SCSCPAA1         110 CICSTS
```

Figure 3-2 Display of CICS SYSOUT data sets obtained with the SDSF DA operand

The key SYSOUT data sets to review that may provide problem determination data are the JESMSGLG and MSGUSR data sets. The CEEMSG and CEEOUT data sets will contain Language Environment (LE) problem data usually associated with application problems.

Figure 3-3 shows an example of some transaction abend data included in the MSGUSR SYSOUT data set.

```
DFHIR3783 04/11/2005 01:25:50 SCSCPTA2 Transaction SX2 termid E39 -
Connected transaction abended with message DFHAC2206 01:25:50 SCSCPAA4 Transaction SX2
failed with abend AFCV. Updates to local recoverable resources backed out.
DFHAC2236 04/11/2005 01:25:50 SCSCPTA2 Transaction SX2 abend AZI6 in program *UNKNOWN
term PB09. Updates to local recoverable resources will be backed out.
DFHAC2262 04/11/2005 01:25 (sense code 0824089E).
DFHAC2206 01:25:50 SCSCPAA4 Transaction SX2 failed with abend AFCV.
Updates to local recoverable resources backed out.
```

Figure 3-3 CICS MSGUSR SYSOUT data set sample data

The CICS JESMSGLG SYSOUT data set includes information related to CICS startup and errors related to system problems, not specifically transaction related. Figure 3-4 is a sample taken from the CICS JES Message Log (JESMSGLG).

```
+DFHTR0103 TRACE TABLE SIZE IS 64K
+DFHSM0122I SCSCPTA2 Limit of DSA storage below 16MB is 5,120K.
+DFHSM0123I SCSCPTA2 Limit of DSA storage above 16MB is 60M.
+DFHSM0113I SCSCPTA2 Storage protection is not active.
+DFHSM0126I SCSCPTA2 Transaction isolation is not active.
+DFHSM0120I SCSCPTA2 Reentrant programs will not be loaded into read-only storage
+DFHDM0101I SCSCPTA2 CICS is initializing.
+DFHXS1100I SCSCPTA2 Security initialization has started.
+DFHWB0109I SCSCPTA2 Web domain initialization has started.
+DFHSO0100I SCSCPTA2 Sockets domain initialization has started.
+DFHRX0100I SCSCPTA2 RX domain initialization has started.
+DFHRX0101I SCSCPTA2 RX domain initialization has ended.
+DFHLG0101I SCSCPTA2 Log manager domain initialization has started.
+DFHEJ0101 SCSCPTA2  291
 Enterprise Java domain initialization has started. Java is a
 trademark of Sun Microsystems, Inc.
+DFHDH0100I SCSCPTA2 Document domain initialization has started.
.
+DFHLG0103I SCSCPTA2 System log (DFHLOG) initialization has started.
+DFHLG0104I SCSCPTA2  340
 System log (DFHLOG) initialization has ended. Log stream
 ************************** is connected to structure
 ****************.
+DFHSI1519I SCSCPTA2 The interregion communication session was successfully started
+DFHWB1007 SCSCPTA2 Initializing CICS Web environment.
+DFHWB1008 SCSCPTA2 CICS Web environment initialization is complete.
+DFHSI8430I SCSCPTA2 About to link to PLT programs during the third stage of
initialization
+EYUNX0001I SCSCPTA2 LMAS PLTPI program starting
+EYUXL0003I SCSCPTA2 CPSM Version 220  LMAS startup in progress
+EYUXL0103E SCSCPTA2 CICSPlex SM subsystem (EYUX) not active
+EYUXL0024I SCSCPTA2 Waiting for CICSPlex SM subsystem activation
```

Figure 3-4 CICS JESMSGLG output

3.4 Logrec

The z/OS error log contains data related to hardware and software errors. This data is written to the SYS1.LOGREC data set and is also written to internal storage that is included in a dump. The SYS1.LOGREC data set can be interrogated using the ICFEREP1 program, or if the abend has triggered a dump, the EREP data can be reviewed using the IPCS VERBX LOGDATA command.

Figure 3-5 on page 15 shows the last error record contained in the error log generated when the VERBX LOGDATA command was issued for a dump being reviewed using IPCS. Generally, the error log entries at the end of the display, if they have an influence on the problem being reviewed, will have time stamps that relate to (or immediately precede) the actual abend.

```
JOBNAME: ITSOCIOI   SYSTEM NAME: SC48
ERRORID: SEQ=05462  CPU=0042  ASID=00CE  TIME=15:03:28.1

SEARCH ARGUMENT ABSTRACT

  PIDS/5740XYR00 RIDS/DSNXGRDS#L RIDS/DSNXRIVB AB/S00C7 PRCS/00000000 REGS/0CB2C
  REGS/B6B67 RIDS/DSNTFRCV#R

  SYMPTOM             DESCRIPTION
  -------             -----------
  PIDS/5740XYR00      PROGRAM ID: 5740XYR00
  RIDS/DSNXGRDS#L     LOAD MODULE NAME: DSNXGRDS
  RIDS/DSNXRIVB       CSECT NAME: DSNXRIVB
  AB/S00C7            SYSTEM ABEND CODE: 00C7
  PRCS/00000000       ABEND REASON CODE: 00000000
  REGS/0CB2C          REGISTER/PSW DIFFERENCE FOR R0C: B2C
  REGS/B6B67          REGISTER/PSW DIFFERENCE FOR R0B:-6B67
  RIDS/DSNTFRCV#R     RECOVERY ROUTINE CSECT NAME: DSNTFRCV

OTHER SERVICEABILITY INFORMATION

  DATE ASSEMBLED:          01/29/04
   MODULE LEVEL:            UQ84577
   SUBFUNCTION:             RDS  SQL     DIAGNOSE

 SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE

   RECOVERY ROUTINE LABEL

 TIME OF ERROR INFORMATION

   PSW: 077C1000 9E43EDFC   INSTRUCTION LENGTH: 04   INTERRUPT CODE: 0007
   FAILING INSTRUCTION TEXT: D5244420 B0219680 D5245820
```

Figure 3-5 Final record in logrec data from IPCS VERBX LOGDATA

Tip: Do not ignore the valuable data that is written to the log files.

4

Common problem types

The term that you will most often hear in relation to system or application problems is *abend*, which stands for *abnormal end*. In this chapter we discuss the different types of abends, some key factors that can affect system and application performance, and some of the tools that can assist with determining what is occurring at a given point in time in the system.

The system can enter a wait or the entire system can hang. The terms *hang* and *wait* are used interchangeably in this discussion. Some symptoms of a wait/hang are:

- No response on user's or system operator's console.
- The operator cannot communicate with the system through the console with master authority or the alternate console.
- The system does not issue messages to the console with master authority or the alternate console. The system does not receive messages from these consoles.
- The operator witnesses a series of WAIT indicators followed by a burst of activity.
- A message indicating a wait appears on the system console.

Other problems can be the result of the following:

- Application program abends
- System program abends

Various problems can be analyzed using traces, dumps, and monitors.

4.1 Application program abends

Application program abends are always accompanied by messages in the system log (SYSLOG) and the job log, indicating the abend code and, usually, a reason code. Many abends also generate a symptom dump in the SYSLOG and job log. A symptom dump is a system message, either message IEA995I or a numberless message, which provides some basic diagnostic information for diagnosing an abend. Often the symptom dump information can provide enough information to diagnose a problem.

Figure 4-1 shows the symptom dump for an abend X' 0C4' with reason code X'4'. This symptom dump shows that:

- Active load module ABENDER is located at address X'00006FD8'.
- The failing instruction was at offset X'12' in load module ABENDER.
- The address space identifier (ASID) for the failing task was X' 000C'.

```
IEA995I SYMPTOM DUMP OUTPUT
  SYSTEM COMPLETION CODE=0C4  REASON CODE=00000004
   TIME=16.44.42  SEQ=00057  CPU=0000  ASID=000C
   PSW AT TIME OF ERROR  078D0000   00006FEA  ILC 4  INTC 04
     ACTIVE LOAD MODULE=ABENDER   ADDRESS=00006FD8  OFFSET=00000012
     DATA AT PSW  00006FE4 - 00105020  30381FFF  58E0D00C
     GPR  0-3  FD000008  00005FF8  00000014  00FD6A40
     GPR  4-7  00AEC980  00AFF030  00AC4FF8  FD000000
     GPR  8-11 00AFF1B0  80AD2050  00000000  00AFF030
     GPR 12-15 40006FDE  00005FB0  80FD6A90  00006FD8
   END OF SYMPTOM DUMP
```

Figure 4-1 SYMPTOM dump data as shown in the MVS SYSLOG and related Job log

If the information in a symptom dump is insufficient you can capture additional dump data by including specific DD statements as discussed later in this chapter.

4.2 System program abends

Like application program abends, system program abends are usually accompanied by messages in the system log (SYSLOG). In addition, if there is a SYS1.DUMPxx data set available at the time of the abend, and if this dump code was not suppressed by the dump analysis and elimination (DAE) facility, then an SVC dump will be taken. SVC dumps are discussed later in this chapter.

4.3 I/O errors

I/O errors are most likely caused by a hardware failure or malfunction, and the visible symptom is an abend, accompanied by messages in the SYSLOG that include reason codes, which can identify the type of error; and sense data, which indicates more detailed, hardware-specific information.

I/O errors can also be the result of software conditions that create a situation where subsequent operations will appear as I/O errors. This could be the result of a corruption in a data set, or data set directory, and the rectification process may be as simple as redefining the data set.

Genuine hardware errors generally need to be reviewed by the hardware service representative, and the diagnostic data required to analyze these problems is recorded in the system error log data set, SYS.LOGREC. IBM provides the EREP facility to enable diagnostic information to be extracted from the SYS1.LOGREC data set.

4.4 System wait states

The basic summation of a wait state is “the machine is dead, or will not IPL.” You usually experience this condition during the IPL process, and the disabled wait state code indicates the problem. The cause is often as simple as the system not being able to find some data that is crucial to the IPL process on the IPL volume. Wait codes are documented in *z/OS MVS Systems Codes*, GC28-1780.

4.5 System, subsystem, and application hangs

Hangs are usually caused by a task, or tasks, waiting for either an event that will never happen, or an event that is taking an excessive amount of time to occur. If one of the waiting tasks is a fundamental system task, or is holding control of a resource (for example, a data set), then other tasks will queue up and wait for the required resource to become available. As more tasks enter the system they will also join the queue until the system eventually stops, or the task causing the contention is cancelled. Unfortunately, by the time the system grinds to a halt, the operating system will no longer process any operator commands, so an IPL will be the only alternative. A system hang is more specifically known as an *enabled wait* state.

4.6 Hangs and loops

One of the difficult things to determine is whether a system or subsystem is in a *hung* or *looping* state. While the symptoms in many cases are similar — for example, the inability to process other units of work, or transactions, or the inability to get the system or subsystem to accept commands — the key difference is whether there is CPU and EXCP activity that indicates the system is still performing work.

If no other tasks can be dispatched within a subsystem, and the CPU activity is high, often 100 percent, this is generally a symptom that we have a loop condition. Loops can usually be categorized as either *enabled*, *disabled* or *spin* loops.

Loops are caused by a program, application, or subsystem attempting to execute the same instructions over and over again. The most severe loop condition causes the task experiencing the condition to use all available CPU resources, and subsequently no other task is allowed to gain control. The only way to alleviate the problem is to cancel the problem task, or if this is unsuccessful an IPL will be required. The three types of loop conditions are:

Enabled Enabled loops are usually caused by a programming error, but they do not impact other jobs in the system unless the looping task is a subsystem, which generally impacts the whole system.

Disabled Disabled loops do not allow an interrupt to be processed, and are generally identified by continuous 100 percent CPU utilization.

Spin Spin loops occur when one processor in a multiple-processor environment is unable to communicate with another processor, or is unable to access a resource being held by another processor.

A CPU entering a disabled loop will often be presented to the operators as a SPIN loop, where the system will cycle (or SPIN) through the available CPUs.

There are many tools that can be used to assist with hang/loop problem diagnosis, and many of the system monitoring tools enable you to interrogate at the transaction/thread level, and enable you to cancel/purge the individual unit of work or task associated with the loop.

It is important to remember that the monitoring tools should have a high dispatching priority to enable them to get control when required.

Trace data can be used to assist with loop and hang diagnosis, and even 20 seconds of trace data can help identify a looping sequence and often the associated unit of work or transaction. For example, the CICS Auxtrace facility or CICS internal tracing with all CICS components traced at level 1 and a dump of the suspected problem regions can show, via a quick IPCS review, the type of problem you are experiencing.

4.7 SYSTRACE, RMFMON, and SDSF

The trace and dump process should be used as a dynamic tool to gather information at the time a problem is occurring.

Comparing the dump time with the time of the last trace entry for a specific ASID can indicate a hang condition if the ASID is not generating any trace entries.

Figure 4-2 shows an example of using IPCS to format the System Trace Table by issuing the `SYSTRACE` command from the IPCS Subcommand Entry panel. This displays the trace data for ASID that was dumped.

```
00 00A1 007C1588   SSRV   78          8134CE6E  0000FD03 00001E00 007AE200
                                                00A10000
00 00A1 007C1588   SSRV   78          8134A9CA  0000FF12 00000010 007BC0F0
                                                00A10000
00 00A1 007C1588   SVCR   7A 078D1000 800399BE  00000000 B1DEB000 00000800
00 00A1 007C1588   PC    ...   8      009D7D51              0C70C
00 00A1 007C1588   PR    ...   0      009D7D51 009D9B42
00 00A1 007C1588   PC    ...   8      009D7D51              0C70C
00 00A1 007C1588   PR    ...   0      009D7D51 009D9B42
00 00A1 007C1588  *SVC     D 078D1000 8002D112  00000090 84000000 84000FFD
```

Figure 4-2 Primary ASID System Trace Table data

4.7.1 Displaying trace data for all ASIDs

Figure 4-3 displays the system trace data for all ASIDs as the result of issuing the `SYSTRACE ALL TIME(LOCAL)` command.

The TIME(LOCAL) option formats the trace timestamp data from internal formatted data and this is displayed at the right of each trace entry, for example `15:03:17.020192`. Without TIME(LOCAL) the field looks like: `BCD014BF7F621603`**.** This format is not very helpful if you are trying to match up corresponding syslog or joblog event data.

SDSF and RMFMON can also assist with problem determination to show CPU and EXCP usage for each ASID. The SDSF DA display, and also the `MVS "D A,jobname"` command can be used to assist with these types of problems.

```
00 0006 03F29A60   SRB       070C0000 81514F00  00000000 01F2AFA0 01F2AFD0
                                                00000000 20
00 00A1 007E2E88   DSP       070C0000 8176D230  00800000 80000000 03823000  00
01 0015 040D4DC0   SRB       070C0000 815B34A8  00000015 01D709D4 01D70A00
                                                007E9CF0 20
01 0015 040D4DC0   PC    ...   0      060FE6C0              00331
01 0015 040D4DC0   PR    ...   0      060FE6C0 011DEEF2
01 000B 03E7C040   SRB       070C0000 860EB7A8  0000000B 01D78C0C 21E1EA48
                                                007F8360 A0
01 000B 03E7C040   PC    ...   0      060ECB00              00312
01 000B 03E7C040   PC    ...   0      014322E2              0030A
01 000B 03E7C040   PR    ...   0      014322E2 0140894C
01 000B 03E7C040   PR    ...   0      060ECB00 014325BA
01 000B 03E7C040   PC    ...   0      060ECB6E              0000D
```

Figure 4-3 System trace data for ALL ASIDS.

SDSF panel display by jobname

Figure 4-4 shows an example of the SDSF DA display, where a filter has been used via the PREFIX jobname option, `PREFIX DB7P*` in this case.

```
SDSF DA SC48  (ALL)    PAG    0 SIO   344 CPU   9/  9  LINE 1-5 (5)
COMMAND INPUT ===>                                      SCROLL =
NP   JOBNAME   CPU% ASID ASIDX  EXCP-Cnt   CPU-Time SR Status SysName
     DB7PSPAS  0.00  143  008F       204       6.00           SC47
     DB7PDIST  0.00 1002  03EA       686       4.96           SC47
     DB7PDBM1  0.00 1003  03EB     7,816      13.27           SC47
     DB7PIRLM  0.00 1004  03EC       137     117.89           SC47
     DB7PMSTR  0.00 1005  03ED     5,131     106.95           SC47
```

Figure 4-4 SDSF DA display using a PREFIX DB7P filter*

This is a dynamic display that is updated with each refresh (ENTER) action.

The SDSF DA options `SORT CPU% D` and `SORT EXCP-cnt D` can make finding the problem ASID easier. The DA display will be shown in descending order with the highest CPU% user or the highest EXCP user at the top of the display.

4.7.2 RMF Monitor II

The RMF™ Monitor II can be started by issuing the `TSO RMFMON` command. Figure 4-5 on page 22 shows the RMF DISPLAY MENU. Monitor II is a snapshot reporting tool for very fast information about how specific address spaces or system resources (processor, DASD volumes, storage) are performing. Monitor II has two modes for reporting on the performance of your system.

```
RMF DISPLAY MENU
NAME      PFK#  DESCRIPTION

ARD        1    ADDRESS SPACE RESOURCE DATA
ASD        2    ADDRESS SPACE STATE DATA
ASRM       3    ADDRESS SPACE SRM DATA
CHANNEL    4    CHANNEL PATH ACTIVITY
DDMN       5    -----NOT APPLICABLE IN GOAL MODE-----
DEV        6    DEVICE ACTIVITY
PGSP       7    PAGE/SWAP DATA SET ACTIVITY
SENQ       8    SYSTEM ENQUEUE CONTENTION
SENQR      9    SYSTEM ENQUEUE RESERVE
SPAG       10   PAGING ACTIVITY
SRCS       11   CENTRAL STORAGE / PROCESSOR / SRM
TRX        12   -----NOT APPLICABLE IN GOAL MODE-----
ARDJ            ADDRESS SPACE RESOURCE DATA
ASDJ            ADDRESS SPACE STATE DATA
ASRMJ           ADDRESS SPACE SRM DATA
DEVV            DEVICE ACTIVITY
IOQUEUE         I/O QUEUING ACTIVITY
SDS             RMF sysplex DATA SERVER
LLI             PROGRAM LIBRARY INFORMATION
ILOCK           IRLM LONG LOCK DETECTION
```

Figure 4-5 TSO RMFMON option menu

ARD report

In the ARD report, the number of data lines in the report depends on the number of address space identifiers in the system that meet your selection criteria. The shown report is a sample for a system running in z/Architecture™. Figure 4-6 shows the result of issuing the `ARD` command that will display data for each ASID. The key information we are looking for is who is consuming the CPU or EXCP cycles.

```
20:27:52 DEV    FF   FF PRIV LSQA X C SRM  TCB   CPU  EXCP
JOBNAME  CONN  16M   2G   FF  CSF M R ABS  TIME  TIME RATE
SMF      21.39   0   56    0   75 X   0.0  0.19  0.76 0.00
LLA      35.25   0  129   50  106 X   0.0  6.49  6.87 0.00
JES2AUX  0.015   0   35   13   28     0.0  0.00  0.00 0.00
JES2     399.9  10  294  111  237     0.0 81.36 87.32 4.97
VTAM44   6.851   0  106   31  151 X   0.0 28.27 55.82 0.00
JES2MON  0.000   0   42    0   64     0.0 13.49 22.28 0.00
DFSMSHSM 14.47   0   77    3  121     0.0  2.85  3.04 0.00
DB4BDBM1 4.132   0  453  200  318 X   0.0  0.57  1.07 0.00
DB4BMSTR 11.85   0  130    2  177 X   0.0 21.83 24.58 0.47
```

Figure 4-6 RMFMON address space resource data display

ARDJ report

In the ARDJ report, the number of rows depends on your requests to build a table of information for a particular job.

You can then issue the `ARDJ` command for a specific jobname you are interested in reviewing. Figure 4-7 shows the result of the `ARDJ DB4BMSTR` command.

```
DB4BMSTR DEV    FF   FF PRIV LSQA X C SRM  TCB   CPU  EXCP
  TIME   CONN  16M   2G   FF  CSF M R ABS  TIME  TIME RATE
20:34:01 12.06   0  130    2  177 X   0.0 22.29 25.09 ----
```

Figure 4-7 RMFMON jobname specified address space resource data (ARDJ jobname)

Figure 4-8 shows the result of pressing Enter at intervals. One line is added to the display for each Enter request.

In this case we have no loop condition, but as you can see, the CPU time and EXCP count are increasing with each update request. This example was from a system with very little load.

```
DB4BMSTR DEV    FF   FF PRIV LSQA X C SRM  TCB   CPU  EXCP
  TIME   CONN  16M   2G   FF  CSF M R ABS  TIME  TIME RATE
20:34:01 12.06   0  130    2  177 X   0.0 22.29 25.09 ----
20:34:02 12.06   0  130    2  177 X   0.0 22.29 25.09 2.00
20:34:03 12.06   0  130    2  177 X   0.0 22.29 25.09 0.00
20:34:04 12.06   0  130    2  177 X   0.0 22.29 25.09 0.00
20:34:05 12.06   0  130    2  177 X   0.0 22.30 25.09 0.00
20:34:06 12.06   0  130    2  177 X   0.0 22.30 25.09 2.00
20:34:13 12.07   0  130    2  177 X   0.0 22.30 25.10 0.29
20:34:15 12.07   0  130    2  177 X   0.0 22.30 25.10 1.00
20:34:15 12.07   0  130    2  177 X   0.0 22.30 25.10 ----
20:35:48 12.12   0  130    2  177 X   0.0 22.44 25.25 0.49
```

Figure 4-8 Incremental Address Space Resource data

4.7.3 GRS contention

The `DISPLAY GRS,CONTENTION` (or `D GRS,C`) is excellent to determine if the “wait” is related to a resource contention problem. Figure 4-9 shows the display returned where no contention is detected.

```
RESPONSE=SC64
 ISG343I 23.04.43 GRS STATUS 824
 NO ENQ RESOURCE CONTENTION EXISTS
 NO REQUESTS PENDING FOR ISGLOCK STRUCTURE
 NO LATCH CONTENTION EXISTS
```

Figure 4-9 Display GRS Contention output

This command is excellent for assisting with the resolution of EXCLUSIVE ENQUEUE problems when one ASID has and EXCLUSIVE enqueue on a data set that another ASID is trying to access. This would appear as a hang type condition for the waiting ASID.

The *owning* ASID and the *waiting* ASID, plus the resource that is causing the enqueue wait, will be displayed by the `D GRS,C` command.

It is important to remember that simple commands, such as issuing a **`D T`** command on all systems in the sysplex to display the time, can assist with determining what system is causing the problem. It is not necessarily the system that is experiencing the problem that is the cause of the problem. This system could just be a victim of a problem in another system.

Ensure that all outstanding WTOR messages have been responded to if they relate to the problem system or subsystem. Issue the **`D R,R`** command and check for messages that need to be actioned.

Note: If you display the PSW at the maintenance console at intervals of a few seconds and it does not change then the system is hung.

4.8 Program errors

Program errors (error messages, bad return codes, incorrect processing) require different diagnostic procedures to assist with the problem determination. Application-program-related errors are best solved in consultation with the application development team. A combination of a system dump as well as the source code, and compile and link-edit listings can assist with the diagnosis.

Some of the key program *exception* abend codes you may see include:

- **0C1 - OPERATION EXCEPTION**: An OP CODE is not assigned or the assigned operation is not available.
- **0C2 - PRIVILEGED-OPERATION EXCEPTION**: You cannot use the OP CODE specified; it is a privileged instruction.
- **0C4 - PROTECTION EXCEPTION**: A virtual address could not be translated into a real address. You tried to access a storage location that is not available to your program.
  - **0C4 (Rsn Code=4) - Protection exception**. The key of the storage area that the running program tries to access is different from that of the running program.
  - **0C4 (Rsn Code=10) - Segment-translation exception.** A program that was running disabled attempted to reference storage while the page table for that storage was paged out.
  - **0C4 (Rsn Code=11) - Page-translation exception**. A program that was running disabled attempted to reference storage while that storage was paged out.
- **0C7 - DATA EXCEPTION**: The sign or digit codes of operands in decimal arithmetic, editing operations, or in convert to binary are incorrect, or fields in decimal arithmetic overlap incorrectly.

5

MVS messages and codes

The MVS operating system issues messages from the base control program components, the job entry subsystems (JES2 and JES3), the Data Facility Product (DFP), system products, and application programs running under the system. The system issues messages in different ways and to different locations:

- Most messages are issued through WTO and WTOR macros to one of the following locations:
  - Console
  - Hard-copy log
  - Job log
  - SYSOUT data set
- Other messages are issued through the WTL macro or the **LOG** operator command to the system log (SYSLOG).
- Dump messages are issued through the dumping services routines and can appear in:
  - SVC dumps, stand-alone dumps, or SYSMDUMP ABEND dumps formatted by the interactive problem control system (IPCS)
  - Trace data sets formatted by the interactive problem control system (IPCS)
  - ABEND dumps or SNAP dumps produced by the dumping services

  In dump or trace data sets formatted by IPCS, the messages appear interactively on a terminal or in a printed dump.
- Some messages are issued through the Data Facility Product (DFP) access methods directly to one of the following locations:
  - Output data set
  - Display terminal

5.1 Message formats

A displayed or printed message can appear by itself or with other information, such as a time stamp. This section defines the format of the messages as well as information accompanying the messages on the MCS® console and on the hard-copy log in a JES2 system and a JES3 system.

The message formats, shown in Figure 5-1, are as follows:

- **id** is the reply identifier and is optional. It appears if an operator reply is required. The operator specifies it in the reply.
- CCCnnn, CCCnnns, CCCnnnns, CCCnnnnns, CCCSnnns are the message identifiers, where:
  - **CCC** is a prefix to identify the component, subsystem, or product that produced the message. The prefix is three characters.
  - **S** is the subcomponent identifier, which is an optional addition to the prefix to identify the subcomponent that produced the message. The subcomponent identifier is one character.
  - nnn, nnnn, nnnnn is a serial number that identifies the individual message. The serial number is three, four, or five decimal digits.
  - **s** is an optional type code which can be specified as follows:

A	**Action**	The operator must perform a specific action.
D	**Decision**	The operator must choose an alternative.
E	**Eventual action**	The operator must perform action when time is available.
I	**Information**	No operator action is required. Most information messages are for a programmer.
S	**Severe error**	Severe error messages are for a programmer.
W	**Wait**	Processing stops until the operator performs a required action.

```
id CCCnnn text
id CCCnnns text
id CCCnnnns text
id CCCnnnnns text
id CCCSnnns text
```

Figure 5-1 Message formats

5.2 Message examples

Some messages have asterisks (*) before or after the message identifier. Two asterisks after the message identifier for IDC messages indicates a second-level message that further explains a preceding message. Figure 5-2 shows some message examples.

When reporting problems to IBM, always provide accompanying messages: identifiers and texts.

```
EYUXE0001I CPSM subsystem (EYUX) initialization complete

IEC161I 056-084,SCSCCMAS,SCSCCMAS,DFHLCD,,,  595
IEC161I CICSSYSF.CICS620.CMAS.DFHLCD,
IEC161I CICSSYSF.CICS620.CMAS.DFHLCD.DATA,CATALOG.TOTICF1.VTO

DFHEJ0101 SCSCCMAS  675
Enterprise Java domain initialization has started.

IST129I UNRECOVERABLE OR FORCED ERROR ON NODE TRL0516P - VARY INACT SCHED
IST259I INOP RECEIVED FOR TRL0523P CODE = 01

DSNZ002I -D7Q2 DSNZINIT SUBSYSTEM D7Q2 SYSTEM PARAMETERS LOAD MODULE NAME IS DSNZPAQ2

DSN3201I  -D7Q2 ABNORMAL EOT IN PROGRESS FOR  305
USER=CICSTS CONNECTION-ID=SCSCPJA7 CORRELATION-ID=
JOBNAME=SCSCPJA7 ASID=008B TCB=007F6288

CSQE013I =MQZ1 Recovery phase 1 completed for  319
structure APPLICATION1 connection name CSQEMQZGMQZ101

DFS0579W - FIND FAILED FOR DDNAME PROCLIB  MEMBER = DFSRSR00 RETURN CODE=X'00000004'
```

Figure 5-2 Message examples

5.3 System codes

System codes include system completion codes (or abend codes) identified by three hexadecimal digits, and user completion codes identified by four decimal digits, and are usually the result of a system or an application program abnormally ending. The completion code indicates the reason for the abnormal end.

System codes also identify wait states. The wait state code appears in the program status word (PSW) when the operating system enters a wait state.

A valid PSW for a coded wait state in ESA (31-bit) mode has one of the following general formats:

- 000A0000 xrrrrwww
- 000A0000 xrr00www
- 000A0000 x0rrrwww
- 000A0000 xrrr0www

A description of the PSW is as follows:

- A - Bits 12-15 (the CMWP bits, with the 'C' and 'W' bits being on).
- x - Bits 32-35. Not part of the wait state information.
- rrrr, rr00, 0rrr, rrr0 - Bits 36-51, where r is the reason code for 8/12/16 bits and 0=zero.
  - It is a supplementary code accompanying the wait state code.
  - The wait state determines the size and position of the supplement code.
  - Usually the supplementary code is a reason code. Some wait state codes do not provide a supplementary code in the PSW. See the description of the individual wait state code for more information.

5.4 Wait state codes

Some wait state codes do not provide a supplementary code in the PSW. See the description of the individual wait state code for more information. In Figure 5-3, the wait state code is shown as `www` in Bits 52 to 63.

The IBM-supplied wait state codes are explained in *z/OS MVS System Codes*, SA22-7626.

For z/Architecture (64-bit) mode the wait state code still appears in the program status word (PSW) when the operating system enters a wait state.

A valid PSW for a coded wait state in z/Architecture mode has one of the general formats shown in Figure 5-3.

```
00020000 x0000000 00000000 0rrrrwww
00020000 x0000000 00000000 0rr00www
00020000 x0000000 00000000 00rrrwww
00020000 x0000000 00000000 0rrr0www
```

Figure 5-3 PSW showing the wait state codes (www)

The middle two words provide no relevant wait state information.

For additional discussion of the PSW bits, see 8.1.1, “Program status word details” on page 56.

6

SYS1.PARMLIB diagnostic parameters

This chapter describes some of the diagnostic aids that can be used via members in SYS1.PARMLIB. These facilities simplify the diagnostic data collection process by enabling you to prepare data collection parameters in advance to ensure that complex dump procedures do not have to be typed in when a problem arises and prompt, error-free action is required.

6.1 IEAABD00, IEADMP00, and IEADMR00

The SYS1.PARMLIB members that can simplify the diagnostic data collection process include:

- IEAABD00: Contains IBM defaults or installation-assigned parameters, or both, for ABDUMP, for use when an ABEND dump is written to a SYSABEND data set.

  SYSABEND is the largest of the ABEND dumps, containing a summary dump for the failing program plus many other areas useful for analyzing processing in the failing program. This dump is formatted. A SYSABEND DD statement controls whether a SYSABEND dump will be captured and where it will be written.

- IEADMP00: Contains IBM defaults and installation parameters for ABDUMP for use when an ABEND dump is written to a SYSUDUMP data set.

  SYSUDUMP is the smallest of the ABEND dumps, containing data and areas only about the failing program. This dump is formatted. A SYSUDUMP DD statement controls whether a SYSUDUMP will be captured and where it will be written.

- IEADMR00: Contains IBM defaults and installation parameters for ABDUMP for use when an ABEND dump is written to a SYSMDUMP data set.These members contain the SDATA and PDATA options that will be used when an abend dump is triggered.

  SYSMDUMP contains a summary dump for the failing program, plus some system data for the failing task. SYSMDUMP dumps are the only ABEND dumps that are unformatted and must be formatted with IPCS. A SYSMDUMP DD statement controls whether a SYSMDUMP will be captured and where it will be written.

These members contain the SDATA and PDATA options that will be used when an abend dump is triggered.

6.1.1 SDATA options

Following are the SDATA options:

- **ALLSDATA:** All the options are automatically specified (except ALLVNUC and NOSYM).

The following parameters request dumps of specific SDATA areas, as indicated:

- **ALLVNUC:** Entire virtual nucleus. SQA, LSQA, and the PSA are included.
- **NOSYM:** No symptom dump is to be produced.
- **SUM:** Requests that the dump contain summary data, which includes the following:
  - Dump title.
  - Abend code and PSW at the time of the error.
  - If the PSW at the time of the error points to an active load module:
    - The name and address of the load module
    - The offset into the load module indicating where the error occurred
    - The contents of the load module
  - Control blocks related to the failing task.
  - Recovery termination control blocks.
  - Save areas.
  - Registers at the time of the error.
  - Storage summary consisting of 1K (1024) bytes of storage before and 1K bytes of storage after the addresses pointed to by the registers and the PSW. The storage will

be printed only if the user is authorized to obtain it, and, when printed, duplicate addresses will be removed.

  - System trace table entries for the dumped address space.

- **NUC:** Read/write portion of the control program nucleus. SQA, LSQA, and the PSA are included.
- **PCDATA:** Program call information for the task being dumped.
- **SQA:** The system queue area.
- **LSQA:** Local system queue area for the address space. If storage is allocated for subpools 229, 230 and 249, they will be dumped for the current task.
- **SWA**: Scheduler work area used for the failing task.
- **CB:** Control blocks related to the failing task.
- **ENQ:** Global resource serialization control blocks for the task.
- **TRT:** System trace table and GTF trace, as available.
- **DM:** Data management control blocks (DEB, DCB, IOB) for the task.
- **IO:** IOS control blocks (UCB, EXCPD) for the task.
- **ERR:** Recovery termination control blocks (RTM2WA, registers from the SDWA, SCB, EED) for the task.

6.1.2 PDATA options (only valid for IEADMP00)

Following are the PDATA options:

- **ALLPDATA** - All the following options are automatically specified.

The following parameters request dumps of specific PDATA areas, as indicated:

- **PSW:** Program status word at entry to ABEND.
- **REGS:** Contents of general registers at entry to ABEND.
- **SA or SAH:** SA requests save area linkage information and a backward trace of save areas. This option is automatically selected if ALLPDATA is specified.
- **SAH:** Requests only save area linkage information.
- **JPA:** Contents of the job pack area that relate to the failing task. These include module names and contents.
- **LPA:** Contents of the LPA related to the failing task. These include module names and contents. Also includes active SVCs related to the failing task.
- **ALLPA:** Contents of both the job pack area and the LPA, as they relate to the failing task, plus SVCs related to the failing task.
- **SPLS:** User storage subpools (0-127, 129-132, 244, 251, and 252) related to the failing task.
- **SUBTASKS:** Problem data (PDATA) options requested for the designated task will also be in effect for its subtasks.

6.2 SDATA and PDATA recommendation

The following SDATA and PDATA parameters will provide you and IBM with sufficient data to solve most problems.

```
SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM)
PDATA=(PSW,REGS,SPLS,ALLPA,SA)
```

6.3 IEADMCxx (dump command parameter library)

IEADMCxx enables you to supply **DUMP** command parameters through a parmlib member. IEADMCxx enables the operator to specify the collection of dump data without having to remember and identify all the systems, address spaces, and data spaces involved.

This parmlib enables you to specify lengthy dump commands without having to reply to multiple writes to operator with reply (WTORs). Any errors in an original specification may be corrected and the dump command re-specified.

IEADMCxx is an installation-supplied member of SYS1.PARMLIB that can contain any valid dump command. A dump command may span multiple lines and contain system static and (dump command SYMDEF-defined) symbols and comments.

Figure 6-1 shows a sample of what might be included in a SYS1.PARMLIB(IEADMCxx) member. As you can see, to key in this data when you need to capture a dump would be time consuming and prone to errors. This simplifies the process so when you need to capture a dump you can refer to the IEADMCxx member in the dump command. For example:

```
DUMP TITLE=(CICS Looping),PARMLIB=CI
```

where CI is the IEADMCxx parmlib member using the suffix SYS1.PARMLIB(IEADMCCI).

The title is the name (1 to 100 characters) you want the dump to have. This title becomes the first record in the dump data set. COMM= and TITLE= are synonyms.

You can also use the parmlib parameter as follows:

```
DUMP COMM=(.........),PARMLIB=(xx)
```

```
TITLE=(DYNDUMP FOR IMS810I,IVP8IRC1,IVP8IDL1,IVP8IM11,
IVP8IM12,IVP8IM13,RRS,APPC)
JOBNAME=(IMS810I,IVP8IRC1,IVP8IDL1,IVP8IM11,IVP8IM12,IVP8IM13,
RRS,APPC)
DSPNAME=('APPC'.*,'RRS'.*)
SDATA=(PSA,SQA,LSQA,RGN,LPA,TRT,CSA,SWA,SUM,ALLNUC,GRSQ)
```

Figure 6-1 IEADMCxx example

6.4 IEASLPxx (SLIP commands)

Use IEASLPxx to contain SLIP commands. The commands can span multiple lines, and the system processes the commands in order.

It is recommended that you move any SLIP commands in the COMMNDxx and IEACMDxx parmlib members into a IEASLPxx parmlib member. By using IEASLPxx to contain your SLIP commands, you avoid restrictions found in other parmlib members.

Figure 6-2 on page 33 shows a sample of what may be contained in SYS1.PARMLIB(IEASLPxx). In this example we are actually suppressing dumps.

```
SLIP SET,C=013,ID=X013,A=NOSVCD,J=JES2,END
SLIP SET,C=213,ID=X213,A=NOSVCD,END
SLIP SET,C=028,ID=X028,A=NOSVCD,END
SLIP SET,C=058,ID=X058,A=NODUMP,DATA=(15R,EQ,4,OR,15R,EQ,8,OR,
     15R,EQ,C,OR,15R,EQ,10,OR,15R,EQ,2C,OR,15R,EQ,30,OR,
     15R,EQ,3C),END
SLIP SET,C=0E7,ID=X0E7,A=NOSVCD,END
SLIP SET,C=0F3,ID=X0F3,A=NODUMP,END
SLIP SET,C=13E,ID=X13E,A=NODUMP,END
SLIP SET,C=1C5,RE=00090004,ID=X1C5,A=NODUMP,END
SLIP SET,C=222,ID=X222,A=NODUMP,END
SLIP SET,C=322,ID=X322,A=NODUMP,END
SLIP SET,C=33E,ID=X33E,A=NODUMP,END
SLIP SET,C=422,ID=X422,A=NODUMP,END
SLIP SET,C=47B,DATA=(15R,EQ,0,OR,15R,EQ,8),ID=X47B,A=NODUMP,END
SLIP SET,C=622,ID=X622,A=NODUMP,END
```

Figure 6-2 SYS1.PARMLIB(IEASLPxx)

Figure 6-3 shows a much more complex SLIP that will capture dumps in multiple MVS images, when a certain message, IXC521I is generated and Register 5 contains some specific data. It will dump the Console address space, the MSOPS address space and also the XCFAS address space.

```
SLIP SET,MSGID=IXC521I,
DATA=(5R?+0,EQ,C8C1E240,+4,EQ,D9C5C1C3),
ACTION=SVCD,JOBLIST=(CONSOLE,XCFAS),
DSPNAME=('XCFAS'.IXCDSL01),
REMOTE=(SYSLIST=(SC55,SC66),
JOBLIST=(CONSOLE,MSOPS,XCFAS),DSPNAME=('XCFAS'.IXCDSL01)),
SDATA=(NUC,CSA,GRSQ,LPA,LSQA,PSA,RGN,SQA,SWA,TRT),
MATCHLIM=3,ID=RON1,END
```

Figure 6-3 SLIP example with increased complexity

The SLIP is activated by issuing the **SET SLIP=xx** MVS command, where xx is the IEASLPxx parmlib member you want to activate.

7

Cancelling tasks and taking dumps

Cancelling a problem task can be initiated from either an MVS console or from an SDSF session running under TSO provided sufficient security privileges have been set up. The MVS console has the highest dispatching priority, which allows commands to be issued at a sufficient level to handle most system loop or hang conditions. An IPL will be required if the problem task cannot be terminated using these procedures. Attempting to cancel a looping task via an SDSF session executing under TSO will often fail because the TSO session will have an insufficient dispatching priority to interrupt the loop process, but this is dependant on the severity of the looping process.

This chapter describes the following:

- Cancelling tasks
- Dumping address spaces
- Diagnostic dumps
- Slip dumps
- Snap dumps
- Stand-alone dumps
- SVC dumps
- Allocating dumps
- Dump analysis

7.1 Cancelling a task

CANCEL can be performed as follows:

1. Issue the **CANCEL jobname** command from the master console, where **jobname** is the looping task.
2. If the looping task is a TSO user, then issue **CANCEL U=tsouser**.
3. Optionally, you might want to take a dump during the cancel. This is achieved by adding the **DUMP** option to the **CANCEL** command. For example:

   ```
   CANCEL jobname,DUMP
   ```

   It is recommended that a separate **DUMP** command be issued, and after this has been successfully processed, then **CANCEL** the task. This will dump according to the JCL SYSABEND, SYSUDUMP, or SYSMDUMP DD statements specified in the JCL.

7.2 Forcing a task

If the attempt to **CANCEL** a problem task (which can sometime require several **CANCEL** attempts), has been unsuccessful, then the next step will be to **FORCE** the task. **FORCE** is not a substitute for **CANCEL**. Unless you issue **CANCEL** first for a cancellable job, the system issues error message IEE838I. The steps to use in the process are:

1. Issue the **CANCEL nnn** command, making several attempts if necessary.
2. Use the **DUMP** command if you want a dump produced. Respond to the prompt for parameters with the jobname or ASID of the “stuck” job, as well as ASID(1)=MASTER.
3. Issue the **FORCE nnn,ARM** command for non-cancellable procedures.
4. Issue the **FORCE nnn** command only when the previous steps fail.

Important: Never use the **FORCE** command without understanding that after issuing **FORCE**, you might have to re-IPL. If you issue **FORCE** for a job in execution or for a time-sharing user, the system deletes the affected address space and severely limits recovery unless you use the ARM parameter. If you need a dump, you must issue a **DUMP** command *before* you issue **FORCE**. After you have issued a **FORCE** command it is usually not possible to get a dump of the failing address space. If your system was part of a global resource serialization ring (GRS=START, GRS=JOIN, or GRS=TRYJOIN was specified at IPL) but has been quiesced (by entering the **VARY GRS(system name),QUIESCE** command), **FORCE** processing might not complete immediately. The system suspends termination of all address spaces holding global resources until the quiesced system rejoins the ring or is purged from the ring. Use a **DISPLAY GRS** command to determine GRS status.When you use the **FORCE** command to end the availability manager (AVM) address space, the operator must restart that address space by issuing the command **START AVM,SUB=MSTR**. You can enter **FORCE** only from a console with master authority.

7.3 Dumping address spaces

Generally the system will automatically capture a dump when it detects a serious error with an operating system component (for example, JES, VTAM, and so forth), a subsystem (for example, CICS, DB2, MQ), or an application program. For most system or subsystem failures an SVC (Supervisor Call) dump will be generated and will be written out to a pre-defined, or dynamically defined, dump data set. You do, however, have the ability to manually capture a dump should you need to capture specific diagnostic data.

7.3.1 DUMP command

The **DUMP** command requests a system dump (SVC dump) of virtual storage. The data set may be either a pre-allocated dump data set named SYS1.DUMPxx, or an automatically allocated dump data set named according to an installation-specified pattern. You should request only one dump at a time on one system. A system writes only one SVC dump at a time, so it does not save time to make several requests at once.

The **DUMP** command is issued at the operator console, or via SDSF (as long as you have operator authority). For example:

```
DUMP COMM=(reason for taking dump)
```

You are then be required to enter via the z/OS **REPLY nn**, response the relevant DUMP options. Figure 7-1 shows the reply sequence in response to the **DUMP COMM** command.

```
R xx, JOBNAME=(CICSTOR1,CICSAOR2,IXGLOGR),CONT
R xx, DSPNAME=('IXGLOGR'.*),CONT
R xx,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,
TRT,CSA,GRSQ,XESDATA,WLM),END
```

Figure 7-1 DUMP COMM example

Parmlib specification for dumps

Alternatively, you can set up the dump parameters in the SYS1.PARMLIB IEADMCxx members. Figure 7-2 shows the same dump request parameters using the IEADMCxx parmlib member.

```
TITLE=(DUMP OF CICS TOR, AOR and LOGGER),
JOBNAME=(CICSTOR1,CICSAOR1,IXGLOGR),
DSPNAME=('IXGLOGR'.*),
SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,
SQA,TRT,CSA,GRSQ,XESDATA,WLM)
```

Figure 7-2 IEADMCxx DUMP Command Format

Issuing dumps

The dump can now be captured using the following command:

```
DUMP TITLE=(CICS Looping),PARMLIB=CI
```

CI is the IEADMCxx parmlib member suffix, that is, SYS1.PARMLIB(IEADMCCI).

The title is the name (1 to 100 characters) you want the dump to have. This title becomes the first record in the dump data set. COMM= and TITLE= are synonyms.

You can also use the parmlib parameter as follows:

```
DUMP COMM=(.........),PARMLIB=(xx)
```

The PARMLIB= parameter allows you to provide lengthy **DUMP** command specifications through a parmlib member. The two alphanumeric characters **xx** indicate the IEADMCxx member of SYS1.PARMLIB that contains the **DUMP** command specification. The syntax of a **DUMP** command specified within the IEADMCxx members of SYS1.PARMLIB is identical to that specified on the **DUMP** command through writes to operator with reply (WTORs).

In response to the **DUMP** command, the system prompts you with the following message for the dump options you want to specify:

```
* id IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
```

Figure 7-3 shows the valid responses to the IEE094D message.

```
R id,U
R id,ASID=n
R id,JOBNAME=jobname
R id,TSONAME=name
R id,DSPNAME=dspname-entry -  which is used for data spaces.
```

Figure 7-3 REPLY options for message IEE094D

Note: The WTOR is not issued when the PARMLIB= parameter is specified.

7.4 Diagnostic data - dumps

There are different types of dumps and traces that can be used to analyze problems. The dump types and the procedures that can be used to initiate these processes are discussed in detail in this section.

Dumps can best be described as a *snapshot* of the system at the time a failure is detected by the operating system or application, or at the time the system is dumped by the operator via the **DUMP** command or the stand-alone dump procedure.

7.4.1 ABEND dumps

The system can produce three types of ABEND dumps, as follows:

SYSABEND The largest of the ABEND dumps, containing a summary dump for the failing program plus many other areas useful for analyzing processing in the failing program. This dump is formatted.

SYSMDUMP Contains a summary dump for the failing program, plus some system data for the failing task. SYSMDUMP dumps are the only ABEND dumps that are unformatted and must be formatted with IPCS.

SYSUDUMP The smallest of the ABEND dumps, containing data and areas only about the failing program. This dump is formatted.

You can obtain SYSABEND, SYSUDUMP, and SYSMDUMP dumps by specifying the correct DD statement in your JCL:

- SYSABEND dumps are formatted as they are created and can be directed to either DASD, TAPE, or SYSOUT.

  ```
  //SYSABEND    DD SYSOUT=*
  ```

- SYSUDUMP dumps are formatted as they are created and can be directed to either DASD, TAPE, or SYSOUT.

  ```
  //SYSUDUMP    DD SYSOUT=*
  ```

- SYSMDUMP dumps are unformatted and must be analyzed using the Interactive Problem Control System (IPCS). These data sets must reside on either DASD or TAPE. Figure 7-4 shows an example of a SYSMDUMP DD statement.

```
//SYSMDUMP    DD DSN=MY.SYSMDUMP,DISP=(,CATLG),UNIT=DISK,
//               SPACE=(CYL,(50,20),RLSE),
//               LRECL=4160,BLKSIZE=4160
```

Figure 7-4 SYSMDUMP DD statement

SLIP command

ABEND dumps can be suppressed using the **SLIP** command in member IEASLPxx in SYS1.PARMLIB. These commands used to reside in member IEACMDxx in SYS1.PARMLIB but it is recommended that you move any **SLIP** commands from IEACMDxx to IEASLPxx to avoid restrictions found in other parmlib members. For example:

- IEASLPxx supports multiple-line commands; IEACMD00 does not.
- IEASLPxx does not require any special command syntax; IEACMD00 does.

Figure 7-5 shows the **SLIP** commands in SYS1.PARMLIB member IEASLP00.

```
SET,C=013,ID=X013,A=NOSVCD,J=JES2,END SLIP SET,C=028,ID=X028,A=NOSVCD,END SLIP
SET,C=47B,DATA=(15R,EQ,0,OR,15R,EQ,8),ID=X47B,A=NODUMP,END SLIP
SET,C=058,DATA=(15R,EQ,4,OR,15R,EQ,8,OR,15R,EQ,C,OR,15R,EQ,10,OR,
15R,EQ,2C,OR,15R,EQ,30,OR,15R,EQ,3C),ID=X058,A=NODUMP,END SLIP
SET,C=0E7,ID=X0E7,A=NOSVCD,END SLIP SET,C=0F3,ID=X0F3,A=NODUMP,END SLIP
SET,C=13E,ID=X13E,A=NODUMP,END SLIP SET,C=222,ID=X222,A=NODUMP,END SLIP
SET,C=322,ID=X322,A=NODUMP,END SLIP SET,C=33E,ID=X33E,A=NODUMP,END SLIP
SET,C=422,ID=X422,A=NODUMP,END SLIP SET,C=622,ID=X622,A=NODUMP,END SLIP
SET,C=804,ID=X804,A=(NOSVCD,NOSYSU),END SLIP SET,C=806,ID=X806,A=(NOSVCD,NOSYSU),END
SLIP SET,C=80A,ID=X80A,A=(NOSVCD,NOSYSU),END SLIP SET,C=9FB,ID=X9FB,A=NOSVCD,J=JES3,END
SLIP SET,C=B37,ID=XB37,A=(NOSVCD,NOSYSU),END SLIP
SET,C=D37,ID=XD37,A=(NOSVCD,NOSYSU),END SLIP SET,C=E37,ID=XE37,A=(NOSVCD,NOSYSU),END
SLIP SET,C=EC6,RE=0000FFXX,ID=XEC6,A=NODUMP,END SLIP
SET,C=EC6,RE=0000FDXX,ID=XXC6,A=NOSVCD,END
```

Figure 7-5 SLIP commands in SYS1.PARMLIB member IEASLP00

7.5 SLIP dumps

The **SLIP** command is set via the z/OS operator **SLIP SET** command. This is a most powerful tool, which allows for great complexity to be used to trigger a dump for a specific situation and can be used to check storage associated with an event and trigger a dump when that event is true. For this discussion we concentrate on the most common use of the **SLIP**, where it is set to trigger a dump when a specific message is written to the console. There are two forms of this **SLIP** command, as follows:

- The first, being the *old* way, where we interrogate storage being used by the WTOR routine
- The second, the later and more understandable version of the message SLIP

7.5.1 SLIP using IGC0003E

It is not necessary to set SLIP traps individually and run a failing job multiple times, using one trap for each execution until a dump is taken. You can set SLIP PER traps at multiple points in a load module as follows: Use a non-IGNORE PER trap to monitor the range that

encompasses all of the points in which you are interested, followed by several IGNORE PER traps to prevent the SLIP action from being taken on the intervening instructions in which you are not interested.

Figure 7-6 shows a **SLIP** command example.

```
SLIP SET,IF,LPAMOD=(IGC0003E,0),
DATA=(1R?+4,EQ,C3E2D8E7,1R?+8,EQ,F1F1F1C5),
JOBNAME=ssidCHIN,
JOBLIST=(ssidMSTR,ssidCHIN),
DSPNAME=('ssidCHIN'.CSQXTRDS),
SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),
MATCHLIM=1,END
```

Figure 7-6 SLIP SET example

SLIP processing

This **SLIP** command example would interrogate the Register 1 storage owned by WTOR routine IGC0003E, and check for the values, staring at offset 4, to see if they match, CSQX (x'C3E2D8E7), and the Register 1 values starting at offset 8, 111X (x'F1F1F1C5). If the matching message was written, in this case, by job ssid CHIN, then the MQ MSTR and CHIN address spaces, and associated CHIN dataspace, will be dumped for a maximum match limit of 1 time. No further dumps will be taken if this job generates this message again.

7.5.2 SLIP using MSGID

Figure 7-7 shows the new form of the message SLIP, and as you can see, is much more user friendly because the MSGID can be included in its ASCII form, not as a HEX representation.

```
SLIP SET,MSGID=CSQX111E,
JOBNAME=ssidCHIN,
JOBLIST=(ssidMSTR,ssidCHIN),
DSPNAME=('ssidCHIN'.CSQXTRDS),
SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),
MATCHLIM=1,END
```

Figure 7-7 SLIP SET using the MSGID parameter

Another simple use of the SLIP is to capture a dump when a specific application abend occurs. For example, you might be getting an S0C4 abend in an application program and require an SVC dump to assist with this instead of a application or transaction dump. Figure 7-8 shows an example of a completion code SLIP.

```
SLIP SET,ENABLE,COMP=0C4,ERRTYP=PROG,JOBNAME=JOBXYZ,LPAMOD=MOD01,END
```

Figure 7-8 Completion Code SLIP example

This example will capture an SVC dump when there is an S0C4 program check interruption while module MOD01 and job JOBXYZ are in control.

7.6 SLIP dump using a z/OS UNIX reason code

If a z/OS UNIX reason code is obtained and additional diagnostics are required, the IBM Support Center personnel may ask that you set a slip to collect a dump or trace on a recreate of the problem. The general instructions on how to gather this documentation follow.

7.6.1 Obtain a dump on a specific reason code

Figure 7-9 shows an example of a SLIP that will produce a dump on the issuance of a specific reason code.

```
SLIP SET,IF,A=SYNCSVCD,
RANGE=(10?+8C?+F0?+1F4?),DATA=(13R??+b0,EQ,xxxxxxxx),DSPNAME=('OMVS'.*),
SDATA=(ALLNUC,PSA,CSA,LPA,TRT,SQA,RGN,SUM),j=jobname,END
```

Figure 7-9 Register 13 reason code SLIP example

In this example:

- xxxxxxxx is the 8 digit (4 byte) reason code that is to be trapped.
- j=jobname is the optional jobname that is expected to issue the error (for example, j=IBMUSER).

Note: In rare instances this SLIP will not capture the requested reason code if the module in question does not use R13 as a data register. Your IBM software support provider can check the specific reason code and determine if this is the reason the SLIP did not match.

7.7 SNAP dumps

Use a SNAP dump when testing a problem program. A SNAP dump shows one or more areas of virtual storage that a program, while running, requests the system to dump. A series of SNAP dumps can show an area at different stages in order to picture a program's processing, dumping one or more fields repeatedly to let the programmer check intermediate steps in calculations. SNAP dumps are preformatted, you cannot use IPCS to format them.

Note: A SNAP dump is written while a program runs, rather than during abnormal end.

7.7.1 Obtaining a SNAP dump

Obtain a SNAP dump by taking the following steps:

1. Code a DD statement in the JCL for the job step that runs the problem program to be dumped with a ddname other than SYSUDUMP, SYSABEND, SYSMDUMP, or another restricted ddname. The statement can specify that the output of the SNAP dump should be written to one of the following:
   - Direct access storage device (DASD). For example:

     ```
     //SNAP1 DD DSN=MY.SNAP.DUMP,DISP=(OLD)
     ```

   - Printer. Note that a printer is not recommended because the printer cannot be used for anything else while the job step is running, whether a dump is written or not.
   - SYSOUT. SNAP dumps usually use SYSOUT. For example:

     ```
     //SNAP1 DD SYSOUT=X
     ```

– Tape. For example:

```
//SNAP1 DD DSN=SNAP.TO.TAPE,UNIT=TAPE,DISP=(OLD)
```

2. In the problem program:
   a. Specify a data control block (DCB) for the data set to receive the dump. For a standard dump, which has 120 characters per line, the DCB must specify:

      ```
      BLKSIZE=882 or 1632
      DSORG=PS
      LRECL=125
      MACRF=(W)
      RECFM=VBA
      ```

      For a high-density dump, which has 204 characters per line and will be printed on an APA 3800 printer, the DCB must specify:

      ```
      BLKSIZE=1470 or 2724
      DSORG=PS
      LRECL=209
      MACRF=(W)
      RECFM=VBA
      ```

   b. Code an OPEN macro to open the DCB.

      Before you issue the SNAP or SNAPX macro, you must open the DCB that you designate on the DCB parameter, and ensure that the DCB is not closed until the macro returns control. To open the DCB, issue the DCB macro with the following parameters, and issue an OPEN macro for the data set:

      ```
      DSORG=PS,RECFM=VBA,MACRF=(W),BLKSIZE=nnn,LRECL=xxx,DDNAME=datasetname
      ```

      The DDNAME can be any name but SYSABEND, SYSMDUMP or SYSUDUMP.

      If the system loader processes the program, the program must close the DCB after the last SNAP or SNAPX macro is issued.

   c. Code a SNAP or SNAPX assembler macro to request the dump. We recommend the use of the SNAPX macro since this allows for programs running in Access-Register (AR) mode to cause the macro to generate larger parameter lists. In the following example, the SNAPX macro requests a dump of a storage area, with the DCB address in register 3, a dump identifier of 245, the storage area's starting address in register 4, and the ending address in register 5:

      ```
      SNAPX  DCB=(3),ID=245,STORAGE=((4),(5))
      ```

      Repeat this macro in the program as many times as wanted, changing the dump identifier for a unique dump. The system writes all the dumps that specify the same DCB to the same data set.

   d. Close the DCB with a CLOSE assembler macro.

7.7.2 Customizing SNAP dumps

An installation can customize the contents of SNAP dumps through the IEAVADFM or IEAVADUS installation exits. IEAVADFM is a list of installation routines to be run and IEAVADUS is one installation routine. The installation exit routine runs during control block formatting of a dump when the CB option is specified on the SNAP or SNAPX macro. The routine can format control blocks and send them to the data set for the dump. See *z/OS MVS Installation Exits*, SC28-1753, for more information.

7.8 Stand-alone dumps

Stand-alone dumps are not produced by z/OS, but by a program called SADMP that is IPLed in place of z/OS.

These dumps show central storage and some paged-out virtual storage occupied by the system or stand-alone dump program that failed. Stand-alone dumps can be analyzed using IPCS.

The term *stand-alone* means that the dump is performed separately from normal system operations and does not require the system to be in a condition for normal operation.

7.8.1 Allocating the stand-alone dump data set

In the SYS1.SAMPLIB data set use the AMDSADDD REXX utility to allocate and initialize the SADMP dump data sets. You can EXEC this REXX utility from the ISPF data set utility option **3.4**, and either VIEW (**V**), BROWSE (**B**) or EDIT (**E**) the SYS1.SAMPLIB data set. Issue the **EXEC** command next to member AMDSADDD. For example:

```
EXEC_____ AMDSADDD
```

Alternatively, issue the following command from the ISPF option line and the utility prompts you as shown in Figure 7-10.

```
TSO EXEC 'SYS1.SAMPLIB(AMDSADDD)'
```

```
What function do you want?
Please enter DEFINE if you want to allocate a new dump data set
Please enter CLEAR if you want to clear an existing dump data set
Please enter REALLOC if you want to reallocate and clear an existing
       dump data set
Please enter QUIT if you want to leave this procedure
define

 Please enter VOLSER or VOLSER(dump_dataset_name)
SYS001
 Please enter the device type for the dump data set
 Device type choices are 3380 or 3390 or 9345
3390
 Please enter the number of cylinders
300
 Do you want the dump data set to be cataloged?
 Please respond Y or N
Y
 TIME-08:59:31 AM. CPU-00:00:03 SERVICE-549023 SESSION-01:18:42 APRIL 9,

 Initializing output dump data set with a null record:
 Dump data set has been successfully initialized

 Results of the DEFINE request:

   Dump data set Name   : SYS1.SADMP
   Volume               : SYS001
   Device Type          : 3390
Allocated Amount     : 3
***
```

Figure 7-10 Prompts issued by the AMDSADDD REXX utility

7.8.2 SADMP program

The SADMP program produces a high-speed, unformatted dump of central storage and parts of paged-out virtual storage on a tape device or a direct access storage device (DASD). The SADMP program that you create must reside on a storage device that can be used to IPL.

You must create the SADMP program by using the AMDSADM macro to produce the following:

- A SADMP program that resides on DASD, with output directed to a tape volume or to a DASD dump data set
- A SADMP program that resides on tape, with output directed to a tape volume or to a DASD dump data set

Create the SADMP program by using the following JCL as an example.

```
//SADMPGEN JOB  MSGLEVEL=(1,1)
//OSG      EXEC PGM=AMDSAOSG
//SYSLIB   DD   DSN=SYS1.MACLIB,DISP=SHR
//         DD   DSN=SYS1.MODGEN,DISP=SHR
//DPLTEXT  DD   DSN=SYS1.NUCLEUS(AMDSADPL),DISP=SHR
//IPLTEXT  DD   DSN=SYS1.NUCLEUS(AMDSAIPD),DISP=SHR
//PGETEXT  DD   DSN=SYS1.NUCLEUS(AMDSAPGE),DISP=SHR
//GENPRINT DD   DSN=SADMP.LIST,DISP=OLD
//GENPARMS DD   *
         AMDSADMP IPL=DSYSDA,VOLSER=SPOOL2,                          X
               CONSOLE=(1A0,3277)
         END
/*
```

7.8.3 ADMSADMP macro

AMDSADMP processing does not allocate the data set or check to see that a valid MVS data set name has been provided. Therefore, you should ensure that:

- The AMDSADDD REXX utility is used to allocate and initialize the same data set name specified on the OUTPUT= keyword.
- The data set name specified should be fully qualified (without quotes).
- The necessary data set management steps are taken so that the SADMP dump data sets will not be placed into a migrated state or moved to a different volume.
- Alphabetic characters appearing in the dump data set name should be specified as capital letters.

Default DASD device

If the default DASD device is to be used and no dump data set name is provided, the SADMP program will assume that the default dump data set name is SYS1.SADMP if the DDSPROMPT=NO parameter was also specified. Otherwise, if DDSPROMPT=YES was specified, the SADMP program will prompt the operator at run-time for a dump data set name to use.

- At run-time, only a null response to message AMD001A will cause the SADMP program to use the default device or dump data set name.
- Do not place a data set that is intended to contain a stand-alone dump on a volume that also contains a page or swap data set that the stand-alone dump program may need to dump. When SADMP initializes a page or swap volume for virtual dump processing, it

checks to see if the output dump data set also exists on this volume. If it does, the SADMP program issues message AMD100I and does not retrieve any data from page or swap data sets on this volume. Thus, the dump may not contain all of the data that you requested. This lack of data may impair subsequent diagnosis.

- You cannot direct output to the SADMP residence volume.

7.8.4 Stand-alone dump procedure

Use the following procedure to initialize the SADMP program and dump storage:

1. Select a processor that was online when the system was stopped.
2. If the processor provides a function to IPL a stand-alone dump without performing a manual STORE STATUS, use this function to IPL SADMP. If you do not use such a function, perform a STORE STATUS before IPLing stand-alone dump. If the operator does not store status, virtual storage is not dumped.

   The hardware store-status facility stores the current program status word (PSW), current registers, the processor timer, and the clock comparator into the unprefixed prefix save area (PSA). This PSA is the one used before the nucleus initialization program (NIP) initialized the prefix register.

   If you IPL the stand-alone dump program from the hardware console, it is not necessary to perform the STORE STATUS operation. Status is automatically stored when stand-alone dump is invoked from the hardware console and automatic store status is on.

   If the operator does not issue the STORE STATUS instruction before IPLing a stand-alone dump, the message `"ONLY GENERAL PURPOSE REGS VALID"` might appear on the formatted dump. The PSW, control registers, and so on, are not included in the dump.

   Note: Do not use the LOAD CLEAR option. Using the LOAD CLEAR option erases main storage, which means that you will not be able to diagnose the failure properly.

3. Make the residence device ready. If it is a tape, mount the volume on a device attached to the selected processor and ensure that the file-protect ring is in place. If it is a DASD volume, ensure that it is write-enabled.
4. IPL SADMP

   SADMP does not communicate with the operator console. Instead, SADMP loads an enabled wait PSW with wait reason code X' 3E0000'. The IPLing of the stand-alone dump program causes absolute storage (X'0' through X'18' and storage beginning at X'110') to be overlaid with CCWs. You should be aware of this and not consider it as a low storage overlay.

   Note: SADMP uses the PSW to communicate with the operator or systems programmer.

   SADMP waits for a console I/O interrupt or an external interrupt.
5. Select the system console or an operator console with a device address that is in the console list that you specified at SADMP generation time (in the CONSOLE keyword of AMDSADMP). At SADMP run time, the operator can choose either a console specified with the CONSOLE= keyword or the system console to control SADMP operation. If an operator console is chosen, press Attention or Enter on that console. (On some consoles, you might have to press Reset first.) This causes an interruption that informs SADMP of the console's address. Message AMD001A appears on the console.

a. Make an output device ready. When you dump to devices that have both real and virtual addresses (for example, dumping a VM system), specify only the real address to the SADMP program. If you are dumping to tape, ensure that the tape cartridge is write-enabled. If you are dumping to DASD, ensure that the DASD data set has been initialized using the AMDSADDD REXX utility.

b. Reply with the device number for the output device. If you are dumping to a DASD device and DDSPROMPT=YES was specified on the AMDSADMP macro, message AMD002A is issued to prompt the operator for a dump data set. If DDSPROMPT=NO was specified, message AMD002A is not issued and the SADMP program assumes that the dump data set name is SYS1.SADMP.

Note: Pressing Enter in response to message AMD001A will cause the SADMP program to use the default device specified on the OUTPUT= keyword of the AMDSADMP macro. If the default device is a DASD device, then pressing the Enter key in response to message AMD001A will cause the SADMP program to use both the default device and the dump data set specified on the OUTPUT= keyword of the AMDSADMP macro. If no dump data set name was provided on the OUTPUT= keyword and the DDSPROMPT=YES keyword was specified, message AMD002A is issued to prompt the operator for a dump data set. If DDSPROMPT=NO was specified, then the SADMP program assumes that the dump data set name is SYS1.SADMP.

If you reply with the device number of an attached device that is not of the required device type, or if the device causes certain types of I/O errors, SADMP might load a disabled wait PSW. When this occurs, use procedure b to restart SADMP.

7.8.5 SADMP processing

SADMP prompts you, with message AMD011A, for a dump title. When no console is available, run SADMP without a console.

- Ready the default output device that was specified on the OUTPUT parameter on the AMDSADMP macro. For tapes, ensure that the tape cartridge is write-enabled. For DASD, ensure that the dump data set has been initialized using the AMDSADDD REXX utility.
- Enter an external interruption on the processor that SADMP was IPLed from. SADMP proceeds using the default output device, the default dump data set, or both. No messages appear on any consoles; SADMP uses PSW wait reason codes to communicate to the operator.

When SADMP begins and finishes dumping central storage, it issues message AMD005I to display the status of the dump. SADMP may end at this step.

When SADMP begins dumping real storage it issues message AMD005I. Message AMD095I is issued every 30 seconds to indicate the progress of the dump. Message AMD005I will be issued as specific portions of real storage have been dumped, as well as upon completion of the real dump. SADMP may end at this step.

If you specified PROMPT on the AMDSADMP macro, SADMP prompts you for additional storage that you want dumped by issuing message AMD059D.

SADMP dumps instruction trace data, paged-out virtual storage, the SADMP message log, and issues message AMD095I every 30 seconds to indicate the progress of the dump.

When SADMP completes processing, SADMP unloads the tape, if there is one, and enters a wait reason code X'410000'.

7.9 SVC dumps

SVC dumps can be used in different ways:

- Most commonly, a system component requests an SVC dump when an unexpected system error occurs, but the system can continue processing.
- An authorized program or the operator can also request an SVC dump (by using the **SLIP** or **DUMP** command) when they need diagnostic data to solve a problem.

SVC dumps contain a summary dump, control blocks, and other system code, but the exact areas dumped depend on whether the dump was requested by a macro, command, or SLIP trap. SVC dumps can be analyzed using IPCS.

SVC dump processing stores data in dump data sets that you pre-allocate manually, or that the system allocates automatically, as needed. You can also use pre-allocated dump data sets as a backup in case the system is not able to allocate a data set automatically. To prepare your installation to receive SVC dumps, you need to provide SYS1.DUMPxx data sets. These data sets will hold the SVC dump information for later review and analysis. This section describes how to set up the SVC dump data sets.

Note: An incomplete dump, or partial dump, is almost always useless.

7.10 Dump data set size

When the z/OS operating system initiates, or is instructed to dump an address space, or multiple address spaces, the data will be written to a dump data set on a disk device. These data sets can be pre-allocated, as is the case with the traditional SYS1.DUMPxx data sets, or dynamically allocated, in which case a new data set will be allocated whenever the system requests a dump.

In conjunction with the dump data set, the user defined MAXSPACE parameter must be set to ensure sufficient memory is allocated to retain the dump information in use by the address spaces and system areas. The recommended MAXSPACE in today's environment is 2500Mb, which is a lot different than the IBM default of 450Mb. This will need to be increased as products, such as DB2, start to make use of 64-bit virtual addressability.

Application-related dumps can be written to a data set pointed to by the SYSMDUMP DD statement in the JCL. The data written to the SYSMDUMP data set is always required to diagnose application-related problems running under Language Environment control.

The DCB requirements for dump data sets are as follows:

7.10.1 Allocating SYS1.DUMPxx data sets

Allocate SYS1.DUMPxx data sets using the following requirements:

- Name the data set SYS1.DUMPxx, where xx is a decimal number of 00 through 99.
- Select a device with a track size of 4160 bytes. The system writes the dump in blocked records of 4160 bytes. If you want to increase the Block Size for the dump data set, you can do so as long as the blocking factor does not exceed 7, that is, 29120, and the Record Format (RECFM) must be Fixed Block Spanned (FBS).
- Initialize with an end of file (EOF) record as the first record.

- Allocate the data set before requesting a dump. Allocation requirements are:

  UNIT A permanently resident volume on a direct access device.

  DISP Catalog the data set (CATLG). Do not specify SHR.

  VOLUME Place the data set on only one volume. Allocating the dump data set on the same volume as the page data set could cause contention problems during dumping, as pages for the dumped address space are read from the page data set and written to the dump data set.

  SPACE An installation must consider the size of the page data set that will contain the dump data. The data set must be large enough to hold the amount of data as defined by the MAXSPACE parameter on the **CHNGDUMP** command, VIO pages, and pageable private area pages. SVC dump processing improves service by allowing secondary extents to be specified when large dump data sets are too large for the amount of DASD previously allocated. An installation can protect itself against truncated dumps by specifying secondary extents and by leaving sufficient space on volumes to allow for the expansion of the dump data sets. For the SPACE keyword, you can specify CONTIG to make reading and writing the data set faster. Request enough space in the primary extent to hold the smallest SVC dump expected. Request enough space in the secondary extent so that the primary plus the secondary extents can hold the largest SVC dump. The actual size of the dump depends on the dump options in effect when the system writes the dump.

  Note: Approximately 250 cylinders will be sufficient for most single address space SVC dump requirements.

The system writes only one dump in each SYS1.DUMPxx data set. Before the data set can be used for another dump it can be cleared by using the **DUMPDS** command with the **CLEAR** keyword. The format if the command is:

```
DUMPDS CLEAR,DSN=xx
```

In this example, xx is the SYS1.DUMPxx identifier. You can abbreviate the **DUMPDS** command to **DD**, for example:

```
DD CLEAR,DSN=01
```

7.10.2 Dynamic allocation of SVC dump data sets

SVC dump processing supports automatic allocation of dump data sets at the time the system writes the dump to DASD. The dump can be allocated from a set of DASD volumes or SMS classes. When the system captures a dump, it allocates a data set of the correct size from the resources you specify. If automatic allocation fails, pre-allocated dump data sets are used. If no pre-allocated SYS1.DUMPnn data sets are available, message IEA793A is issued, and the dump remains in virtual storage. SVC dump periodically retries both automatic allocation and writing to a pre-allocated dump data set until successful or until the captured dump is deleted either by operator intervention or by the expiration of the CHNGDUMP MSGTIME parameter governing message IEA793A.

You can specify the command instructions to enable or disable automatic allocation either in the COMMNDxx parmlib member, to take effect at IPL, or from the operator console at any time after the IPL, to dynamically modify automatic allocation settings. The **DUMPDS** command provides the following flexibility:

- Activate automatic allocation of dump data sets

- Add or delete allocation resources
- Direct automatic allocation to SMS or non-SMS managed storage
- Deactivate automatic allocation of dump data sets
- Reactivate automatic allocation of dump data sets
- Change the dump data set naming convention

Automatic allocation can be set up using the following steps:

1. Set up allocation authority.
2. Establish a name pattern for the data sets.
3. Define resources for storing the data sets.
4. Activate automatic allocation.

Once active, allocation to SMS classes and DASD volumes is done starting from the first resource you added with the **DUMPDS ADD** command until unsuccessful, then the next resource is used. If you have defined both DASD volumes and SMS classes, SMS classes are used first. Allocation to DASD volumes is not multivolume or striped, whereas allocation to SMS classes can be multivolume or striped, depending on how the storage class is set up by the installation.

The steps to initiate automatic dump data set allocation are:

1. Associate the DUMPSRV address space with a user ID.
2. Authorize DUMPSRV's user ID to create new dump data sets.
3. Set up your installation data set name pattern using the **DUMPDS** command:

```
DUMPDS NAME=SC68;.&JOBNAME;.Y&YR4;M&MON;.D&DAY;T&HR;&MIN;.S&SEQ;
```

4. Add dump data set resources that can be used by the automatic allocation function:

```
DUMPDS ADD,VOL=(SCRTH1,HSM111)
DUMPDS ADD,SMS=(DUMPDA)
```

5. Activate automatic dump data set allocation using the **DUMPDS** command:

```
DUMPDS ALLOC=ACTIVE
```

Note: These steps can be performed after IPL using the **DUMPDS** command from an operator console, or early in IPL by putting the commands in the COMMNDxx parmlib member and pointing to the member from the IEASYSxx parmlib member using CMD=xx.

If you use COMMNDxx, you may want to specify DUMP=NO in the IEASYSxx parmlib member to prevent dumps taken during IPL from being written to SYS1.DUMPxx data sets.

Reinforcing an earlier statement, an incomplete dump, or a partial dump, is useless 99 percent of the time.

7.11 Dumping multiple address spaces in a sysplex

The example discussed in this section can be used as a guide to dump multiple address spaces in a sysplex environment.

Create a SYS1.PARMLIB member called IEADMCI1 containing the DUMP parameters that follow Figure 7-11.

Figure 7-11 shows an example of setting up a dump request by using the IEADMCI1 SYS1.PARMLIB members.

```
JOBNAME=(job1,job2,job3,job4),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ),
REMOTE=(SYSLIST=*('job1','job2','job3','job4'),SDATA)
```

Figure 7-11 IEADMCI1 example

In this example the parameters are:

- job1 = IMS Control Region Jobname
- job2 = IMS DLI region Jobname
- job3 = DBRC Region Jobname
- job4 = IRLM Region Jobname (If IRLM DB Locking is used)

Figure 7-12 shows the creation of a second SYS1.PARMLIB member called IEADMCI2 containing the DUMP parameters that follow the figure.

```
JOBNAME=(job5,job6,job7),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ,XESDATA),
REMOTE=(SYSLIST=*('job5','job6','job7'),SDATA)
```

Figure 7-12 IEADMCI2 example

In this example the parameters are:

- job5 = CCTL Region 1
- job6 = CCTL Region 2
- job7 = CCTL Region 3

7.11.1 Requesting a dump

To request a dump to be captured as per the IEADMCI1 and IEADMCI2 parmlib members, issue the following MVS command:

```
DUMP TITLE=(IMS/CCTL sysplex DUMPS),PARMLIB=(I1,I2)
```

Two dump data sets are created on each MVS image in the sysplex matching the REMOTE specifications for the JOBNAMEs.

If the dataspace DSPNAME parameter is specified, for example, `DSPNAME=('job1'.*)`, then the same dataspace is dumped in the associated address spaces in the other systems if the DSPNAME parameter is included on the REMOTE statement. For example:

```
REMOTE=(SYSLIST=*('job1','job2','job3','job4'),SDATA,DSPNAME)
```

Figure 7-13 and Figure 7-14 show an alternative where IEASLPxx has been used containing the following SLIP entries, using the IEASLPxx example.

In Figure 7-13 the parameters are:

- job1 = IMS Control Region Jobname
- job2 = IMS DLI region Jobname
- job3 = DBRC Region Jobname

- job4 = IRLM Region Jobname (If IRLM DB Locking is used)

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),A=(SYNCSVCD,TARGETID),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ),
JOBLIST=(job1,job2,job3,job4),ID=IMS1,TARGETID=(IMS2),
REMOTE=(JOBLIST,SDATA),D,END
```

Figure 7-13 IEASLPxx example

In Figure 7-14 the parameters are:

- job5 = CCTL Region 1
- job6 = CCTL Region 2
- job7 = CCTL Region 3

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),
JOBLIST=(job5,job6,job7),ID=IMS2,
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,XESDATA),
REMOTE=(JOBLIST,SDATA),
D,END
```

Figure 7-14 IEASLPxx example

Before activating the SLIP, ensure that any existing PER SLIP for each MVS image in the sysplex is disabled, as follows:

```
ROUTE *ALL,SLIP,MOD,DISABLE,ID=trapid
```

To activate the SLIP trap and trigger the associated SVC dumps, enter the following MVS commands:

```
SET SLIP=xx
SLIP MOD,ENABLE,ID=IMS1
```

Two dumps are then be captured on each MVS image in the sysplex matching the REMOTE specifications.

7.12 Dump analysis and elimination (DAE)

DAE suppresses dumps that match a dump you already have. Each time DAE suppresses a duplicate dump, the system does not collect data for the duplicate or write the duplicate to a data set. The ADYSETxx members in SYS1.PARMLIB control the DAE facility. If you find that dumps are being suppressed, as indicated by the messages IEA820I, IEA848I, or IEA838I, review DAE to ensure that you do not suppress this dump. A stop and start of DAE is required to reset the dump suppression count.

A stop of DAE is performed by issuing a **SET DAE=xx**, where the xx in the ADYSETxx member contains a DAE=STOP,GLOBALSTOP command.

Restart DAE by **SET DAE=xx**, where xx is the active ADYSETxx parmlib member. This is often ADYSET00.

7.13 Partial dumps

How can you determine if the dump that has been captured is a complete dump. A partial, or incomplete dump will be missing some key areas of storage that in most cases will make the dump useless when it comes to efficient problem diagnosis. This section describes:

- Partial dumps
- SDATA dump options

Apart from the obvious message that will be generated in the z/OS system log that indicates a dump is partial, or that the dump MAXSPACE has been reached, the only other way to determine if the dump is partial is to interrogate the dump using the Interactive Problem Control System (IPCS).

Figure 7-15 shows an example of the IEA042I message.

```
IEA043I SVC DUMP REACHED MAXSPACE LIMIT - MAXSPACE=xxxxxxxx MEG or
IEA611I PARTIAL DUMP ON dsname
```

Figure 7-15 IEA611I message indicating partial dump

Figure 7-16 shows the result of the IPCS Locate command that can be issued to interrogate the storage to indicate if the dump taken was partial. In this case we are looking at storage at address x'E0' for a length of 16 bytes.

```
Command ===> ip l e0. block(0) l(16)
********************************************** TOP OF DATA **************
LIST E0. BLOCK(0) LENGTH(X'10') AREA
BLOCK(0) ADDRESS(E0.)
000000E0. 00000000 30000000 00000000 00000000 |................ |
**********************************************END OF DATA **************
```

Figure 7-16 IPCS Storage Address Locate for IEA611I reason

The 4 words found at location x'E0' contain partial dump reason codes. These codes are mapped by DSECT SDRSN, and can found in the z/OS data areas manual. The flags are also listed in the z/OS Messages under message IEA611I. The description listed under IEA611I for x'30000000' in the second word is:

20000000 -The system detected an error in the SVC dump task and gave recovery control.

10000000 - The SVC dump task failed.

The reason codes indicate an error caused recovery for the SVC dump task (IEAVTSDT) to be driven that also failed while attempting to take the dump.

If the values displayed at location x'E0' are all zero, then the dump is not partial.

7.14 SDATA options

Figure 7-17 shows the result of the IPCS control block format of the CVT to interrogate the SDATA options that were in effect when the dump was taken. The command is:

```
cbf cvt+23c?+9c str(sdump) view(flags)
```

```
SDUMP_PL: 00FB357C

   ==> FLAGS SET IN SDUFLAG0:
    Set system non-dispatchable while dumping global storage.

   ==> FLAGS SET IN SDUFLAG1:
    SYSMDUMP request.
    SUMLIST specified.
    Ignore CHNGDUMP parameters.
    TSO user extension is present.
    48+ byte parameter list.

   ==> FLAGS SET IN SDUSDATA:
    Dump SQA.
    Dump LSQA.
    Dump rgn-private area.
    Dump LPA mod. for rgn.
    Dump trace data.
    Dump SWA.
    Do not dump all PSA.
```

*Figure 7-17 Example of IPCS "*cbf cvt+23c?+9c str(sdump) view(flags)" command

Even though the SDATA RGN parameter has been specified, the fact that some areas of RGN storage may have been paged out when the dump was taken can result in a `Storage not available` condition.

8

z/Architecture and addressing

z/Architecture is the next step in the evolution from the System/360™ to the System/370™, System/370 extended architecture (370-XA), Enterprise Systems Architecture/370* (ESA/370),and Enterprise Systems Architecture/390® (ESA/390). In order to understand z/Architecture you have to be familiar also with the basics of ESA/390 and its predecessors.

An address space maps all of the available addresses, and includes system code and data as well as user code and data. Thus, not all of the mapped addresses are available for user code and data. This limit on user applications was a major reason for System/370 Extended Architecture (370-XA) and MVS/XA™. Because the effective length of an address field expanded from 24 bits to 31 bits, the size of an address space expanded from 16 megabytes to 2 gigabytes. An MVS/XA address space is 128 times as big as an MVS/370 address space.

This chapter describes:

- Program status word (PSW)
- Address space addressability
- Dumps in 31-bit and 64-bit modes

8.1 Introduction to program status word (PSW)

One very important piece of information that will be crucial to your ability to diagnose a problem on z/OS is the program status word, more commonly referred to as the *PSW*. The PSW includes the instruction address, condition code, and other information to control instruction sequencing and to determine the state of the CPU. The active or controlling PSW is called the *current* PSW.

The PSW is so important because it keeps track of the progress of the system and the executing program. The current PSW usually points to the address of the next instruction to be executed. In some specific cases the PSW will point to the address of the failing instruction. This occurs when the interrupt code is 0010, which is a segment translation exception, or interrupt code 0011, which is a page translation exception.

What this means is that when a task abends and a dump is taken, the PSW is pointing to the next instruction that will be executed in the failing program. By subtracting the instruction-length code (ILC) from the PSW address, you will be looking at the failing instruction for which the abend was triggered.

Note: For page translation and segment translation errors the PSW points to the failing instruction.

8.1.1 Program status word details

The current PSW is a storage circuit located within the CP. It contains information required for the execution of the currently active program, or in other words, it contains the current state of a CP. It has 16 bytes (128 bits). It governs the program currently being executed.

Figure 8-1 describes the PSW from bits 0 to 31.

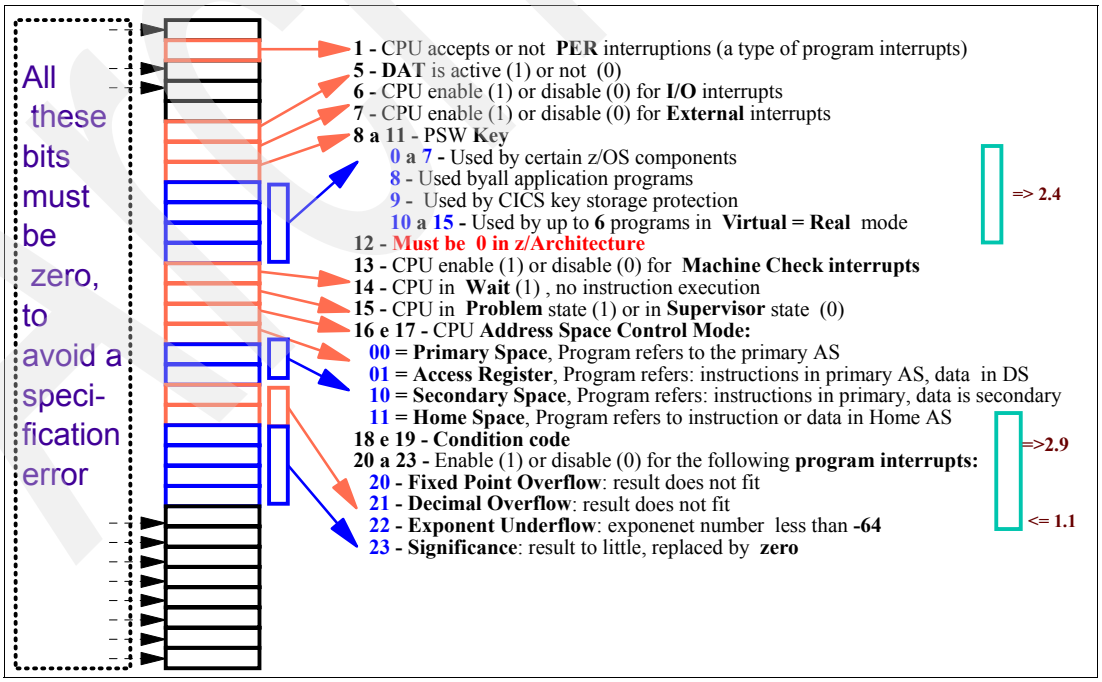


Figure 8-1 PSW from bit 0 to bit 31

PER mask - R (bit 1)

Bit 1 controls whether the CP is enabled for interrupts associated with program-event recording (PER). When the bit is zero, no PER event can cause an interruption. When the bit is one, interruptions are permitted, subject to the PER-event-mask bits in control register 9.

DAT mode - T (bit 5)

Bit 5 controls whether implicit dynamic address translation of logical and instruction addresses used to access storage takes place. When the bit is zero, DAT is off, and logical and instruction addresses are treated as real addresses. When the bit is one, DAT is on, and the dynamic-address-translation mechanism is invoked.

I/O mask - IO (bit 6)

Bit 6 controls whether the CP is enabled for I/O interruptions. When the bit is zero, an I/O interruption cannot occur. When the bit is one, I/O interruptions are subject to the I/O-interruption subclass-mask bits in control register 6. When an I/O-interruption subclass-mask bit is zero, an I/O interruption for that I/O-interruption subclass cannot occur; when the I/O-interruption subclass-mask bit is one, an I/O interruption for that I/O-interruption subclass can occur.

External mask - EX (bit 7)

Bit 7 controls whether the CP is enabled for interruption by conditions included in the external class. When the bit is zero, an external interruption cannot occur. When the bit is one, an external interruption is subject to the corresponding external subclass-mask bits in control register 0; when the subclass-mask bit is zero, conditions associated with the subclass cannot cause an interruption; when the subclass-mask bit is one, an interruption in that subclass can occur.

PSW key (bits 8-11)

Bits 8-11 form the access key for storage references by the CP. If the reference is subject to key-controlled protection, the PSW key is matched with a storage key when information is stored or when information is fetched from a location that is protected against fetching. However, for one of the operands of each of MOVE TO PRIMARY, MOVE TO SECONDARY, MOVE WITH KEY, MOVE WITH SOURCE KEY, and MOVE WITH DESTINATION KEY, an access key specified as an operand is used instead of the PSW key.

Machine-check mask - M (bit 13)

Bit 13 controls whether the CP is enabled for interruption by machine-check conditions. When the bit is zero, a machine-check interruption cannot occur. When the bit is one, machine-check interruptions due to system damage and instruction-processing damage are permitted, but interruptions due to other machine-check-subclass conditions are subject to the subclass-mask bits in control register 14.

Wait state - W (bit 14)

When bit 14 is one, the CP is waiting; that is, no instructions are processed by the CP, but interruptions may take place. When bit 14 is zero, instruction fetching and execution occur in the normal manner. The wait indicator is on when the bit is one. When in wait state, the only way of getting out of such state is through an Interruption, or IPL (a z/OS boot). Certain bits – when off – in the current PSW place the CP in a disabled state, that is, it does not accept Interrupts. So, when z/OS by any error reason (software or hardware) decides to stop a CP, it sets the PSW in Disable and Wait state, forcing an IPL as the way to get the CP in running state.

Problem state - P (bit 15)

When bit 15 is one, the CP is in the problem state. When bit 15 is zero, the CP is in the supervisor state. In the supervisor state, all instructions are valid. In the problem state, only those instructions are valid that provide meaningful information to the problem program and that cannot affect system integrity; such instructions are called unprivileged instructions. The instructions that are never valid in the problem state are called privileged instructions. When a CP in the problem state attempts to execute a privileged instruction, a privileged-operation exception is recognized. Another group of instructions, called semiprivileged instructions, are executed by a CP in the problem state only if specific authority tests are met; otherwise, a privileged-operation exception or a special-operation exception is recognized.

Address-space control - AS (bits 16-17)

Bits 16 and 17, in conjunction with PSW bit 5, control the translation mode.

Condition code - CC (bits 18-19)

Bits 18 and 19 are the two bits of the condition code. The condition code is set to 0, 1, 2, or 3, depending on the result obtained in executing certain instructions. Most arithmetic and logical operations, as well as some other operations, set the condition code. The instruction BRANCH ON CONDITION can specify any selection of the condition-code values as a criterion for branching.

The part of the CP that executes instructions is called the arithmetic logic unit (ALU). The ALU has internally four bits that are set by certain instructions. At the end of such instructions this 4-bit configuration is mapped into bits 18 and 19 of the current PSW.

As an example, the instruction COMPARE establishes a comparison between two operands. The result of the comparison is placed in the CC of the current PSW, as follows:

- CC=00, then the operands are equal
- CC=01, then first operand is lower
- CC=10, then first operand is greater

To test the contents of a CC (set by a previous instruction), use the BRANCH ON CONDITION (BC) instruction. It has an address of another instruction (branch address) to be executed depending on the comparison of the CC and a mask M. The instruction address in the current PSW is replaced by the branch address, if the condition code has one of the values specified by M; otherwise, a normal instruction sequencing proceeds with the normal updated instruction address. Here are the following types of codes:

- Condition code (bits 18,19 PSW),
- Return code, a code associated with how a program ended
- Completion code, a code associated with how a task ended
- Reason code, a code passed in the GPR 15 detailing more about how a task ended

Program Mask (bits 20-23)

During the execution of an arithmetic instruction, the CP may find some unusual (or error) condition, such as: overflows, loss of significance, or underflow. In these cases, the CP generates a program interrupt. When this interrupt is treated by z/OS, usually the current task is abended. However, in certain situations the programmer does not want an ABEND, so through the instruction SET PROGRAM MASK (SPM), the programmer can mask such interrupts by setting *off* some of the program mask bits. Each bit is associated with one type of condition:

- Fixed point overflow (bit 20)
- Decimal overflow (bit 21)
- Exponent underflow (bit 22)

- Significance (bit 23)

Observe that the active program is informed about these events through the condition code posted by the instruction where the events described happened.

The contents of the CP can be totally changed by two events:

- Loading a new PSW from storage along an Interruption
- Executing the instruction LPSW, which copies 128 bits from memory to the current PSW.

Extended addressing mode - EA, BA (bits 31-32)

The combinations of the bits 31 and 32, tell the addressing mode (24, 31 or 64) of the running program. Bit 31 controls the size of effective addresses and effective-address generation in conjunction with bit 32, the basic-addressing-mode bit. When bit 31 is zero, the addressing mode is controlled by bit 32. When bits 31 and 32 are both one, 64-bit addressing is specified.

8.2 What is addressability?

One of the major developments of the MVS operating system was the implementation of 31-bit addressing. Prior to MVS/XA the highest virtual storage location that could be addressed was 16 megabytes, or hexadecimal *FFFFFF*. Actually, it was one byte less that 16 megabytes, because we start at zero. As applications grew larger the 24-bit architecture limitations were recognized, and 31-bit addressability was introduced. The 31-bit standard increased the amount of addressable virtual storage to 2 gigabytes. The addressing mode of a program is determined by the high order bit (bit 32 of the PSW) of the instruction address. If this bit is set to *1* the processor is running in 31-bit mode. If it is *0* then the processor is running in 24-bit mode.

We have now taken the next step in storage addressability with z/OS implementing 64-bit addressing. This means that the maximum storage that can be addressed is 2 ** 64, or 16 exabytes. The highest address when running in 64-bit mode is X'FFFFFFFF_FFFFFFFF' as opposed to the previous 31-bit high address of X'7FFFFFFF'.

8.2.1 Format of the PSW

Prior to z/OS 64-bit mode operations, the PSW was 64 bits in length and comprised of two 32-bit words. The first 32 bits (identified as bits 0 through 31) relate to system state and mode status, but the second 32 bits (identified as bits 32 through 63 as shown in Figure 8-3 on page 61) indicate the addressing mode in the first bit and the address of the next instruction in bits 33 through 63. The second word is what will interest us in most cases, as shown in Figure 8-2 on page 60.

For example,

```
PSW: 075C2000 82CC5BCC Instruction length: 02
```

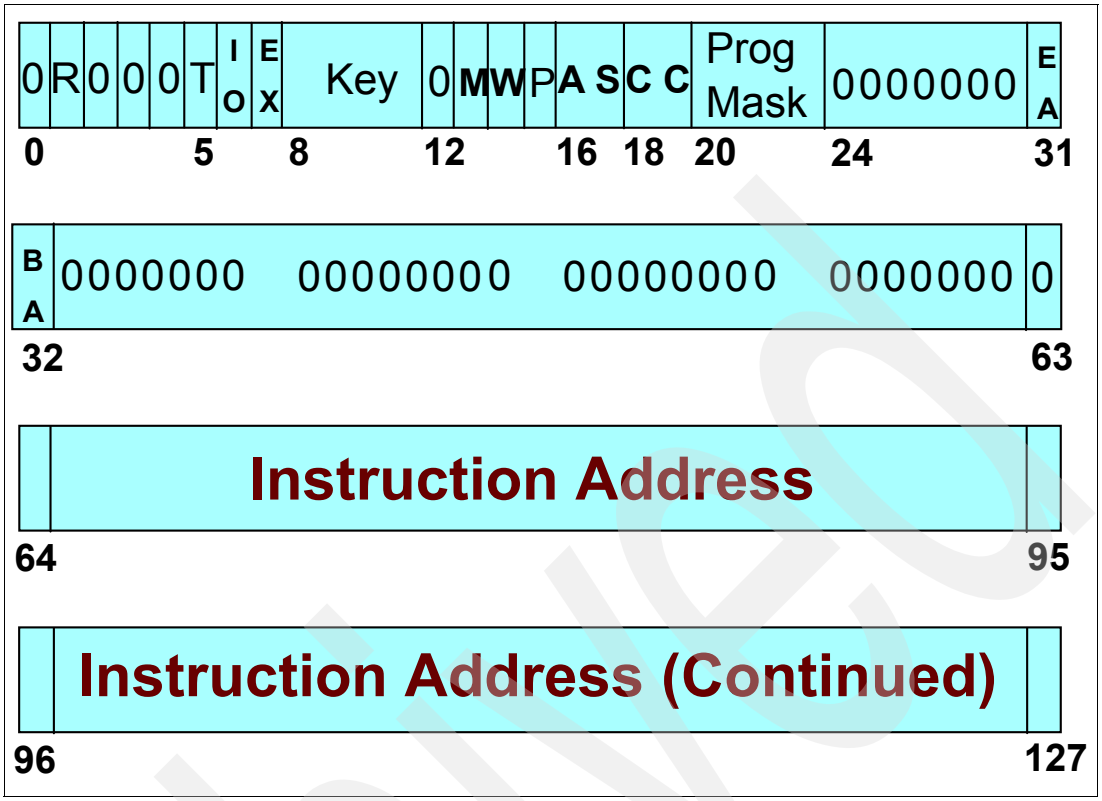


Figure 8-2 Program-status-word format

Instruction address (bits 64 to 127)

Bits 64 to 127, shown in Figure 8-2, point to the storage address of the next instruction to be executed by this CP. When an instruction is fetched from central storage, its length is automatically added to this field. Then it will point to the next instruction address. However, there are instructions such as a BRANCH that may replace the contents of this field, pointing to the branched instruction. The address contained in this PSW field may have 24, 31, or 64 bits, depending on the addressing mode attribute of the executing program. For compatibility reasons, old programs that still address small addresses are still allowed to execute. When in 24- or 31-bit addressing mode, the left-most bits of this field are filled with zeroes.

CP interrupts

The CP has an interrupt capability, which permits the CP to switch rapidly to another program in response to exceptional conditions and external stimuli. When an interrupt occurs, the CP places the current PSW in an assigned storage location, called the old-PSW location, for the particular class of interrupt. The CP fetches a new PSW from a second assigned storage location. This new PSW determines the next program to be executed. When it has finished processing the interrupt, the program handling the interrupt may reload the old PSW, making it again the current PSW, so that the interrupted program can continue.

There are six classes of interrupt: external, I/O, machine check, program, restart, and supervisor call. Each class has a distinct pair of old-PSW and new-PSW locations permanently assigned in real storage.

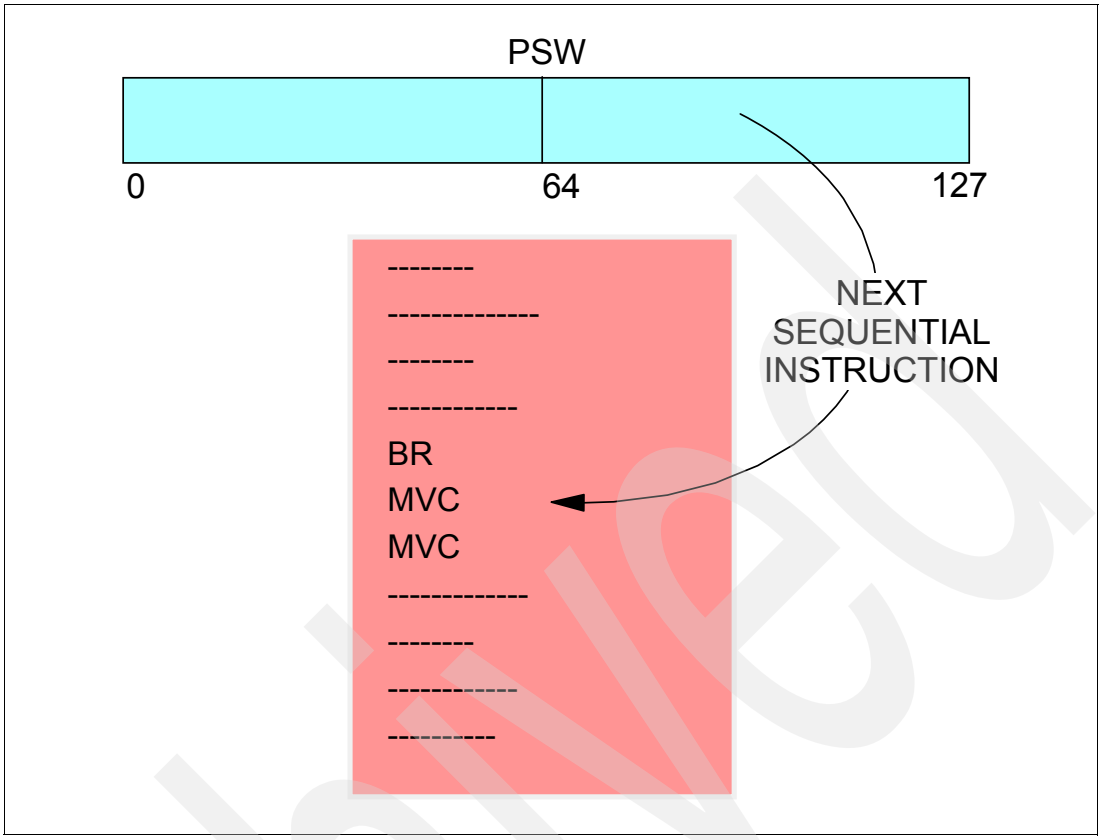


Figure 8-3 Next sequential instruction address

PSW second word

Consider the following PSW example:

```
PSW: 075C2000 82CC5BCC Instruction length: 02
```

The second word of the PSW is **82CC5BCC**. The first number, **8**, indicates that this program is executing in 31-bit mode. In other words, this program runs above the 16 megabyte line. The number **8** in binary is **1000** which indicates the addressing mode bit 32 is ON. A value of zero decimal would be binary zero, 0000, indicating that the addressing mode bit 32 was OFF, which identifies that this location was below the 16 bit line, or in 24-bit mode.

The remaining data points to the next instruction to be executed, in this case, **2CC5BCC**. For the sake of correctness the full address would be **02CC5BCC**.

Subtracting the instruction length value, in this case, 2, from the PSW address, would result in **02CC5BCA**, which would point to the failing instruction.

The PSW has now changed and the z/OS 128-bit PSW is converted by MVS to a 64-bit double word and the z/OS formatted PSW is stored in control blocks. The PSW is represented as follows:

```
AMODE 24
07850000 00000000 00000000 00065788 078D0000 00065788
AMODE 31
04041000 80000000 00000000 00FE5768 040C1000 80FE5768
AMODE 64
04045001 80000000 00000000 01685B28 040C5001_81685B28
```

The bold form of the PSW indicates the *converted* z/OS PSW. The underscore between the two words of the converted PSW indicates that this a 64-bit (above the bar) address.

As you can see, it looks similar to the 31-bit PSW except for the non-zero value of bit 31 in the 1st word of the PSW, 040C500**1,** as well as the non-zero value in bit 32 of the PSW, which is the 1st bit of the second word, **8**1685B28. It is the use of bit 31 and bit 32 that indicates this is a 64-bit address. The address to interrogate in this case would be **1_81685B28.**

In many cases, for most current applications, you will still be interrogating 31-bit storage addresses, but in the future as more application make use of the extended addressability, you will reference storage pointed to by the Addressing Mode (AMODE) 64-bit PSW.

"Using IPCS to find the failing instruction" on page 91 shows how to use this address to locate the failing instruction.

8.3 Is my dump from a z/OS 31-bit or 64-bit system?

The easiest way to determine this is to use ISPF to browse the unformatted dump data set.

The header for each record in the dump will show **DR1** for a system running in 31-bit mode and **DR2** for a 64-bit system dump. Figure 8-4 shows an ISPF browse of the dump data set.

```
BROWSE    APSG.SC48TS.DUMP1
 Command ===>
****************************
DR2 H .....................
DR2 CV.....................
DR2 CV..................]°.
DR2 CV..................]µ.
DR2 CV..................]^.
DR2 CV..................]{.
```

Figure 8-4 64-bit architecture dump header record

A slightly more complex method, for those familiar with IPCS, is as follows:

- 31-bit (2Gb) MVS address spaces have architected Prefix Save Areas starting at x'0' in low-core. These start with the restart new PSW (which begins "040C..."). This is what you would expect to see in low-core dumps from systems that are not running on the new H/W, or which are using the new 64-bit support hardware, but are not running in 64-bit mode.
- If an MVS image has been IPLed to exploit 64-bit architecture, the low-core will look completely different. The PSA is now 2Kb in size, rather than 1Kb and the format of the PSA starting from x'0' is completely different. Only a few of the fields are retained (for compatibility purposes), for example, the CVT address, the current TCB address and current ASCB address.
- To quickly identify whether a dump was taken from an image exploiting the 64-bit architecture you can look at offset x'A3'. If the value x'01' is set, this dump comes from an MVS image running in 64-bit mode. If x'00' is set, it is running in 31-bit mode. Currently no other bits are used in this byte.

It must be said, that apart from the historical significance, you will not see many non-64 bit dumps in most current z/OS environments.

9

z/OS trace facilities

Another useful source of diagnostic data is the *trace*. Tracing collects information that identifies ongoing events that occur over a period of time. Some traces are running all the time so that trace data will be available in the event of a failure. Other traces must be explicitly started to trace a defined event.

In this chapter, the following trace activity is described:

- GTF trace
- Component trace
- Master trace
- GFS trace
- System trace
- SMS tracing

9.1 Using the DISPLAY TRACE command

To display the current trace option in effect issue the MVS **DISPLAY TRACE** command. Figure 9-1 shows an example of the output generated by the **DISPLAY TRACE** command.

Figure 9-1 shows that we have system trace (ST) enabled, with 256K allocated for the system trace table on each processor and 512K allocated to the system trace table buffers. Address space (AS) tracing is ON and branch tracing is OFF, as is explicit software tracing. Master tracing is ON with a master trace table size of 24K. This also displays the status of component and sub-component traces.

```
IEE843I 10.27.13  TRACE DISPLAY 416
        SYSTEM STATUS INFORMATION
 ST=(ON,0256K,00512K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
 COMPONENT MODE  COMPONENT MODE  COMPONENT MODE  COMPONENT MODE
 ------------------------------------------------------------
 CSQXMQ4B  OFF   SYSGRS    ON    SYSJES2   SUB   SYSANTOO  MIN
 SYSRRS    MIN   SYSSPI    OFF   SYSJES    SUB   SYSSMS    OFF
 I8I1      SUB   SYSOPS    ON    SYSXCF    ON    SYSLLA    MIN
 SYSXES    ON    SYSAPPC   OFF   SYSTTRC   OFF   SYSTCPDA  SUB
 SYSRSM    OFF   SYSAOM    OFF   SYSVLF    MIN   SYSTCPIP  SUB
 SYSLOGR   MIN   SYSOMVS   MIN   SYSWLM    MIN   SYSTCPIS  SUB
 SYSTCPRE  SUB   SYSIOS    MIN   SYSANTMN  MIN   SYSIEFAL  ON
```

Figure 9-1 MVS Display Trace command output

9.2 GTF trace

Use a GTF trace to show system processing through events occurring in the system over time. The installation controls which events are traced. GTF tracing uses more resources and processor time than a system trace. Use GTF when you are familiar enough with the problem to pinpoint the one or two events required to diagnose your system problem. GTF can be run to an external data set as well as a buffer.

When you activate GTF, it operates as a system task, in its own address space. The only way to activate GTF is to enter a **START GTF** command from a console with master authority. Using this command, the operator selects either IBM's or your cataloged procedure for GTF. The cataloged procedure defines GTF operation; you can accept the defaults that the procedure establishes, or change the defaults by having the operator specify certain parameters on the **START GTF** command.

Because GTF sends messages to a console with master authority, enter the command only on a console that is eligible to be a console with master authority. Otherwise, you cannot view the messages from GTF that verify trace options and other operating information.

IBM supplies the GTF cataloged procedure, which resides in SYS1.PROCLIB. The cataloged procedure defines GTF operation, including storage needed, where output is to go, recovery for GTF, and the trace output data sets. Figure 9-2 shows the format of the IBM-supplied GTF procedure.

```
//GTF PROC MEMBER=GTFPARM
//IEFPROC EXEC PGM=AHLGTF,PARM='MODE=EXT,DEBUG=NO,TIME=YES',
// TIME=1440,REGION=2880K
//IEFRDER DD DSNAME=SYS1.TRACE,UNIT=SYSDA,SPACE=(TRK,20),
// DISP=(NEW,KEEP)
//SYSLIB DD DSN=SYS1.PARMLIB(&MEMBER),DISP=SHR
```

Figure 9-2 GTF Procedure example

The two primary locations that are used to store GTF trace data are external devices (usually meaning a data set on disk), or Internal (INT) storage which is in memory. The benefit of writing to Internal storage is that if the trace is being taken to be reviewed in conjunction with a dump, the GTF in-storage buffers will be dumped along with the address space. You will have trace and dump data taken at the same time and this can be reviewed using IPCS.

If you need to trace for an extended period of time, then writing to an external device is advisable.

9.2.1 Defining the GTF trace options

It is recommended that GTF be started with the following parameters:

`PARM='MODE(INT)'` and `REGION=2880K`

The GTF parameters SADMP, SDUMP, ABDUMP, and BLOK should all be set to at least **10M**.

Figure 9-3 shows the IBM-supplied GTFPARM parmlib member, which contains the GTF trace options.

```
TRACE=SYSM,USR,TRC,DSP,PCI,SRM
```

Figure 9-3 IBM Supplied GTFPARM member

The GTF trace options have the following meanings:

SYSM Selected system events

USR User data that the GTRACE macro passes to GTF

TRC Trace events associated with GTF itself

DSP Dispatchable units of work

PCI Program-controlled I/O interruptions

SRM Trace data associated with the system resource manager (RSM™)

Note: Details regarding other GTF options can be found in *z/OS MVS Diagnosis: Tools and Service Aids*, SY28-1085.

9.2.2 Starting GTF

To invoke GTF, the operator enters the **START** command as follows:

`{START|S}{GTF|membername}.identifier.`

After the operator enters the **START** command, GTF issues message AHL100A or AHL125A to allow the operator either to specify or to change trace options. If the cataloged procedure or **START** command did not contain a member of predefined options, GTF issues message AHL100A so the operator may enter the trace options they want GTF to use. If the procedure or command did include a member of predefined options, GTF identifies those options by issuing the console messages AHL121I and AHL103I. Then you can either accept these options, or reject them and specify new options. Figure 9-4 shows the sequence of messages that appears on the console when starting GTF.

```
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER memname OF PDS dsname
AHL103I TRACE OPTIONS SELECTED -
keywd=(value),...,keywd=(value)
keywd,keywd,...,keywd
AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
```

Figure 9-4 GTF Master Console Reply messages

Figure 9-5 shows the alternative, where the GTF procedure specifies a member that contains the parameters and the resulting messages that are written to the console.

```
START GTF.EXAMPLE1
AHL121I TRACE OPTION INPUT INDICATED FROM MEMBER GTFPARM OF PDS SYS1.PARMLIB
TRACE=SYSM,USR,TRC,DSP,PCI,SRM
AHL103I TRACE OPTIONS SELECTED--SYSM,USR,TRC,DSP,PCI,SRM
*451 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
REPLY 451,U
AHL031I GTF INITIALIZATION COMPLETE
```

Figure 9-5 GTF parameter

The GTFPARM member contained:

```
TRACE=SYSM,USR,TRC,DSP,PCI,SRM
```

9.2.3 Stopping GTF

The operator can enter the **STOP** command at any time during GTF processing. The amount of time you let GTF run depends on your installation and the problem you are trying to capture, but a common time is between 15 and 30 minutes.

To stop GTF processing, have the operator enter the **STOP** command. The **STOP** command must include either the GTF identifier specified in the **START** command, or the device number of the GTF trace data set if you specified MODE=EXT or MODE=DEFER to direct output to a data set.

If you are not sure of the identifier, or the device number of the trace data set, ask the operator to enter the **DISPLAY A,LIST** command. Figure 9-6 shows the result of this command.

```
DISPLAY A,LIST

IEE114I 14.51.49 1996.181 ACTIVITY FRAME LAST F E SYS=SY1
JOBS M/S TS USERS SYSAS INITS ACTIVE/MAX VTAM OAS
00000 00003 00000 00016 00000 00000/00000 00000
LLA LLA LLA NSW S VLF VLF VLF NSW S
JES2 JES2 IEFPROC NSW S
GTF EVENT1 IEFPROC NSW S
```

Figure 9-6 DISPLAY A,LIST command

The operator must enter the **STOP** command at a console with master authority. The general format of the **STOP** command is:

```
{STOP|P} identifier
```

When the **STOP** command takes effect, the system issues message AHL006I. If the system does not issue message AHL006I, then GTF tracing continues, remaining active until a **STOP** command takes effect or the next initial program load (IPL). When this happens, you will not be able to restart GTF tracing. In this case, you can use the **FORCE ARM** command to stop GTF. If there were several functions started with the same identifier on the **START** command, using the same identifier on the **STOP** command will stop all those functions.

9.3 GTF tracing for reason code interrogation

In some instances your software support provider may request you to capture a GTF trace that will contain all the reason codes issued by a particular job. This is more likely if the reason code is not reported externally. If you choose to look at such a GTF trace, be aware that many reason codes are issued validly and do not represent actually errors (that is, reason codes that indicate file not found are usually quite valid).

Prior to setting the following slip you would need to start GTF with options TRACE=SLIP. The slip that would be set is:

```
SLIP SET,IF,A=TRACE,RANGE=(10?+8C?+F0?+1f4?),TRDATA=(13R??+B0,+B3),END
```

After re-creating the problem, stop GTF and format the output using IPCS command **GTFTRACE**.

9.4 Component trace

The component trace service provides a way for MVS components to collect problem data about events. Each component that uses the component trace service has set up its trace in a way that provides the unique data needed for the component.

A component trace provides data about events that occur in the component. The trace data is intended for the IBM Support Center, which can use the trace to:

- Diagnose problems in the component
- See how the component is running

You will typically use component trace while re-creating a problem.

The installation, with advice from the IBM Support Center, controls which events are traced for a system component. GTF does not have to be active to run a component trace.

9.4.1 Parmlib members

There is a table in *z/OS MVS Diagnosis: Tools and Service Aids*, SY28-1085, that shows if a component has a parmlib member. It indicates if the member is a default member needed at system or component initialization, and if the component has default tracing. Some components run default tracing at all times when the component is running; default tracing is usually minimal and covers only unexpected events. Other components run traces only when requested. When preparing your production SYS1.PARMLIB system library, do the following:

- Make sure the parmlib contains all default members identified in the table. If parmlib does not contain the default members at initialization, the system issues messages.
- Make sure that the IBM-supplied CTIITT00 member is in the parmlib. PARM=CTIITT00 can be specified on a **TRACE CT** command for a component trace that does not have a parmlib member; CTIITT00 prevents the system from prompting for a **REPLY** after the **TRACE CT** command. In a sysplex, CTIITT00 is useful to prevent each system from requesting a reply.

An example for a parmlib definition for z/OS UNIX is:

```
CTncccxx     -
CTIBPX00     -    z/OS UNIX parmlib member
                  (which must be specified in BPXPRM00 member)
```

In this example, BPX is the **ccc**, and 00 is the **xx** and I is the **n**. For some components, you need to identify the component's CTncccxx member in another parmlib member. See the parmlib member listed in the default member column in the table in *z/OS MVS Diagnosis: Tools and Service Aids*.

9.4.2 Trace options

If the IBM Support Center requests a trace, the Center might specify the options, if the component trace uses an OPTIONS parameter in its parmlib member or **REPLY** for the **TRACE CT** command.

You must specify all options you would like to have in effect when you start a trace. Options specified for a previous trace of the same component do not continue to be in effect when the trace is started again. If the component has default tracing started at initialization by a parmlib member without an OPTIONS parameter, you can return to the default by doing one of the following:

- Stopping the tracing with a **TRACE CT,OFF** command.
- Specifying OPTIONS() in the **REPLY** for the **TRACE CT** command or in the CTncccxx member.

9.4.3 Collecting trace records

Depending on the component, the potential locations of the trace data are:

- In address-space buffers, which are obtained in a dump
- In data-space buffers, which are obtained in a dump
- In a trace data set or sets, if supported by the component trace

If the trace records of the trace you want to run can be placed in more than one location, you need to select the location. For a component that supports trace data sets, you should choose trace data sets for the following reasons:

- Because you expect a large number of trace records

- To avoid interrupting processing with a dump of the trace data
- To keep the buffer size from limiting the amount of trace data
- To avoid increasing the buffer size

9.4.4 Starting component trace

Select how the operator is to request the trace. The component trace is started by either of the following:

- A **TRACE CT** operator command without a PARM parameter, followed by a reply containing the options
- A **TRACE CT** operator command with a PARM parameter that specifies a CTncccxx parmlib member containing the options

To start a component trace, the operator enters a **TRACE** operator command on the console with MVS master authority. The operator replies with the options that you specified.

```
trace ct,on,comp=sysxcf
* 21 ITT006A ....
r 21,options=(serial,status),end
```

This example requests the same trace using parmlib member CTWXCF03. When **TRACE CT** specifies a parmlib member, the system does not issue message ITT006A.

```
trace ct,on,comp=sysxcf,parm=ctwxcf03
```

9.4.5 Component trace for the logger address space

More subsystems are now using the z/OS System Logger for logging activity that can be used during Unit-of-recovery processing. This data was previously managed by the subsystems (that is, CICS, DB2, MQ, and so forth) but now the Logger Address Space (IXGLOGR) manages the system and subsystem log data. These can reside in a coupling facility, or on DASD.

Problems with Logger process will often require some additional trace data which can be collected as follows:

- Setup the CTRACE for MVS Logger data as follows. Issue the following command to display the current SYSLOGR trace status:

```
D TRACE,COMP=SYSLOGR
```

Figure 9-7 shows the results of the Display Trace command for component SYSLOGR.

```
IEE843I 01.11.36  TRACE DISPLAY 967
        SYSTEM STATUS INFORMATION
 ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,024K)
  COMPONENT     MODE BUFFER HEAD SUBS
 ------------------------------------------------------------
  SYSLOGR       MIN  0002M
     ASIDS      *NONE*
     JOBNAMES   *NOT SUPPORTED*
     OPTIONS    MINIMAL TRACING ONLY
     WRITER     *NONE*
```

Figure 9-7 DISPLAY TRACE,COMP=SYSLOGR output

To update the CTRACE component for the z/OS System Logger edit the SYS1.PARMLIB member CTILOGxx. CTILOG00 is the supplied Logger CTRACE member.

Figure 9-8 shows the CTILOGxx parmlib member and the specified options.

```
TRACEOPTS ON
BUFSIZE(8M)
OPTIONS('CONNECT','DATASET','SERIAL','STORAGE',
'LOGSTRM','MISC','RECOVERY','LOCBUFF')
```

Figure 9-8 CTILOGxx PARMLIB member

This parmlib member will be used when you issue the following command:

```
TRACE CT,COMP=SYSLOGR,PARM=CTILOGxx
```

There is minimal overhead with the MVS Logger CTRACE.

To start the CTRACE for the z/OS Logger and change the trace parameters you can issue:

```
TRACE CT,8M,COMP=SYSLOGR
R xx,OPTIONS=(ALL),END
```

9.5 Master trace

Master trace maintains a table of the system messages that are routed to the hardcopy log. This creates a log of external system activity, while the other traces log internal system activity. Master trace is activated automatically at system initialization, but you can turn it on or off using the **TRACE** command.

Master trace can help you diagnose a problem by providing a log of the most recently issued system messages. For example, master trace output in a dump contains system messages that may be more pertinent to your problem than the usual component messages issued with a dump.

Use the master trace to show the messages to and from the master console. Master trace is useful because it provides a log of the most recently-issued messages. These can be more pertinent to your problem than the messages accompanying the dump itself. Master tracing is usually activated at IPL time and the data can be reviewed with IPCS and is saved when an SVC dump or stand-alone dump is taken.

At initialization, the master scheduler sets up a master trace table of 24 kilobytes. A 24-kilobyte table holds about 336 messages, assuming an average length of 40 characters. You can change the size of the master trace table or specify that no trace table be used by changing the parameters in the SCHEDxx member in SYS1.PARMLIB.

9.5.1 Starting the master trace

You can change the size of the master trace table using the **TRACE** command. For example, to change the trace table size to 500 kilobytes, enter:

```
TRACE MT,500K
```

Start, change, or stop master tracing by entering a **TRACE** operator command from a console with master authority. For example, to start the master tracing:

```
TRACE MT
```

To stop master tracing:

```
TRACE MT,OFF
```

You can also use the **TRACE** command to obtain the current status of the master trace. The system displays the status in message IEE839I. For example, to ask for the status of the trace, enter:

```
TRACE STATUS
```

Figure 9-9 shows a sample of the master trace table. This is an in-storage copy of the system log (SYSLOG) and the amount of data contained in the table is dependant on the size of the table.

```
2003062 03:48:04.21 STC08076 00000090  ITSO1O SYS 1: READY FOR COMMUNICATION
2003062 03:48:33.24 STC04022 00000094 >+CSQX500I =MQU1 CSQXRCTL Channel MQM1.ITSO81O started
2003062 03:49:03.39 STC04022 00000094 >+CSQX202E =MQU1 CSQXRCTL Connection or remote listener
152 00000094 > channel MQM1.ITSO81O,
152 00000094 > connection 9.9.9.90,
152 00000094 > TRPTYPE=TCP RC=00000468
2003062 03:49:03.42 STC04022 00000094 >+CSQX599E =MQM1 CSQXRCTL Channel MQM1.ITSO81O ended
2003062 03:50:01.85 ZZ4NM002 00000294  $RALL,R=*,D=W91A.*,Q=789
2003062 03:50:01.89 ZZ4NM002 00000084  $HASP683 NO JOBS OR DATA SETS REROUTED
```

Figure 9-9 IPCS MTRACE output

9.6 GFS trace

Use GFS trace to collect information about requests for virtual storage via the GETMAIN, FREEMAIN, and STORAGE macros. GTF must be active to run a GFS trace. The following procedure explains how to request a GFS trace.

1. In the DIAGxx parmlib member, set the VSM TRACE GETFREE parameter to *ON* and define the GFS trace control data.

   Example: DIAGxx parmlib member for starting GFS tracing

   The following DIAGxx parmlib member starts GFS trace and limits the trace output to requests to obtain or release virtual storage that is 24 bytes long and resides in address spaces 3, 5, 6, 7, 8, and 9:

   ```
   VSM TRACE GETFREE (ON) ASID (3, 5-9) LENGTH (24) DATA (ALL)
   ```

   Note: If you want the IPCS GTFTRACE output to be formatted, you must include the TYPE and FLAGS data items on the DATA keyword specification of the DIAGxx parmlib member.

   You will need another DIAGxx parmlib member defined to stop GFS tracing specifying:

   ```
   VSM TRACE GETFREE (OFF)
   ```

2. Ask the operator to enter the **SET DIAG=xx** command to activate GFS trace using the definitions in the DIAGxx parmlib member.
3. Start a GTF trace (ask the operator to enter a **START membername** command on the master console). The **membername** is the name of the member that contains the source JCL (either

a cataloged procedure or a job). Tell the operator to specify a user event identifier X'F65' to trace GTF user trace records.

Example: Starting a GTF trace for GFS data

In the following example, the operator starts GTF tracing with cataloged procedure GTFPROC to get GFS data in the GTF trace output. The contents of cataloged procedure GTFPROC are as follows:

```
//GTF     PROC MEMBER=GTFPROC
//* Starts GTF
//IEFPROC EXEC PGM=AHLGTF,REGION=32M,TIME=YES,
//  PARM='MODE=EXT,DEBUG=NO,TIME=YES,BLOK=40K,SD=0K,SA=40K'
//IEFRDER DD DSN=MY.GTF.TRACE,
//           DISP=SHR,UNIT=3390,VOL=SER=VOL001
```

The operator then replies to messages AHL100A with the USRP option. When message AHL101A prompts the operator for the keywords for option USRP, the operator replies with USR=(F65) to get the GFS user trace records in the GTF trace output.

4. To stop the GTF trace, ask the operator to enter a **STOP procname** command on the master console.
5. To stop GFS trace, create a DIAGxx parmlib member with `VSM TRACE GETFREE(OFF)` and have the operator enter a **SET DIAG=xx** command.

GTF places the GFS trace data in a user trace record with event identifier X' F65'. To obtain GFS trace data, do one of the following:

1. When GTF writes the trace data to a data set, format and print the trace data with the IPCS **GTFTRACE** subcommand.
2. When GTF writes trace data only in the GTF address space, use IPCS to see the data in an SVC dump. Request the GTF trace data in the dump through the SDATA=TRT dump option.
3. Issue the IPCS **GTFTRACE** subcommand to format and see the trace in an unformatted dump. Figure 9-10 shows an example of formatted Getmain/Freemain (GFS) trace data.

```
IPCS
GTFTRACE DA('MY.GTF.TRACE')   USR(F65)
IKJ56650I TIME-03:42:20 PM. CPU-00:00:01 SERVICE-52291 SESSION-00:00:20
BLS18122I Initialization in progress for DSNAME(¢ MY.GTF.TRACE¢ )
IKJ56650I TIME-03:42:21 PM. CPU-00:00:01 SERVICE-54062 SESSION-00:00:20
**** GTFTRACE DISPLAY OPTIONS IN EFFECT ****
USR=SEL

**** GTF DATA COLLECTION OPTIONS IN EFFECT: ****
USRP option

**** GTF TRACING ENVIRONMENT ****
Release: SP6.0.6 FMID: HBB6606 System name: CMN
CPU Model: 9672 Version: FF Serial no. 270067

USRDA F65 ASCB 00FA0800 JOBN MYGTF2
Getmain SVC(120) Cond=Yes
Loc=(Below,Below) Bndry=Dblwd
Return address=849CA064 Asid=001A Jobname=MYGTF2
Subpool=229 Key=0 Asid=001A Jobname=MYGTF2 TCB=008DCA70 Retcode=0
Storage address=008D6768 Length=10392 X¢2898¢
GPR Values
0-3 00002898 00000000 7FFFC918 0B601E88
4-7 01FE3240 008FF830 849CA000 00FA0800
8-11 00000000 00000DE8 049CBFFE 849CA000
12-15 049CAFFF 0B601A9C 00FE9500 0000E510
GMT-01/06/1998 21:15:43.111628 LOC-01/06/1998 21:15:43.1

USRDA F65 ASCB 00FA0800 JOBN MYGTF2
Freemain SVC(120) Cond=No
Return address=8B2D608A Asid=001A Jobname=MYGTF2
Subpool=230 Key=0 Asid=001A Jobname=MYGTF2 TCB=008DCA70 Retcode=0
Storage address=7F73DFF8 Length=8 X¢ 8 ¢
GPR Values
0-3 00000000 7F73DFF8 008D82D8 008D7BC0
4-7 008D8958 008D6B08 008D85C8 0B335000
8-11 00000002 00000000 7F73DFF8 008D862C
12-15 8B2D6044 008D8C98 849D242A 0000E603
GMT-01/06/1998 21:15:43.111984 LOC-01/06/1998 21:15:43.1
```

Figure 9-10 Output from IPCS GTFTRACE format

9.7 System trace

Use system trace to see system processing through events occurring in the system over time. System tracing is activated at initialization and, typically, runs continuously. It records many system events, with minimal detail about each. The events traced are predetermined, except for branch tracing. This trace uses fewer resources and is faster than a GTF trace.

System trace tables reside in fixed storage on each processor. The default trace table size is 64 kilobytes per processor, but you can change it using the **TRACE ST** command. We do not recommend running with trace tables smaller than the default 64 kilobytes. You might, however, want to increase the size of the system trace table from the default 64 kilobyte when:

- You find that the system trace does not contain tracing from a long enough time period.

- You want to trace branch instructions (using the BR=ON option on the **TRACE ST** command when you start tracing.)

Because system trace usually runs all the time, it is very useful for problem determination. While system trace and the general trace facility (GTF) lists many of the same system events, system trace also lists events occurring during system initialization, before GTF tracing can be started. System trace also traces branches and cross-memory instructions, which GTF cannot do.

Issue the following command to increase the system trace table size to 256K:

```
TRACE ST,256K
```

System tracing allows you the option of tracing branch instructions, such as BALR, BASR, BASSM, and BAKR, along with other system events.

Note: Running with branch tracing on can affect your system performance and use very large amounts of storage. Do not use branch tracing as the default for system tracing on your system. You should only use it for short periods of time to solve a specific problem. The default system tracing does not include branch instructions.

When you want to trace branch instructions, do the following:

```
TRACE ST,BR=ON
```

System tracing will be captured in all dump situations by default, except during a SNAP dump where SDATA=TRT must be specified. Figure 9-11 shows some sample SYSTRACE data.

```
  01 000A 00AEF430   SVCR   7B 070C0000 868985D2  00000000 00000000 04379238
  01 000A 00AEF430   PGM   011 070C2000 868985F2  00040011 12004000

  01 000A 00AEF430  *RCVY PROG                     940C4000 00000011 00000000

  01 000A 00AEF430   SSRV  12D          813DE814  00AEF430 000C8000 FF3A0000
                                                  00000000
  01 000A 00AEF430   SSRV  12D          813DE830  00AEF430 000B8000 00000000
                                                  00000000
  01 000A 00AEF430   DSP        070C2000 812FADEA  00000000 00FD0E20 12004780
  01 000A 00AEF430  *SVC      D 070C2000 812FADEC  00000000 00FD0E20 12004780
  01 000A 00AEF430   SSRV   78          86A0A4AE  0000FF50 000000C8 00AFB5D8
```

Figure 9-11 IPCS SYSTRACE output

9.8 SMS tracing

If you need to trace the interaction between a data set allocation and SMS, collecting SMS trace data may be of assistance. The procedures to collect and review SMS trace data follow. In this example we are tracing logger address space (IXGLOGR) data set allocation.

1. Issue the following MVS command:

   ```
   SETSMS SIZE(0m)
   ```

2. Issue the following MVS command:

   ```
   SETSMS TRACE(ON),TYPE(ALL),SIZE(1M),DESELECT(ALL),
   SELECT(ALL),JOBNAME(IXGLOGR)
   ```

3. Force Logger to allocate a data set.

4. Turn off the trace by issuing:

   ```
   SETSMS TRACE(OFF)
   ```

5. Make note of the data set name to assist IBM support in locating the trace entries.
6. Take a dump of the SMS address space. For example:

   ```
   DUMP COMM=(any dump title you desire)
   R #,JOBNAME=SMS,CONT
   R #,SDATA=(LPA,CSA,ALLNUC,GRSQ,LSQA,SWA,PSA,SQA,TRT, RGN,SUM)
   ```

7. Invoke IPCS and review the SMS trace by issuing the following IPCS command:

   ```
   VERBX SMSDATA 'TRACE'
   ```

10

Interactive Problem Control System (IPCS)

The most powerful diagnostic tool at your disposal is Interactive Program Control System (IPCS). IPCS is a tool provided in the MVS system to aid in diagnosing software failures. IPCS provides formatting and analysis support for dumps and traces produced by MVS, other program products, and applications that run on MVS.

SVC dumps, stand-alone dumps, and some traces are unformatted and need to be formatted before any analysis can begin. IPCS provides the tools to format dumps and traces in both an online and batch environment. IPCS provides you with commands that will let you interrogate specific components of the operating system and allow you to review storage locations associated with an individual task or control block. IPCS allows you to quickly review and isolate key information that will assist with your problem determination process.

This chapter describes:

- Setting IPCS defaults
- ASIDs to be dumped
- **VERBX MTRACE** command
- IPCS **SUMMARY** command
- IPCS virtual storage commands
- Using IPCS to browse dumps
- Searching IBM problem databases

10.1 Setting the IPCS defaults

Option **0** from the Primary Option Menu enables you to identify the data set that contains the dump you will be analyzing. Figure 10-1 shows the IPCS default option menu.

```
Source ==> DSNAME('SYS1.DUMP01')
Address Space ==> ASID(X'0001')
Message Routing ==> NOPRINT TERMINAL
Message Control ==> FLAG(WARNING) NOCONFIRM VERIFY
Display Content ==> MACHINE REMARK REQUEST STORAGE SYMBOL
```

Figure 10-1 IPCS Default Option panel

The initial display will show Source ==> NODSNAME and no value in Address space. When you enter your dump DSNAME (in single quotes), you must manually change the NODSNAME for DSNAME. Pressing Enter will then update the Address Space field with the primary ASID for the dump.

If the dump was captured via the **DUMP COMM** command, the ASID will always equal x'0001', the MASTER Address space, but the dump data set will also include any address spaces that you requested to be dumped.

You will be able to change the Address Space ASID when you know what ASID dump date you need to review.

After setting the IPCS defaults, return to the IPCS Primary Option menu and select Option **6**, Command. The first IPCS command you enter will start the initialization process for the dump you have specified.

Figure 10-2 shows the messages that are issued during the initialization process.

```
TIME-05:14:53 AM. CPU-00:00:46 SERVICE-673781 SESSION-00:48:42 APRIL 13
BLS18122I Initialization in progress for DSNAME(¢ SYS1.DUMP03¢ )
BLS18124I TITLE=COMPID=DF115,CSECT=IGWLGMOT+1264,DATE=02/18/94,MAINTID= NONE
RC=00000024,RSN=12088C01
BLS18222I ESA mode system
BLS18160D May summary dump data be used by dump access? Enter Y to use, N to
bypass
Y Note. Enter Yes
BLS18123I 4,616 blocks, 19,202,560 bytes, in DSNAME(¢ SYS1.DUMP03¢ )
IKJ56650I TIME-05:15:05 AM. CPU-00:00:46 SERVICE-702725 SESSION-00:48:53 APRIL 13
***
```

Figure 10-2 IPCS Dump initialization messages

10.1.1 Select the IPCS subcommand entry panel

Return to the IPCS Primary Option menu and select option **6**. When you press Enter, the IPCS Subcommand Entry panel is displayed.

Use the IPCS subcommand **STATUS FAILDATA** to locate the specific instruction that failed and to format all the data in an SVC dump related to the software failure. This report gives information about the CSECT involved in the failure, the component identifier, and the PSW address at the time of the error.

Note: For SLIP dumps or CONSOLE dumps, use **SUMMARY FORMAT** or **VERBEXIT LOGDATA** instead of **STATUS FAILDATA**. Any valid IPCS command would have started the initialization process and the related display that results after initialization. It should be noted that the dump is only initialized the first time it is referenced via IPCS, and will only be initialized again if the dump is deleted from the IPCS inventory.

After the initialization process, the address space field in the IPCS Default Values panel will now contain the address space identifier (ASID) information stored in the dump data set SYS1.DUMP00. For example:

```
Address Space ==> ASID(X'009E')
```

Diagnostic report

The IPCS **STATUS FAILDATA** command shows us a diagnostic report that summarizes the failure. Figure 10-3 shows an example of the IPCS STATUS FAILDATA report.

```
* * * DIAGNOSTIC DATA REPORT * * *
SEARCH ARGUMENT ABSTRACT
PIDS/5695DF115 RIDS/IGWLHHLS#L RIDS/IGWLGMOT AB/S00F4 PRCS/00000024
REGS/0E00C REGS/0B225 RIDS/IGWLHERR#R
Symptom Description
------- -----------
PIDS/5695DF115 Program id: 5695DF115
RIDS/IGWLHHLS#L Load module name: IGWLHHLS
RIDS/IGWLGMOT Csect name: IGWLGMOT
AB/S00F4 System abend code: 00F4
PRCS/00000024 Abend reason code: 00000024
REGS/0E00C Register/PSW difference for R0E: 00C
REGS/0B225 Register/PSW difference for R0B: 225
RIDS/IGWLHERR#R Recovery routine csect name: IGWLHERR
OTHER SERVICEABILITY INFORMATION

Recovery Routine Label: IGWFRCSD
Date Assembled: 02/18/94
Module Level: NONE
SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE
Subfunction
Time of Error Information
PSW: 075C2000 82CC5BCC Instruction length: 02 Interrupt code: 000D
Failing instruction text: 41F00024 0A0D5880 D19C5840
```

Figure 10-3 IPCS STATUS FAILDATA output

Note: The STATUS FAILDATA data in this case shows that the load module that was pointed to by the program status word (PSW) was IGWLHHLS, the CSECT within that load module was IGWLGMOT, the abend code (0F4), and the abend reason code (0024). This information is also displayed during the initialization of the dump data set but is not formatted as it is here.

With the information we currently have we could perform a search of the IBM problem databases for a possible solution, but in this instance we will pursue the problem using IPCS to enable you to develop a better understanding of problem analysis techniques.

10.1.2 What ASIDs have been dumped

The **SELECT ALL** command will show what address spaces were *active* when the dump was taken. It *does not show* what address spaces are included in the dump. Figure 10-4 shows an example of the IPCS **SELECT ALL**.

```
ASID JOBNAME  ASCBADDR  SELECTION CRITERIA
---- -------- --------  ------------------
0001 *MASTER* 00FD1480  ALL
0002 PCAUTH   00FBDB80  ALL
0003 RASP     00FBDA00  ALL
0004 TRACE    00FBD880  ALL
0005 DUMPSRV  00FBD700  ALL
0006 XCFAS    00FB4700  ALL
0007 GRS      00FB4580  ALL
0008 SMSPDSE  00FA1480  ALL
0009 SMSVSAM  00FA1300  ALL
000A CONSOLE  00FB4400  ALL
000B WLM      00FB4280  ALL
000C ANTMAIN  00FB4100  ALL
000D ANTASOOO 00FA3780  ALL
000E OMVS     00FAF080  ALL
0010 IEFSCHAS 00FBFE80  ALL
0011 JESXCF   00FBFD00  ALL
0012 ALLOCAS  00FBF300  ALL
0013 IOSAS    00FBF180  ALL
0014 IXGLOGR  00FA3600  ALL
0015 SMF      00FA3480  ALL
007C CMAS     00F43400  ALL
007D CAS      00F43580  ALL
007E EYUX140  00F43280  ALL
0080 MQT1CHIN 00F45700  ALL
0081 MQC1CHIN 00F38500  ALL
0082 NETMOPS  00F57B80  ALL
0084 NETMSNA  00F57880  ALL
0086 IOAOMON1 00F57580  ALL
0087 XCOM     00F57280  ALL
0088 CICSCCTR 00F57100  ALL
0089 DWTSPAS  00F63B80  ALL
008B CICSUA1B 00F4E280  ALL
008C CICSCA3B 00F47A00  ALL
008D CICSTA3B 00F47880  ALL
```

Figure 10-4 IPCS SELECT ALL output

SELECT CURRENT command

The **SELECT CURRENT** command will display the address space that was executing at the time the dump was initiated. If the dump was issued via the console dump command, the **SELECT CURRENT** command will display the Master scheduler address space. Figure 10-5 shows the IPCS **SELECT CURRENT** output.

```
ASID JOBNAME  ASCBADDR  SELECTION CRITERIA
---- -------- --------  ------------------
000A CONSOLE  00FB4400  CURRENT
```

Figure 10-5 IPCS SELECT CURRENT Output

This shows that the CONSOLE ASID was dispatched at the time of the abend.

If the dump was taken while in cross memory mode, both address spaces involved in the cross memory operation will be included in the dump. Figure 10-6 shows the IPCS **SELECT CURRENT** output showing the ASIDs involved in the cross memory function.

```
   ASID JOBNAME ASCBADDR SELECTION CRITERIA
   ---- -------- -------- ------------------
   0033 CICSFILE 00F4E680 CURRENT
   008E CICSJG03 00ED8100 CURRENT
```

Figure 10-6 IPCS SELECT CURRENT cross memory output

Identify address spaces in dump

To identify which address spaces are contained in the dump, you can also use IPCS as follows:

1. Format the CVT (IPCS command **CBF CVT**)
2. Issue a **FIND** command for **RTCT** to locate the address of the Recovery Termination Control Table.
3. At offset +x'10C' in the RTCT begins a list of 1 word entries for the address spaces in the dump. The first half of the word contains the ASID.

Figure 10-7 shows the first step required to find out what address spaces are contained in the dump.

```
cbf rtct;f astb
ASTB

    SDAS  SDF4  SDF5
    ----  ----  ----
001 03EB  F8    00
002 045F  F8    00
003 0445  F8    00
004 008A  F8    00
005 010F  F8    00
006 0000  00    00
007 0000  00    00
```

Figure 10-7 IPCS control block format output of RTCT for the ASTB (SVC DUMP ASID TABLE)

Figure 10-8 shows the result of the following IPCS command where we will use the ASID values returned in the previous format of the RTCT ASTB.

```
ip select asid(x'3eb',x'45f',x'445',x'8a',x'10f')
```

```
ASID JOBNAME  ASCBADDR  SELECTION CRITERIA
---- -------- --------  ------------------
008A MQT1CHIN 00ED8A00  ASID
010F IMSTFAFM 00F0E280  ASID
03EB IMSTCTL  00F60D00  ASID
0445 MQT1MSTR 00F17700  ASID
045F IMSTDLI  00FA2B80  ASID :
```

Figure 10-8 IPCS SELECT ASID command output

The ASIDs and associated JOBNAMEs that are contained in the dump are displayed.

10.2 VERBX MTRACE

The next command we will issue will be the first of our trace commands. Figure 10-9 shows an example of the **VERBX MTRACE** output, which is similar to the SYSLOG output.

```
00000090 IEA995I SYMPTOM DUMP OUTPUT
137 00000090 SYSTEM COMPLETION CODE=0F4 REASON CODE=00000024
137 00000090 TIME=17.21.42 SEQ=00084 CPU=0000 ASID=0008
137 00000090 PSW AT TIME OF ERROR 075C2000 82CC5BCC ILC 2 INTC 0D
137 00000090 NO ACTIVE MODULE FOUND
137 00000090 NAME=UNKNOWN
137 00000090 DATA AT PSW 02CC5BC6 - 41F00024 0A0D5880 D19C5840
137 00000090 GPR 0-3 12088C0C 440F4000 00000008 00000583
137 00000090 GPR 4-7 00FD1060 12088C0C 06BA3998 7F7697C8
137 00000090 GPR 8-11 00FD102C 02CC79A5 02CC69A6 02CC59A7
137 00000090 GPR 12-15 82CC49A8 7F769B48 82CC5BC0 00000024
137 00000090 END OF SYMPTOM DUMP
```

Figure 10-9 IPCS VERBX MTRACE Output

The command displays the following:

- The master trace table entries for the dumped system. This table is a wrap-around data area that holds the most recently issued console messages in a first-in, first-out order.

  The previous MTRACE output shows a small sample of what is contained in the MTRACE. In this sample we see details of the symptom dump for our problem.

  All data that is displayed on the MVS master console will be captured in the master trace table. The amount of data kept is related to the master trace table buffer size, such as:

  – The NIP hard-copy message buffer

  – The branch entry and NIP time messages on the delayed issue queue

10.3 SYSTRACE

The system trace can be examined by issuing the **SYSTRACE** command from the IPCS subcommand entry panel. Issuing the **SYSTRACE** command on its own will display trace entries associated with the dumped ASID only. Issuing the **SYSTRACE ALL** command will display all system trace entries. To display the time field in local time, add the **TIME(LOCAL)** parameter. A complete system trace command is as follows:

```
SYSTRACE ALL TIME(LOCAL)
```

Figure 10-10 shows a small sample of the system trace. The time stamps would appear on the right-hand side of the display but have been removed for presentation reasons.

```
SYSTRACE Example 1 (*SVC)
-------------------------------------------------------------------
CP ASID   TCB      TRACE ID         PSW           R15       R0   R1
--|----|--------|-----------|-----------------|---------|--------|-
00 0008 007FD720  *SVC     D 075C2000 82CC5BCC  00000024 12088C0C
00 0008 007FD720   SSRV   78          828BC3F0  0000FF50 000000C8
                                               00080000
00 0008 007FD720   SSRV   78          828BC41A  0000FF70 00000FB0
                                               00080000
00 0008 007FD720   EXT  1005 070C0000 813B54AC  00001005
-------------------------------------------------------------------
                   SYSTRACE Example 2 (*RCVY)
-------------------------------------------------------------------
00 0153 008DA530   SSRV   78          40E5269C  4050E612 000002B8
                                               01530000
00 0153 008DA530   SSRV   78          80E52704  4050E612 00000080
                                               01530000
02 0013 008C5E88  *RCVY PROG                   940C4000 00000011

02 0013 008C5E88   SSRV   78          8109CADC  4000EF50 00000818
                                               00010000
02 0013 008C5E88  *RCVY FRR  070C0000 9056FBE8  940C4000 00000011
```

Figure 10-10 IPCS SYSTRACE ALL output

10.3.1 Reviewing SYSTRACE data

The SYSTRACE data can be best reviewed by going to the end of the trace output, and issuing a **FIND "*SVC" PREV** command. This should help you locate the trace entry that indicates the abend. Another useful trace point to search for is ***RCVY,** which indicates a recovery action. Entries prior to this can assist with problem diagnosis. An SVC D is the abend SVC. Note the PSW, which is the same as identified in previous steps. This will point to the next instruction to be processed.

The SVC trace entries are as follows:

- An SVC trace entry is for processing of a Supervisor Call (SVC) instruction.
- An SVCE trace entry is for an error during processing of an SVC instruction.
- An SVCR trace entry is for return from SVC instruction processing.

The actual SVC identified in the SYSTRACE is the hexadecimal identification. This must be converted to decimal to enable the correct research, for example:

- The SYSTRACE entry for SVC 78 would convert to a decimal SVC number of 120, which, when referencing *z/OS MVS Diagnosis Reference*, SY28-1084, would identify the GETMAIN/FREEMAIN SVC.

This is an example of just one of the many trace entries that are created during the life of a z/OS task. For a further explanation of other trace entries, refer to *z/OS Diagnosis: Tools and Service Aids*, SY28-1085.

10.4 IPCS SUMMARY command

Use the **SUMMARY** subcommand to display or print dump data associated with one or more specified address spaces.

SUMMARY produces different diagnostic reports depending on the report type parameter, FORMAT, KEYFIELD, JOBSUMMARY, and TCBSUMMARY, and the address space selection parameters ALL, CURRENT, ERROR, TCBERROR, ASIDLIST, and JOBLIST. Specify different parameters to selectively display the information you want to see.

Note: Installation exit routines can be invoked at the system, address space, and task level for each of the parameters in the **SUMMARY** subcommand.

The **SUMMARY FORMAT** command displays task control block (TCB) and other control block information. By issuing the **MAX DOWN**, or **M** PF8 command the TCB summary will be located.

The TCB summary can be located at the end of an IPCS summary format report as shown in the following example. By reviewing the data in the CMP field, we see that TCB 007FD588 has a non-zero CMP field that reflects the 44**0F4**4000 abend. Figure 10-11 shows the TCB Summary.

```
* * * * T C B S U M M A R Y * *
JOB SMXC ASID 0008 ASCB 00FBC580 FWDP 00FBC400 BWDP 00F4E600 PAGE
TCB AT CMP NTC OTC LTC TCB BACK PAGE
007FE240 00000000 00000000 00000000 007FDE88 007FF1D8 00000000 000014
007FF1D8 00000000 00000000 007FE240 00000000 007FDE88 007FE240 000018
007FDE88 00000000 007FF1D8 007FE240 007FD588 007FDB70 007FF1D8 000021
007FDB70 00000000 00000000 007FDE88 00000000 007FD588 007FDE88 000024
007FD588 440F4000 02000000 00000000 00000000 00000000 007FBFB8 000026
```

Figure 10-11 TCB Summary at the bottom of the SUMMARY FORMAT display

By issuing a **Find "TCB: 007FD588" prev** command, we will be taken to the failing TCB data in the Summary Format display. From this point we want to locate the RTM2WA area. This can contain information that in many cases identifies the failing program.

The PSW is still the major reference point. Issue the following command from the command line:

```
IP WHERE 02CC5BCC.
```

You can see in which load module the failure occurred and where that module was located as follows:

```
ASID(X¢0008¢) 02CC5BCC. IGWLHHLS+02DBCC IN EXTENDED PLPA
```

10.5 What is VERBX?

One of the more common IPCS commands is **VERBX**. This stands for VERB Exit and indicates that a product-specific exit routine will be used to format the dump. For example, to format dump data for CICS/TS Release 1.3 we would use the exit routine, DFHPD530. This program is supplied with CICS/TS Release 1.3 to enable you to format the CICS/TS-specific data.

For example, the commands could be used as follows:

- Format the CICS Dispatcher data contained in the dump.

```
VERBX DFHPD640 'DS=1'
```

- Format the IMS savearea.

```
VERBX IMSDUMP 'imsjobname FMTIMS savearea'
```

- Format the DB2 thread data.

```
VERBX DSNWDMP 'sumdump=no,subsys=itso,ds=1'
```

10.5.1 IPCS VERBX LOGDATA command

Specify the LOGDATA verb name on the **VERBEXIT** subcommand to format the logrec buffer records that were in storage when the dump was generated. LOGDATA locates the logrec records in the logrec recording buffer and invokes the EREP program to format and print the logrec records. The records are formatted as an EREP detail edit report.

Use the LOGDATA report to examine the system errors that occurred just before the error that caused the dump to be requested.

Another valuable source of diagnostic information in the dump is the system error log entries which are created for all hardware and software error conditions. To review these records the **VERBX LOGDATA** command can be used and the last records should relate to the abend. This is not always the case, but reviewing this data from the last entry and moving backwards in time can often present information that relates to the problem or may indicate what the cause was. This may indicate a hardware or software error. In our case the logdata does include records for our problem and is representative of data already found. Figure 10-12 shows the start of the last error log entry displayed.

```
TYPE: SOFTWARE RECORD REPORT: SOFTWARE EDIT REPORT DAY.
(SVC 13) REPORT DATE: 103.99
FORMATTED BY: IEAVTFDE HBB6601 ERROR DATE: 103.99
MODEL: 9021 HH:MM:SS
SERIAL: 060143 TIME: 17:21.42
JOBNAME: MSTJCL00 SYSTEM NAME:
ERRORID: SEQ=00080 CPU=0000 ASID=0008 TIME=17:21:42.3
SEARCH ARGUMENT ABSTRACT
PIDS/5695DF115 RIDS/IGWLHHLS#L RIDS/IGWLGMOT AB/S00F4 PRCS/00000024
REGS/0C7E8 RIDS/IGWLHERR#R
SYMPTOM DESCRIPTION
------- -----------
PIDS/5695DF115 PROGRAM ID: 5695DF115
RIDS/IGWLHHLS#L LOAD MODULE NAME: IGWLHHLS
RIDS/IGWLGMOT CSECT NAME: IGWLGMOT
AB/S00F4 SYSTEM ABEND CODE: 00F4
PRCS/00000024 ABEND REASON CODE: 00000024
REGS/0E00C REGISTER/PSW DIFFERENCE FOR R0E: 00C
REGS/0C7E8 REGISTER/PSW DIFFERENCE FOR R0C: 7E8
RIDS/IGWLHERR#R RECOVERY ROUTINE CSECT NAME: IGWLHERR
OTHER SERVICEABILITY INFORMATION
RECOVERY ROUTINE LABEL: IGWFRCSD
DATE ASSEMBLED: 02/18/94
MODULE LEVEL: NONE
SERVICEABILITY INFORMATION NOT PROVIDED BY THE RECOVERY ROUTINE
SUBFUNCTION
TIME OF ERROR INFORMATION
PSW: 075C2000 82CC5190 INSTRUCTION LENGTH: 02 INTERRUPT CODE: 000D
FAILING INSTRUCTION TEXT: 41F00024 0A0DBF0F D1D44780
```

Figure 10-12 VERBX LOGDATA output

System error log

The system error log can also be interrogated via a batch utility. The program used to extract this data from either the online error log data set, SYS1.LOGREC, or a historical error log data set is IFCEREP1. This program can be used to produce hardware and software failure reports in both a summary and detailed format. Figure 10-13 shows the JCL required to process a software summary report.

```
/jobcard
//*
//STEP1 EXEC PGM=IFCEREP1,PARM=CARD
//SYSPRINT DD SYSOUT=*
//SERLOG DD DSN=SYS1.LOGREC,DISP=SHR
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//EREPPT DD SYSOUT=(*,DCB=BLKSIZE=133)
//TOURIST DD SYSOUT=(*,DCB=BLKSIZE=133)
//SYSIN DD *
PRINT=PS
TYPE=SIE
ACC=N
TABSIZE=512K
ENDPARM
//
```

Figure 10-13 IFCEREP1 sample JCL

Logrec data

If your LOGREC data is stored in a Coupling Facility log stream data set you can use the IFCEREP1 program to access this. Figure 10-14 shows the JCL that will enable you to produce error log reports from the log stream data set.

```
//jobcard
//*
//EREPLOG EXEC PGM=IFCEREP1,REGION=4M,
// PARM=(¢ HIST,ACC=N,TABSIZE=512K,PRINT=PS,TYPE=SIE¢ )
//ACCIN DD DSN=sysplex.LOGREC.ALLRECS,
// DISP=SHR,
// SUBSYS=(LOGR,IFBSEXIT,¢ FROM=(1999/125),TO=YOUNGEST¢ ,
// ¢ SYSTEM=SC42¢ ) ,
// DCB=(RECFM=VB,BLKSIZE=4000)
//DIRECTWK DD UNIT=SYSDA,SPACE=(CYL,5,,CONTIG)
//TOURIST DD SYSOUT=*,DCB=BLKSIZE=133
//EREPPT DD SYSOUT=*,DCB=BLKSIZE=133
//SYSABEND DD SYSOUT=*
//SYSIN DD DUMMY
```

Figure 10-14 IFCEREP1 JCL to format Coupling Facility LOGREC data

Error log reports

When generating error log reports from log stream data, remember that the log stream data set contains error information for all systems in the sysplex connected to the Coupling Facility. You should use the *SYSTEM* option of the SUBSYS parameter to filter the log stream records. Date and time parameters will also assist with the filtering.

Other information is included in the error log information in the component information. This can assist with isolating the specific product that is being affected and the maintenance level of the module that detected the failure. The maintenance level or service release level is also know as the PTF level, or you might be requested for the replacement modification identifier (RMID). It should be noted that the maintenance level of the failing load module is not necessarily the maintenance level of the failing CSECT, or module, within the load module.

Figure 10-15 shows some of the component data that can be located in the system error log.

```
COMPONENT INFORMATION:
COMPONENT ID: 5695DF115
COMPONENT RELEASE LEVEL: 1B0
PID NUMBER: 5695DF1
PID RELEASE LEVEL: V1R2
SERVICE RELEASE LEVEL: UW04733
DESCRIPTION OF FUNCTION: PDSE LATCH SUPPORT
PROBLEM ID: IGW00000
SUBSYSTEM ID: SMS
```

Figure 10-15 LOGREC Error Component Data

10.6 IPCS virtual storage commands

Interrogating Virtual Storage usage in a dump is achieved by using the IPCS **VERBX VSMDATA** command. Some examples of this command are:

```
VERBX VSMDATA 'NOG SUMMARY'
VERBX VSMDATA 'OWNCOMM'      (Check Common Storage Tracking)
VERBX VSMDATA 'OWNCOMM DETAIL ALL SORTBY(ASIDADDR)'
VERBX VSMDATA 'OWNCOMM DETAIL ASID(ddd) SORTBY(TIME)'
VERBX VSMDATA 'NOGLOBAL,JOBNAME(xxxxDBM1)'
```

The end of the VSMDATA NOG SUMMARY display has an interesting summary that can be very helpful for assisting with S878/80A abends. Figure 10-16 on page 89 and Figure 10-17 on page 90 show a sample of the data displayed for the Virtual Storage Manager.

```
L O C A L   S T O R A G E   D A T A   S U M M A R Y


LOCAL STORAGE MAP
_____________________________
|                            |80000000  <- TOP OF EXT. PRIVATE
| Extended                   |
| LSQA/SWA/229/230           |80000000  <- MAX EXT. USER REGION ADDRESS
|____________________________|7EB8E000  <- ELSQA BOTTOM
|                            |
| (Free Extended Storage)    |
|____________________________|30C77000  <- EXT. USER REGION TOP
|                            |
| Extended User Region       |
|____________________________|2AF00000  <- EXT. USER REGION START
:                            :
: Extended Global Storage    :
=======================================<- 16M LINE
: Global Storage             :
:____________________________:  A00000  <- TOP OF PRIVATE
|                            |
| LSQA/SWA/229/230           |
|____________________________|  9B3000  <- LSQA BOTTOM
|                            |
| (Free Storage)             |  986000  <- MAX USER REGION ADDRESS
|____________________________|  564000  <- USER REGION TOP
|                            |
| User Region                |
|____________________________|    6000  <- USER REGION START
: System Storage             :
:____________________________:       0
Input Specifications:

    Region Requested         =>              0
    IEFUSI/SMF Specification => SMFL :   980000  SMFEL: 79E00000
                                SMFR :   880000  SMFER: 79800000
    Actual Limit             => LIMIT:   980000  ELIM : 55100000

Summary of Key Information from LDA (Local Data Area) :

STRTA  =     6000  (ADDRESS of start of private storage area)
SIZA   =   9FA000  (SIZE of private storage area)
CRGTP  =   564000  (ADDRESS of current top of user region)
LIMIT  =   980000  (Maximum SIZE of user region)
LOAL   =   54F000  (TOTAL bytes allocated to user region)
HIAL   =    4D000  (TOTAL bytes allocated to LSQA/SWA/229/230 region)
SMFL   =   980000  (IEFUSI specification of LIMIT)
SMFR   =   880000  (IEFUSI specification of VVRG)

ESTRA  = 2AF00000  (ADDRESS of start of extended private storage area)
ESIZA  = 55100000  (SIZE of extended private storage area)
ERGTP  = 30C77000  (ADDRESS of current top of extended user region)
ELIM   = 55100000  (Maximum SIZE of extended user region)
ELOAL  =  5CE7000  (TOTAL bytes allocated to extended user region)
EHIAL  =   C07000  (TOTAL bytes allocated to extended LSQA/SWA/229/230)
SMFEL  = 79E00000  (IEFUSI specification of ELIM)
SMFER  = 79800000  (IEFUSI specification of EVVRG)
```

Figure 10-16 VERBX VSMDATA storage map output

This is followed by a SUBPOOL storage usage summary for each TCB. An example is shown in Figure 10-17.

```
LOCAL SUBPOOL USAGE SUMMARY

     TCB/OWNER   SP#   KEY      BELOW    ABOVE    TOTAL
     ---------   ---   ---      -----    -----    -----

       9FF410    129    0      340000  3100000  3440000
       9FF410    130    8      100000  1200000  1300000
       9FF410    130    9       80000   200000   280000
       9FF410    131    8        4000   62B000   62F000
       9FF410    132    4           0    1E000    1E000
       9FF410    132    8        C000    86000    92000
        LSQA     205    0           0    A3000    A3000
        LSQA     215    0           0    19000    19000
        LSQA     225    0           0    15000    15000
       9FFE88    229    0           0     D000     D000
       9FFBF8    229    0           0    1C000    1C000
       9FF5A8    229    0        1000     2000     3000
       9FF410    229    0           0     1000     1000
       9FF180    229    0           0     1000     1000
       9FB280    229    0           0     8000     8000
       9ECE88    229    0           0     1000     1000
       9ECAD8    229    0        3000     9000     C000
```

Figure 10-17 VERBX VSMDATA Subpool Usage Summary

10.7 Using IPCS to browse storage

Another function of IPCS is the ability to browse storage locations with the dump. Although you have identified the failing instruction text and PSW in many of the displays you have reviewed, there will be many times when you will need to look at storage locations. This is achieved by selecting the **BROWSE** option from the IPCS primary option menu. The next panel will identify the current dump data set, and you can change this if necessary. You will usually enter this function after reviewing other information in the dump, so the requirement to change the dump data set will be unlikely.

Figure 10-18 shows an example of the IPCS browse panel.

```
DSNAME('SYS1.DUMP03') POINTERS -----------------------------------
Command ===> SCROLL ===CSR
ASID(X'0008')  is the default address space
PTR Address Address space Data type
00001 00000000 ASID(X'0008') AREA
```

Figure 10-18 IPCS BROWSE storage panel

From this panel you can overtype the *address* field with the address specified in the PSW, and **S**elect in the PTR field. Figure 10-19 shows the Select being performed for the storage address.This shows the browse panel with **S** in the PTR field and the address **02CC5BCA** specified, which is the PSW address minus the instruction length (2).

```
DSNAME('SYS1.DUMP03') POINTERS -----------------------------------
Command ===> SCROLL ===CSR
ASID(X'0008')  is the default address space
PTR Address Address space Data type
S0001 02CC5BCA ASID(X'0008') AREA
```

*Figure 10-19 IPCS BROWSE Storage **S**ELECT option*

Figure 10-20 shows the storage at location 02CC5BCA, which identifies the failing instruction as 0A0D. Instruction 0A is the supervisor call, or SVC instruction. 0D identifies the SVC number, which in this case is 13. (The hexadecimal value 0D is equal to 13 decimal.)

```
ASID(X¢0008¢)  STORAGE ------------------------------
Command ===>
02CC5BCA 0A0D 5880D19C
02CC5BD0 5840803C D213D110 AF325040 D11C4150
02CC5BE0 D1985050 D1244180 D1F45080 D12858F0
02CC5BF0 932B4110 D11005EF 585092BF 18054110
```

Figure 10-20 IPCS BROWSE Selected Storage

The STATUS FAILDATA display and IPCS VERBX LOGDATA display we reviewed previously both had a line displayed which identified the failing instruction text: 41F00024 0A0D5880 D19C5840. When compared to the storage displayed previously for location 02CC5BCA you will find it to be the same value, 0A0D5880 D19C5840.

SVC 13 is the abend SVC, and in this example we can see that the program itself has taken the abend, as coded. An error condition has occurred that has caused a branch to this abend instruction.

Scrolling backwards through the dump, and reviewing the data presented on the right-hand side of the screen leads us to the start of the module, and also shown will be the PTF level of that module, as well as the link-edit date. This can be useful when searching the problem databases or reporting the problem to IBM. The different types of SVCs are documented in *z/OS MVS Diagnosis Reference*, SY28-1084.

Figure 10-21 shows the storage as we scroll back through the dump and look for the start of a module.

```
02CC4A60 00504AFF 000407FF B20A0000 D203D048 | .&¢.........K.} |
02CC4A70 1000B20A 005047F0 C0F420C9 C7E6D3C7 | .....&.0{4.IGWLG|
02CC4A80 D4D6E3F0 F261F1F8 61F9F4C8 C4D7F4F4 | MOT02/18/94HDP44|
02CC4A90 F1F040E4 E6F0F3F3 F8F94000 183D4180 | 10 UW03389 .....|
02CC4AA0 00501F38 58203018 58702000 5070D1A8 | .&.........&.J |
02CC4AB0 5880702C D207D280 81105820 7040D207 | ....K.K.a.... K |
```

Figure 10-21 Dump storage eyecatcher data

10.8 Using IPCS to find the failing instruction

Here are the quick steps to find the failing instruction. Most of the time this works well, but there are some variations that cause this to fail, for example, no RTM2WA.

1. Format the dump using the **SUMM FORMAT REGS** command.

2. Max PF8 to the bottom of this display.
3. Find the TCB with a non-zero CMP (completion) value. This often relates to the abend, for example, 0C4.
4. Issue the **`FIND 'TCB: xxxxxxxx' PREV`** where xxxxxxxx is the TCB identified in STEP 2.
5. Scroll down from the start of the TCB data and find the **RTM2WA** (Recovery Termination Manager Work Area).
6. Find the PSW that is in the RTM2WA for that TCB. Also note the ILC (Instruction Length Code) that indicates the length of the instruction being executed.
7. Use the **IPCS** Browse function to look at the storage pointed to by the PSW.
8. In some cases the PSW will point to the failing instructions; in most cases, though, it will point to the next instruction to be executed. Subtract the ILC from the PSW and this should be the failing instruction address.
9. Scrolling up from this point should also lead you to the module header for the failing program.

Will knowing the failing assembler instruction help diagnose a user application problem if you do not have the program source and compile/lked listings? Usually no, but it is a nice thing to know when you are looking for similar reported problems and when dealing with IBM code defects may assist you.

Naturally, as is often the case, the abend symptom data usually includes the module and offset and displays the data at the PSW location, but it is nice to know you can check this quickly using IPCS.

10.9 Searching IBM problem databases

At this point we have evaluated the available diagnostic data. Look in *z/OS MVS Systems Codes*, GC28-1780 to find the meaning of the 0F4 abend. Figure 10-22 shows the explanation for the 0F4 abend.

```
Explanation:  An error occurred in DFSMSdfp support.
Source: DFSMSdfp
System Action:  Prior to the ABEND error occurring, a return code was
placed in the general register 15 and a reason code in general register
0. An SVC dump has been taken unless the failure is in IGWSPZAP
where register 15 contains 10. The DFSMSdfp recovery routines retry to
allow the failing module to return to its caller.
See DFSMS/MVS DFSMSdfp Diagnosis Guide for return code information.
Programmer Response:  System error. Rerun the job.
System Programmer Response:  If the error recurs and the program is not
in error, search problem reporting data bases for a fix for the
problem. If no fix exists, contact the IBM Support Center.
Provide the JCL, the SYSOUT output for the job, and the logrec data set
error record.
```

Figure 10-22 Documented abend S0F4 Information

Note: The *z/OS MVS System Messages and Systems Codes* manuals should always be your first reference point. These manuals contain important information that could solve your problem and give you suggestions regarding the possible causes and resolutions.

Figure 10-23 displays what we know of the abend details.

```
LOAD MODULE NAME: IGWLHHLS   - Maintenance level UW04733
    CSECT NAME:       IGWLGMOT   - Maintenance level UW03389
    SYSTEM ABEND CODE: 00F4
    ABEND REASON CODE: 00000024
    RSN=12088C01
```

Figure 10-23 Abend details

This information can be used to build the IBM problem database search arguments. The search arguments should use the following formats:

Abend: The format should be ABEND*xxx* or ABENDS*xxx* where *xxx* is the abend code.

Messages The format should be MSG*xxxxxxx* where *xxxxxxx* is the message code.

Return and Reason Codes The format should be RC*xx* where *xx* is the reason or return code. A reason code alternative is:

Reason Codes The format can be RSN*xxxxx*, where *xxxxx* is the reason code.

These are the recommended formats to be used when querying the problem database or reporting problems. These are not the only formats that are used, and some creativity and imagination can assist with expanding your search. These search arguments are also called a symptom string. If the problem being diagnosed was already reported and the symptoms entered in the database, the search will produce a match.

11

CICS problem diagnosis

This chapter describes problem diagnosis for CICS. CICS, CICSPlex/SM, DB2, IMS, Language Environment, and MQ are just some of the products can be analyzed using IPCS.

We provide you with some diagnostic processes that can assist you with problem determination for particular products; you can then use this information as a starting point to apply the techniques in your particular environment.

This chapter describes:

- CICS messages
- CICS abend codes
- Analyzing CICS SVC dumps
- CICS VERBEXIT options
- CICS internal trace
- CICS trace control facility

11.1 Problem reference points

The first reference points for all problems are:

- System log
- Job log
- System error log
- Message and codes manuals

The following sections provide you with some fundamental CICS problem determination techniques.

While a dump will be automatically captured is most circumstances when CICS detects a serious error, dumps may be suppressed by DAE, or if they have not been included in the CICS dump table. For example, CICS Short-On-Storage (SOS) conditions generate DFHSM0131or DFHSM0133 messages, but do not trigger dumps. If these conditions persist and a dump is required, you need to add the SM0131 or SM0133 code to the CICS dump table.

To add a specific error or abend code to the CICS dump table the following commands can be used. For errors that are denoted by a DFHxxxxxx message, you can issue:

```
CEMT SET SYDUMPCODE(xxxxxx) ADD SYS MAX(nn)
```

where xxxxxx is a CICS error message suffix, for example, SM0102. You can use the MAX parameter to limit the number of dumps captured.

A transaction abend code, for example AKEB, could be added to the CICS dump table as follows, and while a transaction abend would usually generate a transaction dump, a system or SVC dump is really required to enable support teams to perform efficient analysis. To add this abend code to the dump table and request a system (SVC) dump to be captured, the following can be used:

```
CEMT SET TRDUMPCODE(AKEB) ADD SYS MAX(nn)
```

When you set a SLIP, or issue a **DUMP COMM** command to capture a dump, it is important to understand the topology of the CICS environment that is having the problem, and to ensure that the related TORs, AORs, and FORs are dumped at the same time if the problem appears to be a cross-region issue.

11.2 CICS messages

CICS messages consist of the prefix DFH followed by a two-letter component identifier (cc), and a four-digit message number (nnnn). The component identifier shows the domain or the component which issues the message. Some examples of the component identifier are:

- AP - The application domain
- DS - The dispatcher domain
- SM - The storage manager domain
- XM - The transaction manager
- XS - The CICS security component

Thus, the CICS message DFHAP0002 is issued from the application domain, identified by the two-character identifier AP.

11.3 CICS abend codes

An abend code indicates the cause of an error that may have been originated by CICS or by a user program. For most of the abend codes described, a CICS transaction dump is provided at abnormal termination.

All CICS transaction abend codes are four-character alphanumeric codes of the form *Axxy*, where:

xx Is the two-character code assigned by CICS to identify the module that detected an error.

y Is the one-character alphanumeric code assigned by CICS.

Figure 11-1 shows the description of the CICS transaction abend code ASRA.

```
ASRA
Explanation:  The task has terminated abnormally because of a program
check.
System Action:  The task is abnormally terminated and CICS issues either
message DFHAP0001 or DFHSR0001.  Message DFHSR0622 may also be issued.

User Response:  Refer to the description of the associated message or
messages to determine and correct the cause of the program check.
```

Figure 11-1 CICS transaction abend example

Each release of CICS and CICS/Transaction Server have their own IPCS formatting load modules. Figure 11-2 shows the naming convention for these load modules is DFHPD*xxx*, where *xxx* identifies the CICS release.

```
CICS/TS 1.3 - DFHPD530
CICS/TS 2.2 - DFHPD620
CICS/TS 2.3 - DFHPD630
CICS/TS 3.1 - DFHPD640
```

Figure 11-2 CICS/TS dump formatting modules

Most commands associated with CICS problem diagnosis using IPCS will be entered from the IPCS Subcommand Entry panel, which is Option **6 Command** on the IPCS Primary Option Menu. These commands will be prefixed by `VERBX` and the relevant CICS module `DFHPDxxx`, followed by the parameter, for example:

```
VERBX DFHPD530 parameter
```

Many parameters have several levels of complexity. For example, level **1** might be a summary, level **2** might be an expanded view, and level **3** may contain both level 1 and 2 data.

11.4 Analyzing CICS SVC dumps

The best place to start when analyzing a CICS SVC dump is with the kernel error stack. This stores information about the last 50 errors that have occurred within the CICS subsystem. This table will only hold 50 errors, so if you have a system that is generating excessive errors, that in turn percolate, this can fill up, and the oldest entries will be discarded. In the majority of

cases this should not be a problem, as the kernel error stack will hopefully hold few error identifiers and their related diagnostic information.

The kernel stack entries contain information that has been saved by the kernel on behalf of programs and subroutines on the kernel linkage stack. If you refer the problem to the IBM Support Center, they might need to use the stack entries to find the cause of the failure.

Figure 11-3 shows an example of the first section of the kernel error stack display, which summarizes the error condition that caused the dump to be taken.

```
KE=1
=== SUMMARY OF ACTIVE ADDRESS SPACES
ASID(hex): JOBNAME:
0282 MYCICS01
-- DFHPD0121I FORMATTING CONTROL BLOCKS FOR JOB CICSSA05
DUMPID: 1/0005
DUMPCODE: XM0002
DATE/TIME:18/02/99 09:27:51 (LOCAL)
MESSAGE: DFHXM0002 CICSSA05 A severe error (code X'100A')   has occured
in DFHXMIQ
SYMPTOMS: PIDS/565501800 LVLS/530 MS/DFHXM0002 RIDS/DFHXMIQ PTFS/ESA530
TITLE: (None)
CALLER: (None)
ASID: X'0111'
```

Figure 11-3 CICS kernel error domain output

Figure 11-4 shows the next section of the kernel error stack display, which identifies all tasks active in the CICS environment at the time the dump was taken, and more importantly it identifies the ***Running*** task, at the time of the abend.

```
KE_NUM KE_TASK  STATUS        TCA_ADDR TRAN_# TRANSID DS_TASK  KE_KTCB
0001   16D32C80 KTCB Step     00000000                00000000 16D6C020
0002   16D32900 KTCB QR       00000000                1733B000 16D6F008
0003   16D32580 KTCB RO       00000000                1733D000 16D6E010
0004   16D32200 KTCB FO       00000000                1733F000 16D6D018
0005   16D4BC80 Not Running   00000000                1734A080 16D6E010
0006   16D4B900 Not Running   17485680 00037  CSHQ    1734A280 16D6F008
0007   16D4B580 KTCB          00000000                17374000 1738D008
0008   16D4B200 Not Running   00000000                173E8080 16D6F008
0009   16D64C80 KTCB 00000000                         1738E000 17391008
000A   174D9080 Unused
000C   174D9400 Unused
000E   17C5B080 Not Running   00054680 00006  CSSY    17389580 16D6F008
.
.
0050   17BFF080 ***Running**  00056680 00128  ABC1    17396780 16D6F008
0051   17BFF400 Unused
0052   17BFF780 Unused
0053   17BFFB00 Unused
0054   17C3C080 Not Running   17486680 00066  ABC2    17396880 16D6F008
0055   17C3C400 Not Running   17484680 00029  COIE    17396280 16D6F008
0056   17C3C780 Not Running   17484080 00022  COIO    17396180 16D6F008
0057   17C3CB00 Not Running   17483680 00020  CONL    17396080 16D6F008
005A   17C9D780 Not Running   00055680 00019  CSNC    1734A680 16D6F008
```

Figure 11-4 CICS kernel error stack data

The CICS kernel domain - level 1 - task summary is an abbreviated version of a CICS task summary list. The most important thing to look for is the ***Running*** task. This is *most likely* the problem task.

The ***Running*** task is transaction ABC1, which has TRAN_# *00128* and KE_NUM *0050*.

As you scroll down (*PF8*) the display the next section shows the CICS processing flow of each task. Scroll down until you find the KE_NUM associated with the tasks identified in the previous step. Figure 11-5 shows the transaction-specific processing flow in the KE domain.

```
KE_NUM @STACK     LEN TYPE  ADDRESS  LINK  REG  OFFS ERROR     NAME
0050   17C24020  0120 Bot  96C01180 96C01458    02D8           DFHKETA
0050   17C24140  0230 Dom  96C11548 96C11640    00F8           DFHDSKE
0050   17C24370  0430 Dom  96C35D78 96C38BC2    2E4A           DFHXMTA
                       Int     +2D8A 96C3765A    18E2           RMXM_BACKOUT_TRAN
0050   17C247A0  0780 Dom  96CA06E0 96CA18DE    11FE           DFHRMUW
                       Int     +0862 96CA07E4    0104           RMUW_BACKOUT_UOW_
0050   17C24F20  02E0 Dom  97A30150 97A30352    0202           DFHLTRC
0050   17C25200  0260 Dom  97A31020 97A31510    04F0           DFHTFRF
                       Int     +0418 97A31166    0146           RELEASE_FACILITY
                       Int     +0488 97A31466    0446           FREE_TERMINAL
                       Int     +04B6 97A314CC    04AC           TC_FREE_DETACH
0050   17C25460  0490 Lifo 176E3D18 976E6496    277E           DFHZISP
0050   001090A0  0228 Lifo 177C00F0 977C06EC    05FC           DFHZEMW
0050   17C258F0  0DE0 Lifo 177705D0 97770724    0154           DFHMGP
0050   17C266D0  0400 Dom  96C3AF90 96C3D21A    228A           DFHXMIQ
                       Int     +0CE2 96C3B084    00F4           INQUIRE_PARAMETER
                       Int     +21AA 96C3C182    11F2           SEVERE_ERROR
0050   17C26AD0  0FC0 Dom  96C64FF0 96C6895A    396A           DFHMEME
                       Int     +2EE0 96C6516A    017A           SEND
                       Int     +156C 96C67FBE    2FCE           CONTINUE_SEND
                       Int     +3892 96C665EE    15FE           TAKE_A_DUMP_FOR_C
0050   17C27A90  04C0 Dom  96CE6730 96CE7DE4    16B4           DFHDUDU
                       Int     +0B5C 96CE6822    00F2           SYSTEM_DUMP
                       Int     +19F8 96CE778E    105E           TAKE_SYSTEM_DUMP
```

Figure 11-5 Kernel error data associated with the failing transaction

The CICS kernel domain task summary - task flow KE_NUM=0050 shows information about the processing flow of transaction ABC1. When the failure occurred one of the following was indicated:

- INQUIRE_PARAMETER
- SEVERE_ERROR
- DFHMEME
- SEND
- CONTINUE_SEND
- TAKE_A_DUMP_FOR_CALLER
- DFHDUDU
- SYSTEM_DUMP
- TAKE_SYSTEM_DUMP

Often the ERROR column will indicate a failure with YES next to the failure. In this case it does not, but the SEVERE_ERROR following the INQUIRE_PARAMETER points us in the right direction.

Figure 11-6 displays the end of the KE=1 format and shows all errors in the kernel error stack, with a maximum of 50 errors records being kept.

```
==KE: KE Domain Error Table Summary
ERR_NUM  ERR_TIME KE_NUM ERROR TYPE           ERR_CODE MODULE   OFFSET
=======  ======== ====== ==========           ======== ======   ======
00000001 09:19:47 0051   PROGRAM_CHECK        0C7/AKEA DFHYC520 000D7A
00000002 09:19:47 0051   TRAN_ABEND_PERCOLATE ---/ASRA DFHSR1   0003C2
00000003 09:19:48 0051   PROGRAM_CHECK        0C4/AKEA -noheda- 0089DC
00000004 09:19:48 0051   ABEND                ---/0999 DFHKERET 000064
00000005 09:23:02 0050   PROGRAM_CHECK        0C7/AKEA DFHYC520 000D7A
00000006 09:23:02 0050   TRAN_ABEND_PERCOLATE ---/ASRA DFHSR1   0003C2
00000007 09:23:02 0050   PROGRAM_CHECK        0C4/AKEA -noheda- 0089DC
00000008 09:23:02 0050   ABEND                ---/0999 DFHKERET 000064
```

Figure 11-6 Kernel error table summary

This display shows the kernel number associated with what we suspect is the failing task, KE_NUM=0050, that abended at 09:23:02. The last entry, ERR_NUM 00000008, although for KE_NUM=0050, is not the record we are interested in because it is most likely the result of earlier abends, in this case, error numbers 5, 6, and 7.

The first abend in the sequence, error number 5, is what we would usually want to investigate, and errors 6, 7, and 8 are most likely the result of a percolation of the error number 5 condition.

The first abend for KE_NUM=0050 is identified as a PROGRAM CHECK, where an ABEND0C7 was detected in load module DFHYC520 at offset X'000D7A'. This is reported to CICS as an AKEA abend.

Note: If you refer to the CICS kernel error stack display - level 1 - abend summary you will notice that the dump was taken for an abend at 09:27:51 and was for a severe error in module DFHXMIQ. The error code was X'100A'. Although there might be a relationship with the errors identified in the kernel error stack for KE_NUM=0050, the time difference of just under 5 minutes between the last recorded error and the dump will need to be checked to verify whether there is a relationship.

The kernel error stack - VERBX DFHPDxxx 'KE=2'

Further investigation of errors identified in the kernel error stack can be facilitated by using **`VERBX DFHPDxxx 'KE=2'`**. This display contains information and abbreviated storage dumps relating to each of these errors. The KE=2 option displays from the top (oldest) to the bottom (newest) entry.

11.5 CICS/TS 2.2 VERBEXIT options

Figure 11-7 shows some of the key CICS/TS **`VERBEXIT`** options that you will find useful to assist with most CICS problems you encounter. Level 1 = Summary only, Level 2 = Full Control Block formatting and Level 3 = Both 1 and 2.

Note: If you omit the level number, it defaults to level 3 for those components that have a summary, and level 2 for those that do not.

For a complete list of CICS problems, see *CICS Problem Determination Guide*, SC34-6002.

:

```
AP = 0|1|2|3 Application Domain
BR = 0|1|2|3 The 3270 bridge
CSA = 0|2 CICS Common System Area
DB2 = 0|1|2|3 The CICS DB2 interface
DS = 0|1|2|3 Dispatcher Domain
FCP = 0|2 File Control Program
KE = 0|1|2|3 CICS Kernel
LD = 0|1|2|3 Loader Domain
LG = 0|1|2|3 Logger Domain
LM = 0|1|2|3 Lock Manager domain
RM = 0|2 Recovery Management
SM = 0|1|2|3 Storage Manager domain
SO = 0|1|2|3 Sockets domain (530)
TCP = 0|1|2|3 Terminal Control Program
TR = 0|1|2|3 Trace domain
TRS = <trace selection parameters>
UEH = 0|2 User Exit Handler
WB = 0|1|2} The web interface
XM = 0|1|2|3 The transaction manager.
```

Figure 11-7 Some key VERBXEXIT options

11.6 CICS internal trace

Possibly the most useful diagnostic information that can be reviewed in a CICS SVC dump is the CICS internal trace. A record of all activity within the CICS region is stored. The default tracing options only capture *exception trace* entries. Unfortunately, we usually like to review what preceded the generation of an exception condition, as this is the only way to see what was the cause, not just the result.

Another unfortunate default is the CICS internal trace size, which is set to 64K. In a busy CICS region this would store less that one second of trace data – hardly enough to enable you to review the flow of the transaction that caused the exception condition. We recommend an internal trace size of at least 2500K. In fact, 5000K is probably an optimum size. This should provide you with sufficient trace information, except in exceptionally busy systems. In a large, high usage environment a 10000K trace table can hold as little a five seconds of trace data. While most CICS transactions are short duration, it is good to be able to review a trace that includes the start of the transaction.

CICS tracing is performed by the trace domain at predetermined trace points in the CICS code during the regular flow of control. This includes user tracing from applications. Tracing is performed when you turn on CICS internal tracing, auxiliary tracing, and GTF tracing. You can control the type of tracing to suit your needs, except when an exception condition is detected by CICS. CICS always makes an exception trace entry. You cannot turn exception tracing off. Trace points are included at specific points in CICS code; from these points, trace entries can be written to any currently selected trace destination. All CICS trace points are listed in alphanumeric sequence the *CICS User's Handbook*, SX33-1188.

Level-1 trace points are designed to give you enough diagnostic information to fix *user* errors. Level-2 trace points are situated between the level-1 trace points, and they provide information that is likely to be more useful for fixing errors within CICS code. You probably will not want to use level-2 trace points yourself, unless you are requested to do so by IBM support staff after you have referred a problem to them.

It is recommended to trace all CICS components at level 1 and that CICS tracing be active all the time. Lack of sufficient trace data can delay problem resolution, and Exception-only trace data is not sufficient in most cases.

11.7 CICS trace control facility

CICS exception tracing is always done by CICS when it detects an exception condition. The sorts of exception that might be detected include bad parameters on a domain call, and any abnormal response from a called routine. The aim is *first failure data capture*, to record data that might be relevant to the exception as soon as possible after it has been detected. The trace options can be set using the CICS/ESA® Trace Control Facility (CETR) transaction. This will enable you to increase the trace table size and let you control CICS component (domain) trace options. Tracing for all CICS components should be set to *level 1* when collecting diagnostic data.

You can also write out CICS trace data to a disk data set using the AUXTRACE facility. This is also controlled via the CETR transaction and can have the benefit of not requiring the CICS region to be dumped to review the data. This is a good option for tracing a re-creatable scenario. You can also request trace data to be collected for a specific transaction or terminal.

The AUXTRACE data can be formatted using the supplied DFHTRxxx program, where xxx represents the CICS release, for example, 530 for TS 1.3 and 620 for TS 2.2.

Figure 11-8 shows the CICS CETR Trace Control screen.

```
CETR CICS/ESA Trace Control Facility
Type in your choices.
Item Choice Possible choices
Internal Trace Status ===> STOPPED STArted, STOpped
Internal Trace Table Size ===> 0016 K 16K - 1048576K
Auxiliary Trace Status ===> PAUSED STArted, STOpped, Paused
Auxiliary Trace data set ===> B A, B
Auxiliary Switch Status ===> ALL NO, NExt, All
GTF Trace Status ===> STARTED STArted, STOpped
Master System Trace Flag ===> OFF ON, OFf
Master User Trace Flag ===> OFF ON, OFf
When finished, press ENTER.
PF1=Help 3=Quit 4=Components 5=Ter/Trn 9=Error List
```

Figure 11-8 CICS CETR trace control information

Issuing PF4 from the CETR transaction will display the CICS component trace options. This enables you to set tracing options for each CICS domain component.

If you have a requirement to capture trace data over a more substantial period of time, or trace the CICS system without taking a dump, you can use the CICS AUXTRACE facility. CICS auxiliary trace entries are directed to one of two auxiliary trace data sets, DFHAUXT and DFHBUXT. These are CICS-owned BSAM data sets, and they must be created before CICS is started. They cannot be redefined dynamically. The amount of data that can be collected is related to the size of the auxiliary trace data sets. You can use the AUXTR system initialization parameter to turn CICS auxiliary trace on or off in the system initialization table. Alternatively, you can select a status of STARTED, STOPPED, or PAUSED for CICS auxiliary trace dynamically using the CETR transaction.

To format an abbreviated trace, which displays one line per trace entry, issue:

```
VERBX DFHPDxxx 'TR=1'
```

To format an expanded trace, which produces a more comprehensive listing for each instruction, issue:

```
VERBX DFHPDxxx 'TR=2'
```

The size of the data formatted for a TR=2 request can be excessive, and due to the limitations of your TSO region size, can make it impossible to MAX to the end of the trace, which is usually where you will want to start. To assist with this process you can select only the trace data that you want to review.

For example, from the KE=1 display you have identified the KE_NUM of the task you want to review. Issuing the following command will let you select only the trace data related to the specified task:

```
VERBX DFHPD530 'TR=2,TRS=<KE_NUM=0050>'
```

Figure 11-9 shows an abbreviated trace listing. The best way to approach the trace is to MAX PF8 to the end of the display, and then issue the **FIND *EXC PREV** command. This will locate the last EXCEPTION in the trace table. The entries immediately preceding this exception can often identify the problem.

```
00128 QR AP 1710 TFRF  ENTRY RELEASE_FACILITY      NO,ABNORMAL,187A92
00128 QR AP 00E0 MGP   ENTRY 02206                 TERM
00128 QR AP 1700 TFIQ  ENTRY INQUIRE_TERMINAL_FACILITY 00000000 , 000
00128 QR AP 1701 TFIQ  EXIT  INQUIRE_TERMINAL_FACILITY/EXCEPTION
                             NO_TERMINAL
00128 QR XM 100A XMIQ  *EXC* Unexpected_return_code_from_TFIQ_INQUIRE
                             _REQUEST INQUIRE_TRANSACTION,000128C
00128 QR KE 0101 KETI ENTRY INQ_LOCAL_DATETIME_DECIMAL
```

Figure 11-9 VERBX DFHPD530 'TR=1,TRS=<KE_NUM=0050> output

Application program related trace points can be found in the trace by the EIP and APLI identifiers. It should also be noted that the trace shows the CICS processing flow of an ENTRY and EXIT for each function.

Figure 11-10 shows some APLI and EIP entries that also display the ENTRY and EXIT relationship.

```
AP 00E1 EIP   ENTRY FREEMAIN
SM 0D01 SMMF  ENTRY FREEMAIN              00140448,EXEC,CICS
SM 0D02 SMMF  EXIT  FREEMAIN/OK           USER24 storage at 001404
AP 00E1 EIP   EXIT  FREEMAIN              OK
AP 1949 APLI  EVENT RETURN-FROM-LE/370    Rununit_Termination OK C
AP 1948 APLI  EVENT CALL-TO-LE/370        Thread_Termination
AP 1949 APLI  EVENT RETURN-FROM-LE/370    Thread_Termination OK
AP 1941 APLI  EXIT  START_PROGRAM/EXCEPTION TRANSACTION_ABEND,ASRA
AP 0510 APAC  ENTRY REPORT_CONDITION
AP 1940 APLI  ENTRY START_PROGRAM         DFHTFP,NOCEDF,FULLAPI,SY
AP 00DD TFP   ENTRY TRANSACTION_ABENDED
SM 0301 SMGF  ENTRY FREEMAIN              1734CC14 , 0000006A,17C6
SM 0302 SMGF  EXIT  FREEMAIN/OK
XM 100A XMIQ  *EXC* - Unexpected_return_code_from_TFIQ_INQUIRE_TERMINAL_
                    (0000128C)
```

Figure 11-10 VERBX 'TR=1' output

Figure 11-11 shows an expanded trace listing example showing the *EXC entry. These expanded entries show the parameter lists that are being used by CICS and to the right of the display (not shown) is an ASCII *eyecatcher* that can also assist with identifying user data.

```
TASK-00128 KE_NUM-0050 TCB-QR   /008C7C40 RET-97770724 TIME-09:26:04.862
  1-0000  00F80000 000000A1 00000000 00000000  B020E000 04A00000 0178010
    0020  33782800 00000217 C1C1C4D4 68000000  00002800 00000000 000000A
    0040  C1C1C4D4 C3C3C1C1 C4C5D4D6 C4C6C8C3  C9C3E2E3 C2563000 0000600
    0060  000000A5 00000000 00000001 01010101  01010101 01010105 A901011
    0080  000060F1 0000128C C25A0001 01010101  01010101 01010117 174087E
    00A0  01010017 41E0D800 00000000 41E0D800  00000000 00100096 C29B7C9
    00C0  C29D5A96 C29F3600 01010100 00000097  629A5017 BFF08000 0060000
    00E0  00000017 C25E4096 C29AF817 C25A0000  00000017 C2527000
  2-0000  00780000 000000CD 00000000 00000000  B4040000 00000000 0100020
    0020  00000000 00000000 00000000 00000000  00000000 00000000 0000000
    0040  00000000 00000000 00000000 00000000  00000000 00000000 0000000
```

Figure 11-11 Trace entry parameter data produced by VERBX TR=2

CICS SVC dumps contain information about all the domain structures that make up CICS. These domains can be formatted using IPCS and are documented in the *CICS Diagnosis Reference*, LX33-6102. Trace information can be found in the *CICS Transactions for z/OS Trace Entries*, SC34-6242. All can be invaluable in assisting with problem determination.

Note: Additional information related to CICS problem diagnosis can be found in:

- *CICS Messages and Codes,* GC34-6241
- *CICS Problem Determination Guide*, SC34-6239

12

z/OS Language Environment

Language Environment (LE) provides a common run-time environment across multiple high-level languages. These languages include:

- COBOL
- C/C++
- PL/I
- FORTRAN
- Assembler (not HLL)

12.1 Run-time environment

Language Environment is supported across multiple platforms, z/OS, VM, and VSE.

A run-time environment provides facilities, such as storage control, system time and date functions, error processing, message processing, and other system functions to the high-level languages. The run-time library is *called* by the user program to perform these functions. Before LE, each high-level language had its own run-time library, but LE has combined the functionality required by each language into a single run-time environment.

There are two common execution library (CEL) modules that will indicate a failure, but the cause will be else where. The first is CEEHDSP, which schedules the LE CEEDUMP to be taken. The second module is CEEPLPKA, which will always indicate an ABENDU4039 or ABENDU4038 no matter what the original error. Your diagnostic methodology should exclude failures in these two modules.

The LE event handler modules are identified as CEE*xxx,* where *xxx* represents the language, as follows:

003 C/C++ Run-time (that is, CEEEV003)

005 COBOL

007 FORTRAN

008 DCE

010 PL/I

012 Debug Tool

12.1.1 Common LE messages

The following messages and abend prefixes can assist with problem diagnosis:

CEE Output by CEL but may be reporting a problem elsewhere

IGZ Output by COBOL

IBM Output by PL/I

AFH Output by FORTRAN

EDC Output by C/C++

Some common CEL messages that indicate exception (0Cx) conditions are:

- CEE3201 = ABEND0C1
- CEE3204 = ABEND0C4
- CEE32xx = ABEND0Cy, where y is the hex equivalent of decimal xx

Message CEE3250 indicates a non-exception (0Cx) abend has occurred.

Common CEL ABENDS:

U4038 Some *severe* error occurred but no dump was requested.

U4039 Some *severe* error occurred and a dump was requested.

U4083* Backchain in error - only occurs after some other error.

U4087* Error during error processing.

U4093* Error during initialization.

U4094* Error during termination.

The asterisk (*****) indicates that a reason code is required for this message to be meaningful.

12.2 LE and batch (IMS, WebSphere, and so forth)

1. Specify the following Runtime options:

   ```
   ABTERMENC(ABEND) TERMTHDACT(UADUMP) TRAP(ON)
   ```

   For information about how to specify runtime options refer to the section “Specifying Runtime Options under z/OS Batch” in the C/C++ User's Guide.

2. Include a SYSMDUMP DD card in the JCL specifying:

   ```
   SPACE=(CYL,(100,100),RLSE),DISP=(NEW,DELETE,CATLG),
   DSN=dump_dataset_name,LRECL=4160,RECFM=FBS
   ```

   Informational APAR II11016 explains how to find the PSW and GPRs at the time of failure.

Note: Ensure that your IEADMR00 Parmlib member reflects the following SDATA defaults:
`SDATA=(NUC,SQA,LSQA,RGN,TRT,LPA,CSA,GRSQ,SUM)`

12.3 LE and CICS

A transaction dump should be produced for all LE ABENDU40xx series abends, except ABENDU4038. If a transaction dump is not enough, request a CICS system dump. Use the CICS supplied CEMT transaction to do this:

```
CEMT SET TRD(40xx) SYS ADD
```

A CICS system dump of an ABENDU4038 is not helpful because it is taken at the time of the last termination, not at the point of detection. Instead, specify the following:

```
TERMTHDACT(UADUMP) ABTERMENC(ABEND) TRAP(ON)
```

This produces a CICS transaction dump with an ABENDU4039.

Note: SLIP commands on C=U40xx will not work in CICS. SLIP commands on C=0Cx will work in a CICS environment, but not in batch.

`TERMTHDACT(UADUMP) ABTERMENC(ABEND) TRAP(ON)` produces a CICS transaction dump. It will never produce a SYSUDUMP/SYSABEND/SYSMDUMP since LE's ESTAE routine does not get driven. Info APAR II13228 explains how to find the PSW and GPRs at the time of failure.

12.3.1 Additional procedure for an SVCdump for 40xx abends under CICS

Here are the steps to get an SDUMP for a specific 40xx transaction ABEND under CICS:

1. Make sure the CICS region is started with the DUMP=YES SYSIN input (SIP) parameter.
2. Make sure the SYS1.DUMP data sets are available. Most customers should have all this already set up.

   As an alternative, dynamic allocation facility can be used follows:

   ```
   DUMPDS ALLOC ADD,VOL=xxxxxx
   DUMPDS ALLOC=ACTIVE
   ```

After these commands, MVS will dynamically allocate data sets on the xxxxxx volume containing the dump with the following type of name:

```
SYS1.DUMP.D970910.T191701.SY1.S00001
```

3. Once the CICS region is up, log on and issue the following:

   ```
   CEMT SET TRD(40xx) ADD SYS
   ```

   Substitute the real dump code, for example, 4088 for 40xx. Following is a sample of what CICS sends back for this:

   ```
   SET TRD(4088) ADD SYS
   STATUS: RESULTS - OVERTYPE TO MODIFY
   Trd(4088) Tra Sys     Loc Max( 999 ) Cur(0000)
   ```

4. Run the transaction that creates the U40xx ABEND. A system, or SVC dump, should be produced at the point of the abend. This procedure will work for any transaction dump under CICS, not just U40xx ABENDs.

12.4 LE and UNIX System Services shell

Using UNIX System Services and the LE run-time options, consider the following steps to take system dumps:

1. To write the system dump to a data set, issue the command:

   ```
   export _BPXK_MDUMP=filename
   ```

   The filename is a fully qualified data set name with LRECL=4160 and RECFM=FBS

2. Specify Language Environment run-time options:

   Where the suboption is UADUMP (preferred), UAONLY, UATRACE, UAIMM. If UAIMM is set, `TRAP(ON,NOSPIE)` must also be set.

3. Rerun the program and the dump will be written to the specified data set.

12.5 Find failing module instructions

In most cases LE condition handling will trap original program checks (like ABEND0C4) and turn them into corresponding LE conditions (like CEE3204S). After storing information about the original program check, LE will terminate with an ABENDU40xx. When examining a dump of a U40xx the PSW and registers can be found in a control block called the ZMCH. APAR II11016 is specifically written for those running LE in a non-CICS environment, since the control block structure and condition handling changes when running under CICS.

While reviewing the SYSMDUMP in IPCS you can issue the following commands to format the ZMCH:

```
IP VERBX CEEERRIP 'CM'
IP VERBX LEDATA 'CM'
```

For more information on the syntax, refer to *z/OS Language Environment Run-time Messages*, SA22-7566 and *z/OS Language Environment Debugging Guide,* GA22-7560.

LEDATA searches for an error TCB and formats the control blocks for that TCB. If there is no error TCB (such as in a console dump), the TCB or CAA keywords will need to be specified as follows:

1. Load SYSMDUMP into IPCS. (Instructions on how to get a SYSMDUMP with LE are in info **APAR II10573**.)
2. Issue the command:

   ```
   IP SUMM FORMAT
   ```

3. Issue the command:

   ```
   BOTTOM or MAX (PF8)
   ```

4. Find the TCB with a non-zero completion code. Now issue the command:

   ```
   IP VERBX LEDATA 'CM TCB(xxxxxxxx)'
   ```

If this does not format the ZMCH, locate the CAA with the following steps:

1. Issue the following command, where the x's represent the address of the failing TCB:

   ```
   F 'TCB: xxxxxxxx' PREV
   ```

2. Issue the command:

   ```
   F RTM2WA
   ```

3. Press PF5 to search again. If there is a second RTM2WA for the failing TCB, then use the data contained within the RTM2WA.
4. Find the address in Register 12.
5. Issue the command **"=1"** to go into browse mode, or select option **1** from the IPCS Primary Option menu.
6. Issue the command `L` ***yyyyyyyy*** where *yyyyyyyy* represents the address obtained from Register 12.
7. Verify whether this is a valid CAA with the following:
   a. At the address in R12 verify that the value is "xxxx0800"
   b. Issue L X-18 and the eyecatcher should be **CEECAA**

   This indicates you have found a valid CAA and you can now issue the command:

   ```
   IP VERBX LEDATA 'CM CAA(yyyyyyyy)'
   ```

You have now formatted the ZMCH, so you can begin locating the values you were looking for.

Note: These steps *do not* pertain to an ABENDU4036.

12.5.1 Reason code information

Information for the particular reason code associated with the U4036 ABEND is in *z/OS Language Environment Debugging Guide,* GA22-7560. Follow the instruction there to find the information about the original error.

CEEDUMP output will go to SYSOUT and the system dump is dependant on whether a SYSMDUMP DD card has been included in the JCL.

Note: LRECL=4160 must be used for SYSMDUMP.

Setting a SLIP for an LE-handled abend is pointless because the LE abend is reissued at the end of LE termination and cleanup processing for the LE environment has already been performed.

12.6 IPCS and Language Environment

IPCS provides some facilities to assist with LE problem diagnosis. The IPCS command **VERBX LEDATA** or **VERBX CEEERRIP** shows the LE run-time options and general information about your LE environment at the time of the failure.

CEEERRIP is the LE diagnostic module that is used to format the dump data. Figure 12-1 shows the result of the **VERBX CEEERRIP 'CEEDUMP'** command and related traceback information.

```
Information for enclave main

Information for thread 8000000000000000

Traceback:
DSA Addr Program Unit PU Addr   PU Offset  Entry    E Addr   Status
00022460 CEEHSDMP     12B7F950  -0001FE3A  CEEHSDMP 12B7F950 call
00020018 CEEHDSP      12B27138  +000026A4  CEEHDSP  12B27138 call
000223C0              12B00B80  +00000062  func_#1  12B00B80 exception
00022320              12B00A00  +0000005E  func_#2  12B00A00 call
00022280              12B00880  +0000005E  func_#3  12B00880 call
000221E0              12B005E8  +00000066  main     12B005E8 call
000220C8              12D5BE0E  +000000B4  EDCZMINV 12D5BE0E call
```

Figure 12-1 LE traceback data

Figure 12-2 shows the **VERBX CEEERRIP** display for the LE run-time options.

```
LAST WHERE SET         Override   OPTIONS
*************************************************************************
INSTALLATION DEFAULT    OVR        ABPERC(NONE)
PROGRAM INVOCATION      OVR        ABTERMENC(ABEND)
INSTALLATION DEFAULT    OVR      NOAIXBLD
INSTALLATION DEFAULT    OVR        ALL31(OFF)
INSTALLATION DEFAULT    OVR        ANYHEAP(00016384,00008192,ANY  ,FREE)
INSTALLATION DEFAULT    OVR      NOAUTOTASK
INSTALLATION DEFAULT    OVR        BELOWHEAP(00008192,00004096,FREE)
INSTALLATION DEFAULT    OVR        CBLOPTS(ON)
INSTALLATION DEFAULT    OVR        CBLPSHPOP(ON)
INSTALLATION DEFAULT    OVR        CBLQDA(ON)
INSTALLATION DEFAULT    OVR        CHECK(ON)
INSTALLATION DEFAULT    OVR        COUNTRY(US)
INSTALLATION DEFAULT    OVR        DEBUG
INSTALLATION DEFAULT    OVR        DEPTHCONDLMT(00000010)
INSTALLATION DEFAULT    OVR        ENVAR("")
INSTALLATION DEFAULT    OVR        ERRCOUNT(00000020)
INSTALLATION DEFAULT    OVR        ERRUNIT(00000006)
INSTALLATION DEFAULT    OVR        FILEHIST
DEFAULT SETTING         OVR      NOFLOW
INSTALLATION DEFAULT    OVR        HEAP(00032768,00032768,ANY  ,
```

Figure 12-2 LE run-time options output from the VERBX CEEERRIP command

12.7 Finding the failing CSECT name in LE

Getting the name of the failing csect (function), or any LE-enabled CSECT can be performed as follows:

1. Get address (second word) from MCH_PSW.
2. Select Option **1** (Browse) in IPCS.
3. Issue command: **L xxxxxxxx** (where xxxxxxxx is the address).
4. Issue command **F CEE prev**.
5. Back up five bytes to the 47F0xxxx instruction. This is the beginning of the CSECT/function.
6. Add the value at offset X'0C' to the module address.
7. Go to that location.
8. Add X'20' bytes. This is a two-byte prefixed string (length) with the function name.

Figure 12-3 shows the steps needed to find the failing CSECT/Function.

```
12B00B80   47F0F026   01C3C5C5   000000A0   00000148  | .00..CEE.......
4. Take value at offset x'0C' and add to address 12B00B80+148
2. F CEE PREV
3. Back up 5 bytes this is the beginning of the module/csect/
function 12B00B80:
12B00B90   47F0F001   183F58F0   C31C184E   05EF0000  |.00....0C..+...
12B00BA0   000047F0   303A90E7   D00C58E0   D04C4100  |...0...X}..\}<.
12B00BB0   E0A05500   C3144720   F0145000   E04C9210  |\...C...0.&.\<k
12B00BC0   E00050D0   E00418DE   05304400   C1B04150  |\.&}\.......A..
12B00BD0   00005050   D0984400   C1AC4160   000A8E60  |..&&}q..A..-...
12B00BE0   00205D60   D0985070   D09C4400   C1AC58F0  |..)-}q&.}...A..
1. MCH_PSW points here do L 12B00BE6
12B00BF0   D09C47F0   302E0700   4400C1B8   58D0D004  |}..0......A..}}
12B00C00   58E0D00C   9837D020   051E0707   12B00C20  |.\}.q.}........
12B00C10   12B00CB0   12B00A00   80000001   000006D8  |..............
12B00C20   12B00B80   0000008A   12B00C5C   00000000  |..........*...
12B00C30   02E0004A   00000300   004E0000   03200057  |.\.U.....+.....
12B00C40   00000360   006B0000   03800078   000003A0  |...-.,.........
12B00C50   00780000   03A0007C   000003A0   0E0E0E0E  |.......@.......
12B00C60   0E0E0000   000F86A4   95836DA6   89A3886D  |......func_with
12B00C70   85999996   99400001   93400001   94400000  |error ..l ..m .
12B00C80   00000692   01012000   00000198   D000009C  |...k.......q}..
12B00C90   0000068E   01012000   00000198   D0000098  |...........q}..
12B00CA0   00000698   000006A8   00000000   00000000  |...q...y.......
12B00CB0   00000000   0000008A   000006B8   00010000  |..............
12B00CC0   00000002   F04A3042   10CEA186   FFFFFB08  |....0U....ef...
5. Go to that location (12B00CC8)
12B00CD0   0000008C   00000000   FFC00000   00000000  |.........{.....
12B00CE0   90000000   02C00019   000F86A4   95836DA6  |....{.....func_w
6. Add X'20' bytes to get to the 2 byte prefix string for the function/csect name
12B00CF0   89A3886D   85999996   99405000   0045FFFF  |ith_error.......
```

Figure 12-3 LE diagnostic flow for finding the failing CSECT

Some other helpful IPCS LE diagnostic options are:

- **VERBX CEEERRIP 'SM'** displays all storage management control blocks.

- **VERBX CEEERRIP 'HEAP'** displays HEAP storage management control blocks.
- **VERBX CEEERRIP 'STACK'** displays STACK storage management control blocks.

13

CICSPlex SM diagnostic procedures

The diagnostic data that can be collected will not assist you very much in solving application-type problems because the CICSPlex® SM diagnostic and tracing routines are primarily directed at the *internal* functions of CICSPlex SM. Despite this, it is extremely important that you understand how to turn on tracing and what dumps are required to help the support center assist with your diagnosis.

This chapter describes the following:

- Overview of the CICSPlex environment
- Diagnostic aids
- CICSPlex SM traces
- CICSPlex SM component trace options
- CICSPlex SM dumps
- CICSPlex SM components and IPCS

13.1 Overview of the CICSPlex environment

The CICSPlex SM environment is comprised of a number of address spaces. They are described in this section.

The *CICSPlex SM address space* (CMAS) is the hub of any CICSPlex SM configuration. Every CICSplex is managed by at least one CMAS and is responsible for the *single system image* (SSI) that enables the operator to manage a CICSplex as if it were a single CICS system, regardless of the number of CICS systems defined as belonging to the CICSplex, and regardless of their physical location.

The CMAS implements the BAS, WLM, RTA, and monitoring functions of CICSPlex SM, and maintains configuration information about the CICSplexes it is managing. It also holds information about its own links with other CMASs and it stores this information in its data repository.

A CMAS is a full-function CICS Transaction Server for z/OS systems. Most CMAS components run as CICS tasks, and CMAS connections to other components are implemented using CICS intercommunication methods.

Each running CICS system that is being managed by CICSPlex SM is known as a *managed application system* (MAS).

The *Web user interface* (WUI) server runs as a CICSPlex SM local MAS and communicates with the managed resources via the CMAS to which it is connected. This CMAS needs to manage all CICSplexes that the WUI server needs to access. This is because the WUI server acts as a CICSPlex SM API application. The Web User Interface is accessed using standard Web browser software.

When a CMAS is initialized, up to 9 MVS data spaces are created. These data spaces are used by CICSPlex SM to allow quick access to data from a CMAS and the MASs attached to it. Although the data spaces are logically owned by the CMAS, they are physically owned by the ESSS address space (EYUXxxx).

The *coordinating address space* (CAS) is an MVS subsystem whose main function is to support the MVS/TSO ISPF end-user interface (EUI) to CICSPlex SM. The CAS is not part of a CICSplex, but belongs to the managing topology of CICSPlex SM. The Web user interface is the evolution of the EUI.

13.2 Diagnostic aids

There are several online diagnostic aids that can be very useful to prepare diagnostic data, as follows:

- CICSPlex SM views that provide diagnostic information about:
  - CMAS and MAS status
  - Resource monitoring activity
  - Real-time analysis activity
  - Workload management activity
- CICS commands that produce data similar to CICSPlex SM data
- The CICSPlex SM online utility transaction (COLU)
- The CICSPlex SM interactive debugging transactions (COD0 and CODB)

Messages are often the first or only indication to a user that something is not working. CICSPlex SM writes error and informational messages to a variety of destinations:

- The system console or system log
- The CMAS or MAS job log
- The EYULOG transient data queue
- The SYSOUT data set
- A CICS terminal
- The TSO READY prompt
- The ISPF end-user interface

Any CMAS or local MAS can produce symptom strings in a system or transaction dump. Symptom strings describe a program failure and the environment in which the failure occurred. All CICSPlex SM symptom-strings conform to the RETAIN® symptom-string architecture. They are stored as SYMREC records in the SYS1.LOGREC data set.

LOGRECs are records containing information about an abnormal occurrence within CICSPlex SM. The records are written to the SYS1.LOGREC data set and are available for analysis after a failure.

The LOGRECs produced by CICSPlex SM all contain the same data. The data includes extensive information about the state of CICSPlex SM components in the failing address space at the time the LOGREC is written, such as:

- Identification of the failing module
- Module calling sequence
- Recovery management information

13.3 CICSPlex SM traces

The CICSPlex SM trace facilities provide a detailed record of every exception condition that occurs. They can also be used to trace various aspects of component processing.

In CMASs and MASs, CICSPlex SM writes user trace records to the CICS trace data set, as follows:

- If any local or remote MAS is in communication with a CMAS, trace data is shipped from the MAS to the CMAS, and a full, formatted trace record is produced.
- If any local or remote MAS is not in communication with a CMAS (either because the communications component is not yet active or because there is a problem with communications itself).

In CASs, trace data is written to a wrap-around buffer, with new data overwriting old data. Because CAS trace data is not written to any external device, it can be examined only within the context of an address space dump.

MVS/ESA™ system dumps are an important source of detailed information about problems. For CMASs and local MASs, CICSPlex SM recovery routines produce a system dump when an unexpected error occurs in a supervisory function. Users can also request a system dump at any time.

Whether it is the result of an abend or a user request, a system dump provides an exact image of what was happening in a CICSPlex SM address space at the time the dump was

taken. A dump can be used to determine the state of all components in the address space, allowing you to:

- Examine MVS/ESA system trace entries
- Determine subsystem status
- Analyze CAS message trace table entries
- Locate key data areas
- Provide information on all CICSPlex SM components
- Provide information on all active CICSPlex SM tasks
- Provide SNAPshots of appropriate CICSPlex SM data spaces

13.4 CICSPlex SM component trace options

All CICSPlex SM address space (CMAS), managed application system (MAS), the Web user interface (WUI) address space, and coordinating address space (CAS) components provide trace data.

Note: The CICS internal trace facilities must always be active in a CMAS.

When a CMAS is initialized, CICSPlex SM ensures that the CICS trace facility is active and the trace table is large enough. The trace table settings required by the CMAS, along with the CICS SIT options that you need to use in order to establish these settings, are:

- Internal trace must be ON. SIT parameter is INTTR=ON.
- Trace table size must be at least 2MB. SIT perimeter TRTABSZ=2048.
- Master trace must be OFF. SIT parameter SYSTR=OFF.
- User trace must be ON. SIT parameter USERTR=ON.

Additionally, the CICS AUXTRACE facility should be active (for user records only) in a CMAS. If this facility is not active when a problem occurs, it may be necessary to recreate the problem with the facility turned on.

During normal CMAS and MAS processing all the standard and special trace levels (levels 1 through 32) are usually disabled. Exception tracing is always active and cannot be disabled.

You can turn tracing on for a specific CMAS or MAS component in either of two ways:

- Specify system parameters on a CMAS or MAS startup job.
- Use the ISPF end-user interface to activate one or more levels of tracing dynamically while CICSPlex SM is running.

13.4.1 CMAS and MAS tracing

You can use the CMAS or MAS view to control the tracing that occurs in an active CMAS or MAS. For example, if you want to change the trace levels for the CMAS called EYUCMS1A, do the following:

1. Issue the CMAS **VIEW** command.
2. Either issue the **TRACE** primary action command from the COMMAND field, as shown in the following figure, or enter the **TRA** line action command in the line command field next to EYUCMS1A.

Figure 13-1 shows the CMAS view **TRACE** command.

```
12APR1999  10:32:30 ----------- INFORMATION DISPLAY -------------------
COMMAND  ===>  TRACE EYUCMS1A                 SCROLL ===> CSR
CURR WIN ===> 1        ALT WIN ===>
 W1 =CMAS==============EYUCMS1A=EYUCMS1A=12MAY1997==14:46:30=CPSM======2
CMD Name     Status   Sysid Access   Transit  Transit
--- -------- -------- ----- Type---- CMAS---- Count--
    EYUCMS1A ACTIVE   CM1A  LOCAL                 0
```

Figure 13-1 Controlling CICSplex SM tracing from a CMAS view

3. The component trace input panel appears, as shown in the following figure. This identifies the current trace settings for each component in the CMAS. A setting of Y means that trace level is active for the component; a setting of N means tracing is not active. Figure 13-2 on page 117 shows the CPSM Component Trace levels.

```
-------------------- Component Trace Levels for EYUCMS01 -------------
COMMAND  ===>

    Overstrike the level number with a Y or N to alter the trace level

    Level                 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 3 3 3
         1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2
Component----------------------------------------------------------------
KNL      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
TRC      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
MSG      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
SRV      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
CHE      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
QUE      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
DAT      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
COM      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
TOP      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
MON      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
RTA      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
WLM      N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N N
```

Figure 13-2 CICSPlex SM component trace options

Note: Level 3 through 32 trace points should be activated only for a specific CMAS or MAS component and only at the request of customer support personnel.

4. To change a trace setting for a specific component, such as Kernel Linkage (KNL):
   a. Position the cursor next to KNL.
   b. Move the cursor across the line to the appropriate level (1 through 32).
   c. Type either Y, to activate tracing, or N, to deactivate tracing.
5. When all the trace settings are correct, press Enter. The CMAS view is redisplayed.

Formatting the CMAS and MAS trace data

CICSPlex SM trace data is written to the CICS AUXTRACE data sets, DFHAUXT and DFHBUXT. The CICSPlex SM trace data needs to be formatted using the supplied EYU9XZUT program, that is located in the SEUYLOAD data set.

Some of the more common EYU9XZUT parameters that can be used to format the trace data include:

- **ABBREV -** This provides an abbreviated trace, which has one line per trace record with a sequence number at the far right.

  Use the sequence number to select full trace formatting of specific records.

  The abbreviated trace is written to a SYSOUT file named TRCEABB.
- **COMPID=xxx,...|ALL** - Allows you to specify the three-character identifier of the components whose trace entries you want to format, or ALL for all CICSPlex SM components.
- **EXCEPTION=ONLY|ALL** - Formats only those exception trace records that match all other criteria and ALL formats all exception trace records, as well as any other trace records that match all other criteria.
- **FULL** - Provides full trace formatting of trace records meeting all selection criteria.

Figure 13-3 shows an example of the JCL that can be used to format the CICSPlex SM trace data.

```
//jobname  JOB (acct),'name',CLASS=x,MSGCLASS=x
//TRCLST  EXEC PGM=EYU9XZUT,REGION=2048K
//STEPLIB  DD  DSN=CICSTS31.CPSM.SEYULOAD,DISP=SHR
//SORTWK01 DD  SPACE=(CYL,(3,2)),UNIT=SYSDA
//SYSPRINT DD  SYSOUT=*
//SYSOUT   DD  SYSOUT=*
/TRCEIN   DD  DSN=cics.system.DFHTRACA,DISP=SHR
//         DD  DSN=cics.system.DFHTRACB,DISP=SHR
//TRCEOUT  DD SYSOUT=*,COPIES=1
//TRCEABB  DD SYSOUT=*,COPIES=1
//SYSIN    DD  *
ABBREVIATED
FULL
COMPID=MON
EXCEPTION=ONLY
/*
```

Figure 13-3 Example JCL to format CICSPlex SM trace data

CICSPlex SM transactions

CICSPlex/SM provides three online transactions. These are:

- COLU
- COD0
- CODB

The CICSPlex SM online utility (COLU) is a CICS transaction that can be used to generate reports about various CMAS and local MAS components. The interactive debugging transactions COD0 and CODB provide access to the CICSPlex SM run-time environment.

They can be used to format and manipulate the internal data structures of CICSPlex SM. The debugging transactions can run in CMASs and MASs.

Attention: The CICSPlex SM interactive debugging transactions COD0 and CODB, and online utility COLU should be used only at the request of IBM customer support personnel. You must take steps to ensure that these transactions can be used only by authorized personnel because of the extent of the access to system control areas that they provide. Improper or unauthorized use of COD0 and CODB may have very serious consequences, including without limitation loss of data or system outage. The customer is solely responsible for such misuse.

13.5 CICSPlex SM dumps

A dump of a CICSPlex SM address space can be initiated by the MVS **DUMP** command or via the CICSPlex/SM ISPF end-user interface/Web-user interface. You can use the SNAP action command on the CICSRGN view to request a system dump for an MAS.

To dump all the related CICSPlex SM address spaces using the MVS dump command, the following example can be used as a guide:

```
/DUMP COMM=(your dump title)
/R ww,JOBNAME=(EYUX220,cmasname,lmasnames),DSPNAME=('EYUX220'.*),CONT
/R yy,REMOTE=SYSLIST *('EYUX220','CMAS*',DSPNAME,SDATA),CONT
/R zz,SDATA=(RGN,PSA,ALLNUC,SQA,CSA,PSA,TRT,COUPLE,XESDATA),END
```

The parameters used in this example are the following:

- `EYUX220` is the CPSM release.
- `cmasname` is the name of the CMAS.
- `lmasnames` are the names of the LMASs.

  You can use wildcards if the CMASs and LMASs share the start of the name. For example, if the MASs are named AOR1, AOR2, you could use AOR* in place of a list of their jobnames.
- `DSPNAME` is all of the data spaces are associated with the ESSS.
- `REMOTE` will dump the other CMASs and their ESSS data spaces.

Alternatively, a dump from the COD0 debugger can be taken by logging onto the CMAS, entering the COD0 transaction, then issuing **START XZSD**.

For example, if you want a system dump for the MAS called EYUMAS1A, do the following:

1. Issue the **CICSRGN** view command.
2. Either issue the **SNAP** primary action command from the COMMAND field, as shown, or enter the **SNA** line action command in the line command field next to EYUMAS1A.

Figure 13-4 shows the CPSM SNAP initiation.

```
12APR1999 10:30:30 --------- INFORMATION DISPLAY ----------------------
COMMAND  ===> SNAP EYUMAS1A                    SCROLL ===> CSR
CURR WIN ===> 1         ALT WIN ===>
 W1 =CICSRGN=CICSRGN==EYUPLXO1==EYUCSGO1==12MAY1997==11:30:30=CPSM======
CMD CICS     Job      MVS  Act  CICS   CPU      Page     Page     Total
--- System-- Name---- Loc  Task Status Time---- In------ Out----- SIO---
    EYUMAS1A CICPRODA SYSA   34 ACTIVE 12345678  1234567  1234567 123456
    EYUMAS2A CICAOR1P SYSA   22 ACTIVE      567  1234567  1234567    106
```

Figure 13-4 Initiate CICSPlex SM system dump via SNAP

3. When the CICS SNAP input panel appears, specify:
   - A 1- to 8-character dump code
   - An optional 1- to 8-character caller ID
   - An optional title of up to 79 characters

Figure 13-5 shows the SNAP panel dump option view.

```
 ---------------------------- CICS SNAP -----------------------------
 COMMAND  ===>

 Specify the options to be used for this dump of CICS:

 Dump  Code ===> NORMAL          1- to 8-character dump code

 Caller     ===> NO              1- to 8-character caller ID

                            TITLE (79 characters)

Press Enter to continue CICS dump with the options specified.
```

Figure 13-5 CICSPlex SM system dump via SNAP input panel

The following message appears in the window to confirm the dump request:

```
EYUEI0568I Dump Taken for EYUMAS1A, assigned DUMPID is nn/nnnn
```

In this message, `nn/nnnn` is the dump ID assigned by MVS.

In the job log for EYUMAS1A you will see:

```
10.03.05 JOB00221 +DFHDU0201 EYUMAS1A ABOUT TO TAKE SDUMP. DUMPCODE:code
10.03.12 JOB00221 IEA794I SVC DUMP HAS CAPTURED:
                  DUMPID=005 REQUESTED BY JOB (EYUMAS1A)
                  DUMP TITLE=CICSDUMP: SYSTEM=EYUMAS1A CODE=code ID=nn
10.03.05 JOB00221 +DFHDU0202 EYUMAS1A SDUMP COMPLETE.
```

The interactive problem control system (IPCS) provides MVS users with an interactive facility for diagnosing software failures. You can use IPCS to format and analyze SDUMPs produced by CICSPlex SM or stand-alone dumps obtained while CICSPlex SM was active in the system being dumped. You can either view the dumps at your terminal or print them.

13.5.1 CICSPlex SM IPCS tools

CICSPlex SM provides two types of IPCS tools:

- A set of panels (driven by a corresponding set of CLISTs) that allow you to display:
  - The data in a coordinating address space (CAS) dump
  - The names and locations of control blocks and areas of a CAS dump
  - Subsystem information
  - Address space-related control blocks
  - Modules loaded by CICSPlex SM
  - Tasks created by CICSPlex SM
  - Storage subpools managed by CICSPlex SM
  - BBC LU 6.2 communication information
- A dump formatting routine that can be used with the **VERBEXIT** subcommand to format CMAS or MAS dumps. Figure 13-6 shows the CPSM IPCS Dump Analysis panel.

```
-------------------- CICSPlex SM Subsystem Dump Analysis ----------
 OPTION  ===>


 CAS Subsystem Id  ===>
 Base Tech Version ===> 2

    0  SUBSYSTEMS  - Identify CICSPlex SM CAS subsystems
    1  STATUS      - CAS & connected memory status
    2  MESSAGES    - CAS message trace table
    3  COMPONENT   - CAS component level problem analysis
```

Figure 13-6 CICSPlex SM Subsystem Dump Analysis panel

This panel can be invoked in any of the following ways:

- At the IPCS primary option menu, select option **2** ANALYSIS.
- At the IPCS MVS analysis menu, select option **6** COMPONENT.
- At the IPCS MVS component menu, select **CPSMSSDA**.

After identifying the CAS subsystem, select option **3**, COMPONENTS. The panel shown Figure 13-7 will be displayed.

```
---------------- CICSPlex SM Subsystem Component Analysis -------------
OPTION  ===>

Select component to analyze

   1  PROGRAMS    - Locate/display loaded programs
   2  TASKS       - Display execution unit information
   3  STORAGE     - Display storage block/pool information
   4  BBC         - Display communication information
```

Figure 13-7 CICSPlex SM component analysis

13.6 CICSPlex SM module names, components and IPCS

The names of modules, macros, and other source members distributed with CICSPlex SM take the form **prdtccxx.**

The components of the name have the following meanings:

prd A product code of BBC, BBM, or EYU
t Identifies the type of element, as listed in Element type identifiers
cc Component identifier, as listed in Component identifiers
xx A unique identifier assigned by each component.

For example, EYU0MMIN is an executable module for the Monitor Services component.

13.6.1 Element type identifiers

The element type identifiers are the following:

$ Selection menus
0 Executable modules (C or assembler)
5 EUI record maps
6 Dynamically acquired control blocks or data areas
7 Module entry point descriptors
8 Function/service definition tables and assembled control blocks
9 Load modules
B or R Assembler mapping DSECTs
C C code generation macros
D ISPF display or data entry panels
E CLISTs
F Function variables
G ISPF message definitions
H ISPF help panels
J Screen definitions
M C structure TYPEDEFs
P Profile variables or USERFILE members
Q Assembler code generation macros
S EUI class tables
T View, message, and action tables
U Assembler equate files
V C equate files
W or X Assembled help modules
Z View definitions

13.6.2 CICSPlex SM component identifiers

CICSPlex SM component identifiers begin with one of three prefixes: BBC, BBM, or EYU.

The BBC components are:

Qx PlexManager Data Collectors
Sx Communications Server Controller
Ux End-user Interface Address Space Services
Zx Global Services

The BBM components are:

Cx Data Manager
Hx Information Services

Lx Linkage Services
Mx File Management
Px Low-level Storage Management
Qx PlexManager Selectors
Sx General Services
Tx TSO Support Functions
Xx Transaction Management
Zx System and Application Control

The EYU components are:

Bx Business Application Services
Cx Communications
Ex End-user Interface
Mx Monitor Services
Nx Managed Application System
Px Real-time analysis
Tx Topology Services
Wx Workload Manager
XC Data Cache Manager
XD Data Repository
XE Environment Services System Services
XL Kernel Linkage
XM Message Services
XQ Queue Manager
XS Common Services
XZ Trace Services

13.6.3 The CICSPlex SM components and 3-character identifiers

The CICSPlex/SM components and their 3-character identifiers are:

BAS Business Application Services
SRV Common Services
COM Communications
CHE Data Cache Manager
DAT Data Repository
ESSS Environment Services System Services
KNL Kernel Linkage
MAS Managed Application System
MSG Message Services
MON Monitor Services
QUE Queue Manager
RTA Real-time analysis
TOP Topology Services
TRC Trace Services
WLM Workload Manager

IPCS VERBX command

These components can all be interrogated in a dump using the IPCS **verbx** command. The CICSPlex SM IPCS verbx command is EYU9Dxxx where xxx is 140, 220, 230 or 310 depending on the version of CICSPlex SM. An example is **verbx eyu9d140 'xxx'** where xxx is the CPSM component.

The components can be appended to each other in the **verbx** command as follows:

```
verbx eyu9d140 'bas,top,wlm,knl'
```

One handy tip for interrogating the dump is to find out the RMID (PTF level) of a particular CICSPlex/SM module.

If we had received abend information for module EYU0XDDL, we could tell from the information relating to the format of module names, that this module was related to the CICSPlex/SM XD or Data Repository component.

To format the Data Repository data in a dump we would use the following command:

```
verbx eyu9d140 'dat'
```

Figure 13-8 shows the Load Module information associated with the CPSM Data Repository component.

```
Method header information:
  Load      Entry     Exec      Recv CP/SM  Assem     Assem Oper AR  PTF
  Address   Point     Name      Name  Ver   Date      Time  Sys Mode Level
  1F0B2BB8  1F0B2BF8  EYU0XDAD  XDAD  130   06/03/99  20.28  00  00  PQ25568
  1F0E2158  1F0E2198  EYU0XDGT  XDGT  130   01/17/97  13.46  00  00  CPSM130
  1F0B61F8  1F0B6238  EYU0XDDL  XDDL  130   06/03/99  20.31  00  00  PQ25568
  1F0C0318  1F0C0358  EYU0XDUP  XDUP  130   06/03/99  20.37  00  00  PQ25568
  1F0A9840  1F0A9880  EYU0XDIN  XDIN  130   02/12/99  18.39  00  00  PQ21216
  1F0C0CE0  1F0C0D20  EYU0XDXP  XDXP  130   08/12/99  19.28  00  00  PQ25568
  1F0BA630  1F0BA670  EYU0XDIP  XDIP  130   08/12/99  19.25  00  00  PQ25568
```

Figure 13-8 OUTPUT from IPCS CPSM format of the DAT component

From this we can see that module EYU0XDDL is at the APAR PQ25568 level. We can also see that this module is loaded at x'1F0B61F8'.

Note: Additional information related to CICSPlex SM problem diagnosis can be found in:

- *CICSPlex SM Messages and Codes*, GC33-0790
- *CICSPlex SM Problem Determination*, GC34-6472

14

DB2 problem diagnosis

As with all problems, the messages and codes generated by the DB2 subsystem should be reviewed for symptoms. When diagnosing DB2 problems it will be necessary to dump the DB2 address spaces. DB2 consists of more than one address space. Depending on your installation these could be:

- DB2 master (MSTR) address space
- Database manager (DBM) address space
- Internal resource lock manager (IRLM) address space
- Distributed data facility (DIST) address space
- Established stored procedures (SPAS) address space

Usually you would dump the MSTR, DBM, and IRLM address spaces.

This chapter describes the following:

- System trace table
- DB2 dump collection
- Data sharing and IRLM
- DB2 tracing
- DB2 dump diagnosis using IPCS

14.1 System trace table

Ensure that the MVS system trace table size is set to 999K. This can be set via the MVS command **TRACE ST,999K** and can also be specified in the MVS COMMNDxx SYS1.PARMLIB member.

The default size is only 64K, which is insufficient to provide enough detailed trace data.

The system trace table is page fixed storage and you need to ensure that are enough real page frames to make this specification.

14.1.1 Master trace table

The MVS master trace table size should be set to at least 72K. The default size is only 24K, which will hold approximately 336 messages.

This enables you to review the log from dump data and a larger master trace buffer gives you more data to work with. This data can also be reviewed from the MVS system log (SYSLOG).

To increase the master trace size, issue the MVS command **TRACE MT,72K**. The size can also be set in the SCHEDxx member of SYS1.PARMLIB.

14.1.2 Common storage tracker

The MVS common storage tracking function can be used to track ownership of the CSA/ECSA.

This is set in the DIAGxx member of SYS1.PARMLIB and will be activated at IPL time or via the **SET DIAG=XX** operator command. For example:

```
DIAG xx member: VSM TRACK CSA(ON)
```

The advantage of setting this is that the SVC dumps (or RMF reports) provide CSA/ECSA ownership information with jobname, time and requesting module information.

It should be noted that there is a slight performance overhead associated with CSA storage tracking.

14.1.3 CHNGDUMP MAXSPACE

Ensure adequate dump space is allowed for. This is set via the CHNGDUMP MAXSPACE command and should be specified in the COMMNDxx member of SYS1.PARMLIB. For example:

```
CD SET,SDUMP,MAXSPACE=2500M
```

The default size is 500M. A MAXSPACE value of 2500M is the recommended minimum for DB2 multi-address space SVC dumps.

You should ensure that the local page data sets are large enough to contain their normal peak load plus additional SVC dumps since this can result in a system wait 03C.

Remember, a partial dump is useless most of the time.

14.1.4 SDATA

Make sure that the SDATA setup is as follows:

```
SDATA=(RGN,CSA,SQA,LPA,LSQA,SWA,PSA,ALLNUC,XESDATA,TRT,GRSQ,SUM)
```

Make sure the ABEND04E dump is not suppressed by DAE.

14.1.5 What data to collect for DB2 problems

While it is difficult to give a generic list of commands that should be issued prior to taking a dump of the DB2 address spaces, the following are a good set that will assist with DB2 problem diagnosis:

- `DISPLAY THREAD(*) DETAIL`
- `D A,ALL` (or `D A,ssid*` or `D A,IRL*`)
- `D GRS,CONTENTION`
- `D OPDATA`
- `DISPLAY DATABASE(*) USE/LOCKS LIMIT(*)`
- `DISPLAY UTILITY (*)`

Be sure to keep the MVS SYSLOG that is the **DISPLAY** command output. This will also be stored in the Master Trace table, which if large enough will also contain the results of the commands listed here.

If your installation is experiencing problems in Data Sharing environments, the following commands will collect additional useful data:

- `F irlmproc,STATUS,ALLD`
- `F irlmproc,STATUS,ALLI`
- Run the DB2 DIAGNOSE utility with MEPL option that will produce a Module Entry Point Listing. This shows the PTF level of the DB2 modules. The DB2 MSTR dump also contains this data.

It is possible, in fact probable, that if the DB2 subsystem, or MSTR address space is hung, then you will not get a response from the **DISPLAY** commands. In this case, a dump of the related DB2 address spaces is the only option.

14.2 DB2 dump collection

Be sure that you have MVS common storage tracking enabled via the SYS1.PARMLIB, DIAG01 member. There is no overhead running the MVS common storage tracker and it is recommended that this be enabled, particularly if you are experiencing virtual storage problems. The dump can be captured as follows:

```
DUMP COMM=(reason for taking dump)
```

You will then be required to enter, via the z/OS REPLY nn, response, the relevant DUMP options. For example:

```
R xx, JOBNAME=(*MASTER*,ssidMSTR,ssidDBM1,ssidIRLM,ssidDIST,XCFAS),CONT
R xx,SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,SQA,
TRT,CSA,GRSQ,XESDATA,WLM),END
```

Alternatively, you can set up the dump parameters in the SYS1.PARMLIB IEADMCxx members. For example:

```
TITLE=(DUMP OF CICS TOR, AOR and LOGGER),
JOBNAME=(*MASTER*,ssidMSTR,ssidDBM1,ssidIRLM,ssidDIST,XFCAS),
SDATA=(COUPLE,ALLNUC,LPA,LSQA,PSA,RGN,
SQA,TRT,CSA,GRSQ,XESDATA,WLM)
```

The dump can now be captured using the following command:

```
DUMP TITLE=(CICS Looping),PARMLIB=CI
```

CI is the IEADMCxx parmlib member suffix, namely SYS1.PARMLIB(IEADMCCI).

The title is the name (1 to 100 characters) you want the dump to have. This title becomes the first record in the dump data set. COMM= and TITLE= are synonyms.

You can also use the parmlib parameter as follows:

```
DUMP COMM=(.....),PARMLIB=(xx)
```

For DB2 connectivity issues it may also be necessary to dump the OMVS segment and the TCP/IP address space. For example:

```
R xx,JOBNAME=(OMVS,TCPIP),CONT
R xx,DSPNAME=('OMVS'.*,'TCPIP'.*),END
```

In a sysplex you may also need to dump associated address spaces in other MVS images within the plex. Review the previous section regarding dumping multiple address spaces within a sysplex.

14.3 Data sharing and IRLM

If your data sharing system experiences delays in child lock propagation or P-lock negotiation, you can request dumps of all IRLM instances in the data sharing when delay for child lock propagation lasts 45 seconds or more, or a delay for P-lock negotiation lasts two minutes or more. Issue this console command to request IRLM dumps for delays in child lock propagation:

```
MODIFY irlmproc,DIAG,DELAY
```

Issue this console command to request IRLM dumps for delay in P-Lock Negotiation:

```
MODIFY irlmproc,DIAG,PLOCK
```

Issue this console command to request IRLM dumps for either type of delay:

```
MODIFY irlmproc,DIAG,ALL
```

14.4 DB2 tracing

The DB2 command **START TRACE** starts DB2 traces. This command can be issued from an MVS console, a DSN session, a DB2I panel (DB2 COMMANDS), an IMS or CICS terminal, or a program using the instrumentation facility interface (IFI). To execute this command, the privilege set of the process must include one of the following:

- TRACE privilege
- SYSOPR, SYSCTRL, or SYSADM authority

DB2 commands issued from an MVS console are not associated with any secondary authorization IDs. The format of the **START TRACE** command is:

```
START TRACE (PERFM)   DEST(GTF)   PLAN(plan_name,..) CLASS(class)
        (ACCTG)       (SMF)
        (STAT)        (SRV)
        (AUDIT)       (OP)
        (MONITOR)     (OPX)
```

You must specify a trace type. The options PERFM, ACCTG, STAT, AUDIT, and MONITOR identify the type of trace started, as follows:

(PERFM) Intended for performance analysis and tuning, and includes a record of specific events in the system.

(ACCTG) Intended to be used in accounting for a particular program or authorization ID, and includes records written for each thread.

(STAT) Collects statistical data broadcast by various components of DB2, at time intervals that can be chosen during installation.

(AUDIT) Collects audit data from various components of DB2.

(MONITOR) Collects monitor data. Makes trace data available to DB2 monitor application programs.

14.4.1 Trace output for DB2

The DEST option specifies where the trace output is to be recorded. You can use more than one value, but do not use the same value twice. If you do not specify a value, the trace output is sent to the default destination. If the specified destination is not active or becomes inactive after you issue the **START TRACE** command, you receive message DSNW133I, which indicates that the trace data is lost. This applies for destinations GTF, SRV, and SMF. You also receive this message for destinations OPn and OPX if **START TRACE** is not issued by an application program, as follows:

- GTF - The MVS generalized trace facility (GTF). The record identifier for records from DB2 is X'0FB9'.
- SMF - The system management facility. The SMF record type of DB2 trace records depends on the IFCID record, as shown in the following list:

```
IFCID Record                          SMF Record Type
1 (SYSTEM SERVICES STATISTICS)        100
2 (DATABASE SERVICES STATISTICS)      100
3 (AGENT ACCOUNTING)                  101
202 (DYNAMIC SYSTEM PARAMETERS)       100
230 (DATA SHARING GLOBAL STATISTICS)  100
239 (AGENT ACCOUNTING OVERFLOW)       101
ALL OTHERS                            102
```

- SRV - An exit to a user-written routine. For instructions and an example of how to write such a routine, see the macro DSNWVSER in library prefix SDSNMACS.
- OPn - A specific destination. n can be an integer from 1 to 8.
- OPX - A generic destination which uses the first free OPn slot.

You will most likely trace to DESTination GTF or SMF.

You can trace specific entries, for example:

- **PLAN(plan-name,...)** introduces a list of specific plans for which trace information is to be captured. The default traces all plans.
- **AUTHID(authid,....)** traces specific authids.
- **CLASS(class integer,...)** traces specific classes and is dependant on the type of trace data you are collecting.

Tracing all Performance Class records and writing to GTF would be started as follows:

```
-START TRACE(PERFM) DEST(GTF) CLASS(*)
```

14.5 DB2 dump diagnosis using IPCS

When using IPCS to view a storage-related dump, you may find the following IPCS commands helpful:

```
STATUS
RSMDATA
RSMDATA VIRTPAGE ASID(x'yy')
ASMCHECK
VERBX DAEDATA
VERBX VSMDATA
VERBX VSMDATA 'OWNCOMM'      (Check Common Storage Tracking)
VERBX VSMDATA 'OWNCOMM DETAIL ALL SORTBY(ASIDADDR)'
VERBX VSMDATA 'OWNCOMM DETAIL ASID(ddd) SORTBY(TIME)'
VERBX VSMDATA 'NOGLOBAL,JOBNAME(xxxxDBM1)'
```

DB2 dumps can be processed using IPCS and **VERBX DSNWDMP** will format the DB2 dump data. For example, thread usage in a DB2 dump can be reviewed by using:

```
verbx dsnwdmp 'sumdump=no,subsys=itso,ds=1'
```

Figure 14-1 shows the result of the DB2 IPCS DS=1 format for the ITSO subsystem.

```
ACE: 1C505388 Status: T   Req: 0076    Allied Chain
Authid: PZ01     Plan: SPPOOL   Corrid: ENTRPZ010043 Corrname: ITSOCIC3 Token: 00007B07
EB       Primary(Asid)  Home(Asid)     EBSPAWND TCB/SRB  -Status-- R14
1C505418 ITSOCIC3(0094) ITSOCIC3(0094) 00000000 00991850 Running   9DFC6F7C

ACE: 1A0A81F8 Status: T   Req: 0007    Allied Chain
Authid: LEE0     Plan: SPPOOL   Corrid: ENTRLEE00039 Corrname: ITSOCIC3 Token: 0000755D
EB       Primary(Asid)  Home(Asid)     EBSPAWND TCB/SRB  -Status-- R14
1A0A8288 ITSOCIC3(0094) ITSOCIC3(0094) 00000000 009949E8 Running   9B2EB34E

ACE: 1A0AAA88 Status: T   Req: 007D    Allied Chain
Authid: LMA0     Plan: SPPOOL   Corrid: ENTRLMA00038 Corrname: ITSOCIC3 Token: 00007547
EB       Primary(Asid)  Home(Asid)     EBSPAWND TCB/SRB  -Status-- R14
1A0AAB18 ITSOCIC3(0094) ITSOCIC3(0094) 00000000 00994D48 Running   9DFC6F7C
ACE: 1C57E038 Status: T * Req: 012A    Allied Chain
Authid: RM01     Plan: SPPOOL   Corrid: ENTRRM010031 Corrname: ITSOCIC3 Token: 000073FE
EB       Primary(Asid)  Home(Asid)     EBSPAWND TCB/SRB  -Status-- R14
1C57E0C8 ITSODBM1(001E) ITSOCIC3(0094) 00000000 009A0B28 Suspended 9B2B1CD0
```

Figure 14-1 Example of IPCS verbx dsnwdmp 'sumdump=no,subsys=itso,ds=1'

The DB2 Storage Manager report using IPCS is generated using:

```
verbx dsnwdmp 'sumdump=no,subsys=ITSO,sm=1'
```

The IRLM Lock Manager data is obtained using the following IPCS command:

```
verbx dsnwdmp 'sumdump=no,subsys=ITSO,lm=1'
```

All data areas will be formatted if the following IPCS command is used:

```
verbx dsnwdmp 'sumdump=no,subsys=ITSO,all'
```

The DB2 global trace table can be formatted using the following IPCS command:

```
verbx dsnwdmp 'sumdump=no,subsys=ITSO,tt'
```

15

IMS diagnostic data collection

In this chapter we discuss the diagnostic data collection recommendations for IMS. This includes a discussion of the MVS requirements related to dumping and tracing the IMS address spaces.

This chapter describes the following:

- IMS diagnostic data
- IBM problem diagnosis
- IMS and the master trace table
- IMS DD statement and FMTO
- IMS tracing
- Dumping IMS address spaces
- IMS diagnostic data collection
- IMS dump formatting using IPCS
- IMS dump data set sizes
- IMS dump analysis

15.1 IMS diagnostic data

IMS consists of two licensed programs: the IMS Database Manager (DB) and the IMS Transaction Manager (TM). With the Database Manager, you can generate the batch environment and the database control (DBCTL) environment. With both the Database Manager and the Transaction Manager, you can generate the DB/DC environment. With the Transaction Manager, you can generate the data communication control (DCCTL) environment.

The DB/DC, DCCTL, and DBCTL environments are all considered online IMS systems.

The IMS environment consists of a number of address spaces. These include:

- **IMS control region**

  This is the address space holding the control program that runs continuously in this environment. It is normally started by using the MVS **START** command. The control region then automatically initiates the DBRC address space.

  The IMS control region owns all the databases that can be accessed by online application programs and is responsible for all physical input/output to the databases.

  The control region services all DL/I calls. It supervises the processing of messages and all the communication traffic for the connected terminals. The control region also manages information for restart and recovery purposes and operates the IMS system log.

- **IMS dependent regions**

  The dependent regions are initiated by an MVS **START** command or by a **/START REGION** command from the IMS master terminal.These include:

  – IMS MPP region

    The message processing program region used for processing messages. The control program schedules application programs within the MPP regions. The application programs then run, accessing the online databases and obtaining their transaction input from the message queues. The application programs cannot access z/OS files or issue z/OS checkpoints. The application program output messages can be directed to LTERMs or to other application programs. An application program can remain scheduled in an MPP region even when there is no work to process for that region.

    The MPP region remains in a wait-state (wait-for-input mode) until there is more work for the region to process.

  – IMS BMP region

    This is the batch message processing region, covered in detail in the next section.

15.1.1 Batch message processing region

The batch message processing region is used for processing batch operations. OS/390 schedules the BMP regions. The application programs in those regions are determined by the JCL used to start each region, not by the control region. These application programs can access databases owned by the control region and OS/390 data sets owned by their BMP regions. OS/390 data sets include data entry databases (DEDBs) and main storage databases (MSDBs).

Application programs in BMP regions can access input and output message queues; they can also execute in wait-for-input mode. To access the input message queues, you specify, in the JCL for a BMP region, a transaction code you want to access. Specifying this transaction code also gives you access to the output message queues using terminal program

communications blocks (PCBs) in the application program's specification block (PSB). Even without access to input message queues, if you specify an output LTERM or transaction code in the JCL for the region, the application program can issue output messages.

- **IMS IFP region**

  IMS Fast Path region used for processing Fast Path messages, these are called message-driven programs and utilities that process DEDBs; these are BMPs.

- **IMS JMP Region**

  Java™ Message Processing region, which processes messages with either applications written in Java or applications written in both Java and OO COBOL.

- **IMS JBP region**

  Java Batch Processing region, which processes batch operations with either applications written in Java or application written in both Java and OO COBOL.

- **IMS DBRC region**

  Database recovery control region used for controlling the logs and recovering the databases. It also controls the data sharing environment by allowing (or preventing) access to databases by various IMSs sharing those databases.

DBCTL environment

The DBCTL environment is similar to the DB/DC environment; a DL/I region owns the databases to be processed. DL/I also exists in the DBCTL environment, although DL/I must run in its own address space. Database Recovery Control (DBRC) facilities, required for DBCTL, help to manage database availability, data sharing, system logging, and database recovery.

The greatest dissimilarity between DBCTL and DB/DC is that DBCTL does not support user terminals, a master terminal, or message handling. Therefore, no MPP regions exist. The BMP region is used only by batch applications and utilities. External program subsystems can, however, use an interface that does handle messages, a coordinator controller (CCTL). The interface between the CCTL and the control region is the database resource adapter (DRA). The DRA resides in the same address space as the CCTL.

The CCTL handles message traffic and schedules application programs, all outside the DBCTL environment. It passes database calls through the interface to the control region, which sends the calls to DL/I and passes results back through the interface to the CCTL.

15.2 What must be kept to assist with IMS problem diagnosis

The following items should be kept for problem diagnosis:

1. The JES2 job logs for the IMS control region, IMS DLI/SAS region, IMS DBRC region, any suspicious IMS dependent regions and CQS regions.
2. The IMS master console log.
3. The SYS1.LOGREC data set. This data is also available using the IPCS **`VERBX LOGDATA`** command when reviewing an SDUMP.
4. All associated IMS dumps should be retained. SYSMDUMP for the IMS control, DLI/SAS and DBRC regions need to be reviewed in the case of SDUMP failures.
5. SYSUDUMPs should be saved for IMS dependent regions.
6. The IMS online log data set (OLDS) and IMS System Log Data Set (SLDS) should be saved.

15.3 IMS and the MVS system trace table

While this was discussed previously, it is a good idea to remind ourselves of the benefits of allocating a larger System Trace table than the MVS default of 64K.

The MVS System Trace table size can be set by issuing the MVS **TRACE ST,999K** command or by including this command in the COMMNDxx member is SYS1.PARMLIB. The benefit of setting this is SYS1.PARMLIB(COMMNDxx) ensures that the trace table is correctly sized at IPL time.

The system trace table will be included in the SVC dump provided the SDATA TRT option is specified.

15.3.1 IMS and the MVS master trace table

The master trace table is an instorage copy of the MVS SYSLOG (system log). The default size is 24K, which will hold approximately 336 messages. It is recommended to increase the size of the Master Trace Table to 500K, which will provide additional valuable data that precedes the abend.

The Master Trace Table size can be set via the MVS command, **TRACE MT,99K** or by inclusion in the SCHEDxx member of SYS1.PARMLIB.

15.3.2 IMS dump space recommendations

You must ensure that the MAXSPACE value assigned to SVC dump allocation is sufficient to hold the data for multiple address spaces. The default size is 500M. The recommended size for multiple address space dumps, for IMS and DB2, is 2500M. Failure to set a large enough size will result in only partial dumps being collected.

15.4 IMS dump DD statements and FMTO

In a DC environment, you can request the following types of dump outputs for errors that terminate IMS: SDUMP, SYSMDUMP, SYSABEND, or SYSUDUMP.

To do this, specify the FMTO startup parameter in combination with MVS dump DD statements.

For SYSMDUMP, you should provide operational procedures for saving and formatting dumps; otherwise, you can overlay a SYSMDUMP if you must restart IMS before the previous SYSMDUMP is transferred.

You can also request dump outputs for some errors that do not terminate IMS. Your choice of dump depends on several factors. These include, the type of failure, the FMTO parameter option, and the IMS spinoff and MVS dump DD statements that have been selected.

It is recommended that you specify, on the IMS Control region's EXEC parm, **FMTO=D**. This will produce an SVC dump for terminating and non-terminating errors. Non-terminating errors include IMS dynamic allocation failures and some ESAF failures.

A SYSMDUMP DD statement should be included in the IMS CTL Region, the IMS DLI/SAS Region, and the IMS DBRC Region. The SYSMDUMP specification will be used by IMS in the event that SDUMP processing should fail.

The following dump options should be specified in SYS1.PARMLIB(IEADMR00) to ensure that adequate MVS storage areas are dumped:

```
SDATA=(CSA,LSQA,RGN,SQA,SUM,SWA,TRT)
```

Include a SYSUDUMP DD statement for IMS dependent regions.

The following dump options should be specified in the SYS1.PARMLIB(IEADMP00) member:

```
SDATA=(CB,ERR,SUM) PDATA=(JPA,LPA,PSW,REGS,SA,SPLS)
```

15.5 IMS tracing

To activate IMS Dispatcher, Scheduler, DLI, and Lock tracing, specify the following options in IMS PROCLIB member DFSVSMxx:

```
DSP=ON,SCHD=ON,DL/I=ON,LOCK=ON
```

You could also issue the IMS **/TRA SET ON TABLE nnnn** command where nnnn is alternatively DISP, SCHD, DLI, and LOCK.

The LATCH trace should be active in non-production environments.

IMS external tracing allows for the IMS trace output to be placed in IMS trace data sets rather than the IMS Online Log Data Set. This is performed when DISP=OUT is used in the DFSVSMxx PROCLIC member and the LOG option is used with the IMS **TRACE** commands.

External trace data sets are allocated in the following order:

1. DASD JCL - DFSTRA01 and DFSTRA02 DDNAMES

2. DASD MDA - DFSTRA01 and DFSTRA02 Dynamic Allocation Members

3. TAPE MDA - DFSTRA0T Dynamic Allocation Member

4. IMS OLDS - If none of the above are found

15.5.1 Tracing the BPE and CQS in an IMS environment

The IMS Base Primitive Environment (BPE) is a common system service base upon which many other IMS components are built. BPE provides services such as tracing, message formatting, parsing, storage management, sub-dispatching, and serialization. In IMS Version 8, the following components use BPE:

- Common Queue Server (CQS)
- Operations Manager (OM)
- Resource Manager (RM)
- Structured Call Interface (SCI)

When an IMS component that uses BPE is started, the component loads a copy of the BPE service modules into its address space from the IMS V8 program libraries. The IMS component's modules are specific to that component; however, the BPE service modules are common across the various address spaces. The base system service functions are therefore identical in every address space that uses BPE.

Common Queue Server

The Common Queue Server (CQS) is a generalized server that manages data objects on a coupling facility list structure, such as a queue structure or a resource structure, on behalf of multiple clients. CQS receives, maintains, and distributes data objects from shared queues on behalf of multiple clients. Each client has its own CQS access to the data objects on the coupling facility list structure. IMS is one example of a CQS client that uses CQS to manage both its shared queues and shared resources.

Figure 15-1 shows the CQS tracing options within the BPECFG=nnnnnnnn Proclic member:

```
TRCLEV=(AWE,LOW,BPE)/* AWE Server Trace   *
TRCLEV=(CBS,LOW,BPE)/* CONTROL BLK SRVCS TRACE *
TRCLEV=(DISP,LOW,BPE)/* DISPATCHER TRACE*
TRCLEV=(LATC,LOW,BPE)/* LATCH TRACE*
TRCLEV=(SSRV,LOW,BPE)/* GEN SYSTEM SERVICES TRACE*
TRCLEV=(STG,LOW,BPE)/* STORAGE TRACE*
TRCLEV=(USRX,LOW,BPE)/* USER EXIT TRACE*
TRCLEV=(CQS,LOW,CQS)/* CQS GENERAL TRACE*
TRCLEV=(STR,LOW.CQS)/* CQS STRUCTURE TRACE*
TRCLEV=(INTF,LOW,CQS)/* CQS INTERFACE TRACE*
```

Figure 15-1 CQS trace entry parameters

15.5.2 IMS APPC application program tracing

To turn on program tracing for **TPPCB DL/1** calls do the following:

1. Issue the command

   ```
   /TRACE SET ON PROGRAM pppppppp
   ```

   pppppppp is the application program name
2. Turn on the MVS APPC component trace as follows:

   ```
   TRACE CT,ON,200K,COMP=SYSAPPC
   ```
3. Reply to the WTOR message as follows:

   ```
   nn,OPTIONS=(GLOBAL),END
   ```
4. When the problem has been recreated, stop the CTRACE as follows:

   ```
   TRACE CT,OFF,COMP=SYSAPPC
   ```
5. The APPC component trace sends its trace buffers to the SYS1.DUMP data set. Use the following IPCS commands to format the trace:

   ```
   CTRACE COMP SYSAPPC SHORT
   CTRACE COMP SYSAPPC FULL
   ```

15.5.3 IMS TPIPE and OTMA traces

From the IMS - OTMA side, be sure to capture both the TPipe and OTMA traces in order to have as much data as possible for any future diagnostic work. Issue the following:

```
/TRAce SET ON TMEMBER tmembername TPIPE ALL
/TRAce SET ON TABLE OTMT OPTION LOG
```

15.6 Simplify the dump process for multiple address spaces

To *automate*, or at the least provide a more streamlined approach to dumping the related address spaces that are part of the IMS environment, the following process can be used.

Figure 15-2 and Figure 15-3 show the IEASLPxx members in SYS1.PARMLIB containing the related dump and jobname information.

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),A=(SYNCSVCD,TARGETID),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ),
JOBLIST=(job1,job2,job3,job4),ID=IMS1,TARGETID=(IMS2),D,END
```

Figure 15-2 IEASLPxx #1 example for IMS address spaces

In Figure 15-2 the parameters are:

- job1 = IMS Control Region Jobname
- job2 = IMS DLI Region Jobname
- job3 = DBRC Region Jobname
- job4 = IRLM Region Jobname (If IRLM DB Locking is used)

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),
JOBLIST=(job5,job6,job7),ID=IMS2,
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT),D,END
```

Figure 15-3 IEASLPxx #2 example for IMS address spaces

In Figure 15-3 the parameters are:

- job5 = CCTL Region 1
- job6 = CCTL Region 2
- job7 = CCTL Region 3

Before activating the SLIP, be sure that any existing PER SLIP is disabled by issuing:

```
SLIP MOD,DISABLE,ID=trapid
```

To activate the SLIP trap and trigger the associated SVC dumps. enter the following MVS commands:

```
SET SLIP=xx
SLIP MOD,ENABLE,ID=IMS1
```

After these two commands are entered, two dump data sets are created on the MVS image from which the SLIP command was entered.

15.7 Dumping IMS address spaces in a sysplex

Figure 15-4 and Figure 15-5 show the SYS1.PARMLIB members called IEADMCI1 and IEADMCI2 containing the DUMP parameters defined following the figures.

```
JOBNAME=(job1,job2,job3,job4),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ),
REMOTE=(SYSLIST=*('job1','job2','job3','job4'),SDATA)
```

Figure 15-4 IEADMC I1example for IMS sysplex dumps

In Figure 15-4 the parameters are:

- job1 = IMS Control Region Jobname
- job2 = IMS DLI Region Jobname
- job3 = DBRC Region Jobname
- job4 = IRLM Region Jobname (If IRLM DB Locking is used)

Create a second SYS1.PARMLIB member called IEADMCI2.

```
JOBNAME=(job5,job6,job7),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ,XESDATA),
REMOTE=(SYSLIST=*('job5','job6','job7'),SDATA)
```

Figure 15-5 IEADMC I2example for IMS sysplex dumps

In Figure 15-5 the parameters are:

- job5 = CCTL Region 1
- job6 = CCTL Region 2
- job7 = CCTL Region 3

To request a dump to be captured as per the IEADMCI1 and IEADMCI2 parmlib members, issue the following MVS command:

```
DUMP TITLE=(IMS/CCTL sysplex DUMPS),PARMLIB=(I1,I2)
```

Two dump data sets are created on each MVS image in the sysplex matching the REMOTE specifications for the JOBNAMEs.

Alternatively, you can use IEASLPxx containing the SLIP entries as show in Figure 15-6 and Figure 15-7.

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),A=(SYNCSVCD,TARGETID),
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,GRSQ),
JOBLIST=(job1,job2,job3,job4),ID=IMS1,TARGETID=(IMS2),
REMOTE=(JOBLIST,SDATA),D,END
```

Figure 15-6 IEASLPxx #1 for IMS sysplex dump

In Figure 15-6 the parameters are:

- job1 = IMS Control Region Jobname
- job2 = IMS DLI egion Jobname
- job3 = DBRC Region Jobname
- job4 = IRLM Region Jobname (If IRLM DB Locking is used)

```
SLIP SET,IF,N=(IEAVEDS0,00,FF),
JOBLIST=(job5,job6,job7),ID=IMS2,
SDATA=(CSA,PSA,RGN,SQA,SUM,TRT,XESDATA),
REMOTE=(JOBLIST,SDATA),
D,END
```

Figure 15-7 IEASLPxx #2 for IMS sysplex dump

In Figure 15-7 the parameters are:

- job5 = CCTL Region 1
- job6 = CCTL Region 2
- job7 = CCTL Region 3

Before activating the SLIP, ensure that any existing PER SLIP for each MVS image in the sysplex is disabled by issuing:

```
ROUTE *ALL,SLIP,MOD,DISABLE,ID=trapid
```

To activate the SLIP trap and trigger the associated SVC dumps, enter the following MVS commands:

```
SET SLIP=xx
SLIP MOD,ENABLE,ID=IMS1
```

Two dumps will then be captured on each MVS image in the sysplex matching the REMOTE specifications.

15.8 IMS diagnostic data collection for WAIT/HANG conditions

The difficulty in capturing specific data related to a wait or hang in a multi-address space environment is knowing which address spaces to dump since it is often difficult to determine who is actually causing the wait/hang. Often people misinterpret a loop for a hang, and while meaningful processing in both cases is not possible, the key indicator is the lack of CPU usage. A hang will usually be confirmed by no CPU activity, or no I/O activity; whereas a loop will often show high, and even excessive CPU activity.

While it is necessary to capture a dump of primary IMS address spaces (preferably using the previously discussed procedures), the wait/hang could be related, for example, to VTAM, APPC, or ESAF, which could be interfacing with DB2 or MQSeries.

15.8.1 IMS diagnostic data collection for a suspected Loop

If the IMS Control, DLI, DBRC or Dependent region enters a loop state, the following should be performed:

- Set the MVS trace table size to 999K and turn Branch trace on, as follows:

  ```
  TRACE ST,999K,BR=ON
  ```

- Turn on IMS tracing as follows:

  ```
  /TRA SET ON TABLE nnnn-
  ```

  where nnnn = DISP, SCHD, DLI, LOCK and LATCH

- Dump the related IMS address spaces as previously discussed.

15.8.2 IMS APPC diagnostic data capture procedures

To capture diagnostic data, do the following:

- Turn on IMS LUMI tracing to the external trace data set as follows:

```
/TRACE SET ON TABLE LUMI OPTION LOG
/TRACE SET ON LUNAME xxxxxxxx INPUT
/TRACE SET ON LUNAME xxxxxxxx OUTPUT
```

  In this example xxxxxxxx is the partner LU.

- Turn on VTAM Buffer trace and VTAM Internal Trace to complement the IMS LUMI trace.

> **Note:** GTF must be active with the USR option.

```
F NET,TRACE,TYPE=BUF,ID=nodename
F NET,TRACE,TYPE=VTAM,MODE-EXT.OPT=(API,PUI,MSG)
```

- Dump the related IMS regions as well as the VTAM and APPC address spaces.

15.9 IMS dump formatting using IPCS

IMS dumps can be formatted using the IPCS IMS dump formatting utility. This is reached from the IPCS Primary menu by making the following selections: **2 - Analysis → 6, COMPONENT - MVS component data → DFSAAMPR IMS Interactive Dump Formatter** option. Figure 15-8 shows the IPCS IMS dump formatting primary menu.

```
---------------------- IMS DUMP FORMATTING PRIMARY MENU --------------------
OPTION  ===>

   0  INIT         - IMS formatting initialization and content summary
   1  BROWSE       - Browse Dump dataset                  ******************
   2  HI-LEVEL     - IMS Component level formatting       *USERID  - ITSO131
   3  LOW-LEVEL    - IMS ITASK level formatting           *DATE    - 03/07/08
   4  ANALYSIS     - IMS dump analysis                    *JULIAN  - 03.189
   5  USER         - IMS user formatting routines         *TIME    - 15:10
   6  OTHER COMP   - Other IMS components (BPE, CQS...)   *PREFIX  - U143958
   7  OTHER PROD   - Other IMS-related products           *TERMINAL- 3278
   E  EDA          - IMS Enhanced Dump Analysis           *PF KEYS -
   T  TUTORIAL     - IMS dump formatting tutorial         ******************
   X  EXIT         - Exit IMS dump formatting
 Enter END or RETURN command to terminate IMS component formatting.
 Use PFKeys to scroll up and down if needed.
```

Figure 15-8 IMS IPCS Dump Formatting Primary Menu

IMS dump data can also be formatted using the IMS IPCS VERBEXIT command as follows:

```
verbx imsdump 'imsjobname FMTIMS formatoption'
```

In this case, `imsjobname` is the job name or started task name of either the IMS CTL, DL/I, or the IMS batch address space.

15.9.1 IMS VERBX format option

The IMS VERBEXIT has high-level and low-level format options. The high-level options will give you the best summary data, whereas the low-level options will be more detailed, for an IMS internal control blocks data review.

Figure 15-9 show the IMS VERBX low-level format option setting.

```
CBTE,cbteid
CLB,address or CLB,nodename or CLB,lterm name or CLB, comm id
DPST,address or DPST, number or DPST,name
LLB,link number
LUB,lu name
POOL,poolid or POOL,poolid,MIN
SAP,sapaddr or SAP,ecbaddr
SYSPST,system pst address or SYSPST,system pst name
TRACE,name or TRACE,name,MIN
```

Figure 15-9 IMS VERBX low-level format options

Figure 15-10 on page 143 shows the IMS VERBX high-level format option settings.

:

```
ALL or ALL,MIN
AUTO, or AUTO,MIN, or AUTO,SUM
CBT
DB or DB,MIN
DBRC
DC or DC,MIN
DEDB or DEDB,MIN
DISPATCH or DISPATCH,MIN
EMH or EMH,MIN
LOG or LOG,MIN
LUM
MSDB or MSDB,MIN
QM or QM,MIN
RESTART
SAVEAREA, or SAVEAREA,MIN or SAVEAREA,SUM
SB or SB,MIN
SCD or SCD,MIN
SPST
SUBS
SUMMARY or SUMMARY,MIN
SYSTEM or SYSTEM,MIN
UTIL
```

Figure 15-10 IMS VERBX high-level format options

These formatting options can all be accessed via the IPCS IMS DUMP FORMATTING primary menu.

16

VTAM diagnostic procedures

This chapter describes how details about many VTAM problems can be identified by issuing some relevant VTAM **DISPLAY** commands that can identify problems for devices, lines, major nodes, cross domain resources and buffers, and so forth.

This chapter describes the following:

- VTAM diagnostic commands
- VTAM IPCS dump formatting
- VTAM internal trace

16.1 VTAM diagnostic commands

Some useful commands are:

- **D NET,PENDING**, which displays resources that are in a pending status. This could indicate the resource that is causing a hang during processing or shutdown.
- **D NET,CDRMS** displays the cross-domain resources.
- **D NET,ID=name** displays the status of the named device.
- **D NET,BFRUSE** displays the VTAM buffer statistics.
- **V NET,ACT,ID=name** will activate the named resource.
- **V NET,INACT,ID=name** will inactivate the named resource.

Messages generated by VTAM and more importantly, the sense data that accompanies these messages, can be invaluable when diagnosing problems. Figure 16-1 shows a sample VTAM error message.

The error condition identified in the VTAM error message is obvious, and activating resource SC54ALUC would be a good place to start. If activating this device was unsuccessful, then the sense data provided could be used to identify the cause.

```
IST663I IPS SRQ REQUEST TO ISTAPNCP FAILED, SENSE=08570003 621
IST664I REAL  OLU=USIBMSC.SC55ALUC    REAL  DLU=USIBMSC.SC54ALUC
IST889I SID = ED03979A6B8E83B5
IST264I REQUIRED RESOURCE SC54ALUC NOT ACTIVE
IST891I USIBMSC.SC54M GENERATED FAILURE NOTIFICATION
IST314I END
```

Figure 16-1 VTAM error message example

Alternatively, the more complex VTAM problems can require a dump of the VTAM address, a dump of the NCP, and a VTAM internal trace.

A dump of the NCP should be taken whenever the NCP abnormally terminates or when an error is suspected in the NCP. It may be possible to determine that a problem exists in the NCP by using the VTAM I/O trace to determine what PIUs are being sent to and received from the communication control and by using the NCP line trace to determine what is happening on the lines between the communication controller and the link-attached logical unit.

Dumping the Network Control Program (NCP) can be started as follows:

```
F vtam_procname,DUMP,ID=ncp_name
```

16.1.1 First failure support technology (FFST) for VTAM

First failure support technology is a licensed program that captures information about a potential problem when it occurs. When a problem is detected, a software probe is triggered by VTAM. FFST then collects information and based on the options active for the probe, you get a dump and a generic alert. You will either get a full dump, an FFST minidump (partial dump), or both. If the VTAM internal trace data space is present a full dump will be triggered.

The full dump is taken via the SDUMP macro instruction to provide a full dump of the address space where the potential problem occurred and includes selected MVS control blocks, CSA, ECSA subpools (227, 228, 231, and 241), the PSA, and the VTAM VIT data space if present.

The VIT data space is present if the trace is written using the MODE=INT option. This is discussed later in this chapter.

A FFST minidump contains general purpose registers, selected VTAM control blocks, and the ECSA VIT table; it can be formatted using the EPWDMPFM CLIST. EPWDMPFM formats your minidump and writes it to a data set that you can view online or print using the IEBPTPCH utility program. (FFST minidumps cannot be processed using the IPCS VTAM formatter.)

16.2 VTAM IPCS dump formatting

A VTAM full dump, often referred to as an SVC dump, needs to be formatted using IPCS and the supplied VTAM formatting utilities. These include IPCS VERBEXIT functions and also some specific IPCS CLIST routines that are also invoked from the IPCS sub-command entry panel and allow you to interrogate the VTAM control block structures and the VTAM Internal Trace data.

The IPCS VTAM CLIST routine, ISTVDUMP shows the SDATA options that were in effect when the dump was taken, as shown in Figure 16-2.

```
CLIST ISTVDUMP STARTED AT 09:02:03.

SDATA OPTIONS REQUESTED FOR THIS DUMP:

  SDUALPSA - DUMP ALL PSA'S IN THE SYSTEM
  SDUSQA   - DUMP SQA
  SDULSQA  - DUMP LSQA
  SDURGN   - DUMP REGION (PRIVATE AREA)
  SDULPA   - DUMP ACTIVE LPA MODULE FOR RGN
  SDUTRT   - DUMP TRACE TABLE / GTF BUFFERS
  SDUCSA   - DUMP CSA
  SDUSWA   - DUMP SWA FOR REGION
  SDUSMDMP - SUMMARY DUMP REQUESTED
  SDUALNUC - DUMP ALL NUCLEUS AREAS

CLIST ISTVDUMP ENDED AT 09:02:03. RETURN CODE = 0.
```

Figure 16-2 IPCS ISTVDUMP

Figure 16-3 shows a subset of the data returned from the IPCS **VERBX VTAMMAP 'VTBUF'** command.

```
VTBUF Analysis

                        IO Buffer Analysis

Size of Buffers(bytes)          515   Buffer maximum                182
Total buffers available         137   Static buffers allocated      182
Total number of buffers         182
Buffers in use (%)               24   Available static buffers      137
Bytes in static &
   expanded areas             93730
Slowdown threshold               19   Expansion threshold            48
Contraction threshold         32767
Number of expansions              0   Expansion increment            14
Expansion size                 8192
Total queued RPHs                 0
Fixed or pageable?        FIXED
Buffer pool address            X'215CC558'
Beginning address of pool      X'215B2000'
Ending address of pool         X'215CC000'
Buffer pool has no extensions
```

Figure 16-3 IPCS VERBX VTAMMAP 'VTBUF'

Figure 16-4 shows a sample of the VTAM internal trace formatted via the IPCS **VERBX VTAMMAP 'VTBASIC'** command.

```
VTBASIC Analysis
INTERNAL TRACE TABLE    2103E000

ENT WRAP  BCEB6019FDDA7060  LAST WRAP    BCEB5BA88E9A3B60

ENT ENTRY 21043280          LAST ENTRY   21424FE0
DSP         ASID 1A   CBID  00       FLAG  88       FLAG1 10       LEVEL 00
                                                    LAST  00000000 WEA   00000
SIOX        ASID 1A   STATE 0B       CCWOC 00       TYPE .   CUA   2CA1     NC
                                     CAW   20675050 DATA  00010000 00000000
SIO2        BUF 43020000  MODID IO   DATA  00000000 00000000 00000000 00000000
EXIT        ASID 1A   APNOP 00       PABOF 0040     PST   21642958 PAB   20E65
                                                    WEQ   80000000 NAME  TS8S
RELS        ASID 1A   CBID  25       PST   21642958 BUF   21513010 ISSR  A17B1
                                                    REG1  21513010
INTX        ASID 1A   STATE 0B       ENDOP 00       TYPE .   CUA   2CA1     NC
                                     CODE  7F       SENSE 0000     CSW   20675
INT2        DATA 00000000 00000000 00000000 00000000
SCHD        ASID 1A                  FLGS  280810   PST   21642958 PAB   20E65
                                                    WEQ   00000000 NAME  TSUC
SCHD        ASID 1A                  FLGS  280810   PST   21642958 PAB   20E65
                                                    WEQ   00000000 NAME  TS8S
REQS        ASID 1A   CBID  25       PST   21642958 BUF   21513010 ISSR  A17AA
                                                    REG1  214FDAD0 RC    0000
DSP         ASID 1A   CBID  00       FLAG  88       FLAG1 10       LEVEL 00
                                                    LAST  00000000 WEA   00000
```

Figure 16-4 IPCS VERBX VTAMMAP 'VTBASIC' sample

Another really useful command to assist with finding the maintenance level and location of a specific VTAM module is the IPCS VERBX **VTAMMAP 'VTMODS LIST(Y)'** command.

Figure 16-5 shows a sample of the data returned from the **VTMODS LIST(Y)** command.

```
22800008    ISTPDCUP    03.257      VTXAX16
228000D8    ISTNOCTM    03.257      VTXAX16
228003E0    ISTINCLQ    03.257      VTXAX16
22800628    ISTINM01    04.208      UA12672
22802020    ISTLUCQD    2003.2
22802408    ISTCPCQD    03.257      VTXAX16
22802A60    ISTPUCQD    2003.2
22802AD0    ISTNOCPR    04.334      UA14823
2280B778    ISTINCTR    04.239      UA13446
2280E910    ISTORCEJ    04.203      UA12565
22814888    ISTIECIN    04.204      UA12599
22815100    ISTTRRIR    04.208      UA12672
228152D0    ISTNACNI    04.131      UA10727
22815670    ISTNACNT    2004.1
228158D8    ISTCALOD    03.255      VTXAX16
22815AF0    ISTCICDF    03.257      VTXAX16
228163C8    ISTCICIC    03.257      VTXAX16
```

Figure 16-5 IPCS VERBX VTAMMAP 'VTMODS LIST(Y)' example

An expansion on the previous **VTMODS** command is the **VTFNDMOD** command, which will return additional data related to a specific module as shown in Figure 16-6.

```
VTFNDMOD SYMBOL(ISTNOCPR)
VTFNDMOD Analysis
Module name:                ISTNOCPR
Compile date:               04.334
PTF Number:                 UA14823

Address entered:            22802AD0
Module entry point:         22802AD0
                            --------
Displacement into module:          0

First '40'X bytes of module:
  DATA: 22802AD0
     +0000  A7F40078  011417C9  E2E3D5D6  C3D7D940  | x4.....ISTNOCPR  |
     +0010  F0F44BF3  F3F440E4  C1F1F4F8  F2F3A7F4  | 04.334 UA14823x4 |
     +0020  00690100  A7F40066  0194A7F4  006302D8  | ....x4...mx4...Q |
     +0030  A7F40060  03D4A7F4  005D04A8  A7F4005A  | x4.-.Mx4.).yx4.! |

 Storage around address entered:

  DATA: 22802ABC
     +0000  00000000  00000000  00000000  81262900  | ............a... |
     +0010  07000000  A7F40078  011417C9  E2E3D5D6  | ....x4.....ISTNO |
     +0020  C3D7D940  F0F44BF3  F3F440E4  C1F1F4F8  | CPR 04.334 UA148 |
     +0030  F2F3A7F4  00690100  A7F40066  0194A7F4  | 23x4....x4...mx4 |
```

Figure 16-6 IPCS VTAMMAP 'VTFNDMOD SYMBOL(ISTNOCPR)' example

The **VTAMMAP HOST** command will return information as shown in Figure 16-7.

```
 HOST

                              HOST Analysis

NetID            USIBMSC
ASID (Hex)          001A
ASID (Dec)            26
Subarea (Hex)   00000001   Element  0001
Subarea (Dec)          1   Element     1
CDRM Name           VTAM

This SSCP is not gateway capable
This CDRM supports dynamic CDRSCs
CP Network address  00000001 0006
CP Name             USIBMSC.SC48M
This is a pure end node
```

Figure 16-7 IPCS VERBX VTAMMAP 'HOST' command example

16.2.1 VTAMMAP procedure

Another very valuable VTAMMAP procedure is SIBCHECK. The SIB indicates the status of an LU-LU session. Figure 16-8 on page 151 shows the result of the following IPCS command to format the SIB data:

```
verbx vtammap 'sibcheck ADDR(x''230E5060'')'
```

The SIB address was found by using the IPCS **VERBX VTAMMAP 'ALL'** procedure and going to the end of the formatted output and issuing a find for the specific LU using the PREV option.

Note: The SIBCHECK hex address (ADDR) must be enclosed in 2 pairs of single quotes.

Although not shown in the example in Figure 16-8, the formatted display contains additional data specific to the settings related to the LU. For example:

```
Analysis of resource extension for DLU|PLU S48TOS52 at address 230E5170

SIBRNETC = 20 = SIBRNTEP - Endpoint
SIBRCDTC = 0 - Session has not been associated with the CDTAKEDOWN COMPLETE
               RU for the CDRM specified in SIBRADJN
SIBRLUMA = 1 - LU address in my network added
SIBRSSE  = 1 - LU supports SESSEND
SIBRLDOM = 1 - LU is in this domain
SIBRQNE  = 0 - Do not queue session if the LU is not enabled
SIBRQSLM = 0 - Do not queue session if the LU is at session limit

 Analysis of resource extension for OLU|SLU S52TOS48 at address 230E5110

SIBRNETC = 30 = SIBRNTXD - Cross domain
SIBRCDTC = 0 - Session has not been associated with the CDTAKEDOWN COMPLETE
               RU for the CDRM specified in SIBRADJN
SIBRLUMA = 1 - LU address in my network added
```

```
SIBRSSE  = 1 - LU supports SESSEND
SIBRLDOM = 0 - LU is not in this domain
SIBRQNE  = 0 - Do not queue session if the LU is not enabled
SIBRQSLM = 0 - Do not queue session if the LU is at session limit
```

SIBCHECK analysis

An example of a SIBCHECK analysis is shown in Figure 16-8.

```
SIBCHECK ADDR(X'230E5060')
SIBCHECK Analysis

                                CDRM: VTAM
                               NetID: USIBMSC
                     Network address: 00000001 0001
                         SIB address: 230E5060

          OLU|SLU                                DLU|PLU
 Name (from RDTE): S52TOS48           Name (from RDTE): S48TOS52
     RDTE address: 230EF050               RDTE address: 2305D9B4
        RDTE type: CDRSC                    RDTE type: APPL
      Owning CDRM: SC48M                  Owning CDRM: VTAM
            NetID: USIBMSC                      NetID: USIBMSC
       Alias name: S52TOS48                Alias name: S48TOS52
      Alias netID: USIBMSC                Alias netID: USIBMSC
    Adjacent SSCP: ISTAPNCP              Adjacent SSCP: ........
  Network address: 00000001 003A       Network address: 00000001 0104

 SIBFSMIN: FC = SIBIFSAC - Session Active
 SIBFSMTM: 00 = SIBTFSIS - Initial state
 SIBB_ALS_ACT_FSM: A0 = SIBAFSAA - Pending APPN LU address assignment

                              Analyze SIB Base

                 Original PCID (SIBPCID): ED0385CAB3495ECA
           Timestamp from SIB (SIBBTIME): BCEA3FB57910EE27
          Converted timestamp (SIBBTIME): 04/25/05  14:33:55.628302
   Primary SIB queue elements (SIBBPRIQ): 0
 Secondary SIB queue elements (SIBBSECQ): 0
```

Figure 16-8 verbx vtammap 'sibcheck ADDR(x"230E5060")' example

16.3 VTAM internal trace (VIT)

Most VTAM traces show the information flow between the VTAM program and other network components. However, the VTAM internal trace (VIT) provides a record of the sequence of events within VTAM. These internal events include the scheduling of processes (for example, POST, WAIT, and DISPATCH), the management of storage (for example, VTALLOC), and the flow of internal PIUs between VTAM components. Together with the operator console listing and a dump, output from the VIT can help you reconstruct sequences of VTAM events and find internal VTAM problems more easily.

Trace data for the following VIT options is always automatically recorded in the internal table:

- API - Application program interfaces
- CIO - Channel input and output
- MSG - Messages
- NRM - Network resource management
- PIU - Path information unit flows
- SSCP - System services control point request scheduling and response posting

Use one of the following methods to start the VIT:

- Use the TRACE start option, with TYPE=VTAM specified, to start the VIT when you first start VTAM.
- Use the **MODIFY TRACE** command, with TYPE=VTAM specified, to start the VIT after you have started VTAM.

To prevent the VIT table from being overwritten, VTAM disables the internal VIT when it issues SDUMP and when an FFST™ probe is tripped. The minimum trace table size is 100 pages, and because the five trace option defaults are always running, the table may wrap many times.

Both the TRACE start option and the **MODIFY TRACE** command have an OPTION operand you can use to select VIT options.

VTAM can write the VIT trace data to an internal table or an external device, such as a disk or tape. You specify internal or external with MODE operand of the TRACE start option or the **MODIFY TRACE** command. The VIT record contains the same information regardless of the mode selected.

You can record data externally and internally at the same time, and if desired, you can have different sets of trace options active for each mode. The default trace options API, MSG, NRM, PIU, and SSCP are always recorded internally.

16.4 Recording traces in the internal table (MODE=INT)

If you set MODE=INT on the **MODIFY TRACE** command or as a TRACE start option, or if you let MODE default to INT, VTAM writes the VIT trace records in an internal trace table. The table is allocated and initialized in extended common service area (CSA) storage.

The SIZE operand of the TRACE start option specifies the number of pages (1 through 999) in storage to be allocated for the internal trace table. Each page is 4 KB. If you omit this option, the default size is 100 pages. If you specify fewer than 100 pages, VTAM uses 100. Because it is a wraparound table, specify enough pages to ensure that the VIT will not overwrite important trace records when the table fills and begins to wrap around.

16.5 Recording traces in the external table (MODE=EXT)

When you specify MODE=EXT, information is still written to the internal trace table for the default options. The external trace file is produced by GTF, and the default file name is SYS1.TRACE. You can print the internal trace data with IPCS or TAP. If you use IPCS to print the data, specify the GTFTRACE option, and set USR(FE1).

16.6 Module names in the internal trace records

Many VTAM internal trace records include the associated module names in EBCDIC, without the first prefix and, for some types of trace records, without the sixth letter. For example, you would see TSSR for module ISTTSCSR. You can save time by scanning for these module names when you are following the logic flow through VTAM. You can sometimes isolate a VTAM problem to a specific component or module without even looking at a dump. Module names can also be determined from the ISSR field in some VIT records.

Note: Additional information related to VTAM problem diagnosis can be found in:

- *z/OS V1R6.0 CS: SNA Messages*, SC31-8790
- *z/OS V1R6.0 CS: IP Diagnosis Guide,* GC31-8782-05
- *z/OS V1R6.0 CS: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS V1R6.0 CS: IP Messages Volume 2 (EZB, EZD),* SC31-8784
- *z/OS V1R6.0 CS: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS V1R6.0 CS: IP Messages Volume 4 (EZZ, SNM™)*, SC31-8786
- *z/OS V1R6.0 CS: IP and SNA Codes*, SC31-8791
- *z/OS Communications Server SNA Diagnosis Volume 1: Techniques and Procedures* LY43-0088
- *z/OS Communications Server SNA Diagnosis Volume 2: FFST Dumps and the VIT* LY43-0089

17

TCP/IP component and packet trace

This chapter describes the TCP/IP packet trace, which contains all outgoing and incoming packets for a specified packet option, broken down into headers and trailers. It is very useful for investigating problems with FTP or other applications that transfer data using TCP or UDP.

This chapter describes the following:

- Tracing the TCP/IP data space
- PKTTRACE parameters
- Tracing to an external writer

17.1 Tracing to the TCP/IP data space

The first step to collecting traces to the data space is to ensure that the bufsize in CTIEZB00 in parmlib is set to at least 8Mb. It may need to be set higher depending on the amount of trace data desired, but 8Mb should be a good starting point. TCPIP will need to be restarted for the change in bufsize to take affect.

The starting writer step can be skipped because it will not be needed when writing to the data space. The trace data will be captured via an MVS **Dump Comm** command that will dump the TCPIP data space named TCPIPDS1. Be aware that this method may result in lost trace data since the possibility of wrapping is very possible. The dump command should be issued very soon after the problem happens or the dumps should be collected via a trap or slip given by the support center, as follows:

1. Start CTRACE COMP(SYSTCPIP)

   ```
   TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpipprocname)
   R XX,OPTIONS=(XXX,XXX),END
   ```

2. Recreate the problem.
3. Stop the CTRACE for COMP(SYSTCPIP):

   ```
   TRACE CT,OFF,COMP=SYSTCPIP,SUB=(tcpipprocname)
   ```

4. Start packet trace COMP(SYSTCPDA):

   ```
   TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpipprocname)
   ```

   Reply to the following message:

   ```
   ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND
   r xx,end
   ```

5. This step starts the packet trace processing in TCPIP. Without this step packets cannot be captured by the ctrace component.

   ```
   V TCPIP,tcpipprocname,PKT,ON,IP=xx.xx.xx.xx
   ```

6. Recreate the problem.
7. Stop the packet trace for COMP(SYSTCPDA):

   ```
   V TCPIP,tcpipprocname,PKT,OFF,IP=xx.xx.xx.xx
   TRACE CT,OFF,COMP=SYSTCPDA,SUB=(tcpipprocname)
   ```

8. Collect the dump of the TCPIP data space and the TCPIP address space by issuing the following command from the MVS console:

   ```
   DUMP COMM=('text')
   R xx,JOBNAME=(tcpipprocname),DSPNAME=('tcpipprocname'.*),
   SDATA=(ALLNUC,CSA,LPA,LSQA,RGN,SWA,SQA,TRT),END
   ```

17.2 PKTTRACE parms

There are 2 ways to add parms to the SYSTCPDA CTRACE (PKTTRACE). The first way will AND the parms together and the second way will OR the parms together, as follows:

- **V TCPIP,tcpipproc,PKT,SRCP=21,DEST=1236**

  In this example only packets with source port=21 *and* destination port=1236 will be traced.

- **V TCPIP,tcpipproc,PKT,SRCP=21**
  V TCPIP,tcpipproc,PKT,DEST=1236 *note 2 VARY commands*

  In this example packets with a source port=21, regardless of dest port, *and* packets with dest port=1236, regardless of source port will be traced.

17.3 Tracing to the external writer

To have packet trace or CTRACE data written to an external writer data set, a writer proc first needs to be created. This procedure must either be in SYS1.PROCLIB or in a library concatenated in the master JCL. The following is a sample of the writer procedure:

```
//CTWTR1 PROC
//IEFPROC  EXEC PGM=ITTTRCWR
//TRCOUT01 DD DSNAME=IBMUSER.CTRACE1,VOL=SER=xxxxxx,
//         UNIT=xxxxx,SPACE=(CYL,(xxx),,CONTIG),
//         DISP=(NEW,CATLG)
//SYSPRINT DD SYSOUT=*
```

17.3.1 Starting an external writer

If you do not want to have traces written internally to the TCPIP dataspace (TCPIPDS1) you can start the external writer to the CTRACE component. This writer can be used for multiple components, for instance packet trace (SYSTCPDA) and ctrace (SYSTCPIP). The following command starts the external writer:

```
TRACE CT,WTRSTART=CTWTR1
```

Once the writer has been successfully attached you can proceed with starting the traces you need to run.

If you want to run a CTRACE and PACKET trace using the same writer proceed to the multiple trace step. If packet alone is required proceed to Packet Trace Step. If CTRACE (SYSTCPIP) is required go to the CTRACE STEP.

PKTTRACE step (component SYSTCPDA)

The following steps start the external writer and the trace:

1. Start CTRACE and give it a component to use for tracing. The required reply attaches the external writer that was previously started so it can be used to write the packet trace records.

   ```
   TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpipprocname)
   R xx,WTR=CTWTR1,END
   ```

2. Verify that the trace started successfully:

   ```
   D TRACE,COMP=SYSTCPDA,SUB=(tcpipprocname)
   ```

3. Start the packet trace processing in TCPIP. Without this step packets cannot be captured by the CTRACE component.

   ```
   V TCPIP,tcpipprocname,PKT,ON,IP=xx.xx.xx.xx
   ```

4. Recreate the problem.

5. Stop the packet trace:

   ```
   V TCPIP,tcpipprocname,PKT,OFF
   ```

6. Disconnect the external writer:

```
TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpipprocname)
R xx,WTR=DISCONNECT,END
```

7. Stop the external writer:

```
TRACE CT,OFF,COMP=SYSTCPDA,SUB=(tcpipprocname)
TRACE CT,WTRSTOP=CTWTR1,FLUSH
```

17.3.2 CTRACE step (component SYSTCPIP)

The following steps are for the SYSTCPIP component:

1. Start CTRACE and give it a component to use for tracing. The required reply attaches the external writer that was previously started so it can be used to write the CTRACE records.

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpipprocname)
R xx,WTR=CTWTR1,OPTIONS=(XXXX,XXXX),END
```

Note: For certain problems you should use the jobname of the application when running CTRACE(SYSTCPIP). Trace options that should use jobname are (PFS,SOCKET,ENGINE,TCP). Valid options are contained in parmlib member CTIEZB00.

2. Verify that the trace was started successfully:

```
D TRACE,COMP=SYSTCPIP,SUB=(tcpiprocname)
```

3. Recreate the problem.
4. Stop CTRACE COMP(SYSTCPIP) and disconnect the writer:

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpipprocname)
R xx,WTR=DISCONNECT,END
```

5. Stop the CTRACE writer:

```
TRACE CT,OFF,COMP=SYSTCPIP,SUB=(tcpipprocname)
TRACE CT,WTRSTOP=CTWTR1,FLUSH
```

17.3.3 Multiple trace (CTRACE and packet) step

Following are the steps for a CTRACE and packet trace:

1. Start the traces:

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpipprocname)
R xx,WTR=CTWTR1,OPTIONS=(XXXX,XXXX),END
TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpipprocname)
R xx,WTR=CTWTR1,END
```

Note: You may use multiple writers (i.e. one for SYSTCPIP and one for SYSTCPDA).

2. Verify that the trace started successfully:

```
D TRACE,COMP=SYSTCPDA,SUB=(tcpipprocname)
D TRACE,COMP=SYSTCPIP,SUB=(tcpiprocname)
```

3. This step starts the packet trace processing in TCPIP. Without this step packets cannot be captured by the CTRACE component.

```
V TCPIP,tcpipprocname,PKT,ON,IP=xx.xx.xx.xx
```

4. Recreate the failure.
5. Stop the traces and writers.

17.3.4 Stopping the packet trace

1. To stop the packet trace issue the following command:

```
V TCPIP,tcpipprocname,PKT,OFF
```

2. Disconnect the external writer:

```
TRACE CT,ON,COMP=SYSTCPDA,SUB=(tcpipprocname)
       R xx,WTR=DISCONNECT,END
```

3. Stop the external writer:

```
TRACE CT,OFF,COMP=SYSTCPDA,SUB=(tcpipprocname)
```

4. Stop CTRACE comp(systcpip) and disconnect writer:

```
TRACE CT,ON,COMP=SYSTCPIP,SUB=(tcpipprocname)
R xx,WTR=DISCONNECT,END
```

5. Stop CTRACE writer:

```
TRACE CT,OFF,COMP=SYSTCPIP,SUB=(tcpipprocname)
TRACE CT,WTRSTOP=CTWTR1,FLUSH
```

Figure 17-1 on page 160 shows some sample TCP/IP CTRACE output.

```
SYSNAME   MNEMONIC  ENTRY ID    TIME STAMP     DESCRIPTION
-------   --------  --------  ---------------  ------------
SY1       ENGINE    50010002  12:42:20.182756  STREAMOP Request
 HASID..001C        PASID...001C        SASID..001C       MODID..EZBSSTO
 TCB....009E29A8    REG14...89BD7392    USER...TCPIP33    DUCB...0000401   CID..00000000
  ADDR...00000000  0A052058  LEN....00000004  Stream Descriptor
    +0000  00000000
  ADDR...00000000  09B33700  LEN....00000020  Execution Block
    +0000  00000001  00000001  00000000  00000000  00000000  0000000  00000000  00000000
  ADDR...00000000  09B33720  LEN....00000010  Operation Descriptor
    +0000  00000000  00000000  00000000  00000000
  ADDR...00000000  09B33EC0  LEN....00000080  Operation Block
    +0000  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEF  FEFEFEFE  FEFEFEFE
    +0020  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEF  FEFEFEFE  FEFEFEFE
    +0040  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEF  FEFEFEFE  FEFEFEFE
    +0060  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEFE  FEFEFEF  FEFEFEFE  FEFEFEFE
SY1       ENGINE    50010101  12:42:20.182967  Enable Stream Head Access
HASID..001C        PASID...001C        SASID..001C       MODID..EZBSSAC
TCB....009E29A8    REG14...89BD55C0    USER...TCPIP33    DUCB...0000401   CID..00000000
  ADDR...00000000  09E08098  LEN....00000188  Stream Access Control Block
    +0000  E2D2E2C3  00000000  00000000  00000000  E2D2C1E3  0000000  00000000  00000000
    +0020  00000000  00000000  00000000  00030100  E2E3D9C5  C1D4C1C  E2A39985  81944040
    +0040  E2D2C1E3  00000000  00000000  00000000  00000000  0000000  00000000  00030200
    +0060  E2E3D9C5  C1D4C1C3  E2A39985  81944040  E2D2C1E3  0000000  00000000  00000000
    +0080  00000000  00000000  00000000  00030300  E2E3D9C5  C1D4C1C  E2A39985  81944040
    +00A0  E2D2C1E3  00000000  00000000  00000000  00000000  0000000  00000000  00030400
    +00C0  E2E3D9C5  C1D4C1C3  E2A39985  81944040  E2D2C1E3  0000000  00000000  00000000
    +00E0  00000000  00000000  00000000  00030500  E2E3D9C5  C1D4C1C  E2A39985  81944040
    +0100  00000005  E2E3C1D9  E340C9D7  40D5C1D4  40C4D9C9  E5C5D94  40404040  40404040
    +0120  09E08098  00000000  00000000  00000000  00000000  0000000  00000000  00000000
    +0140  00000000  00000000  00000000  00000000  00000000  0000000  00000000  00000000
    +0160  00000000  00000000  00000000  00000000  00000000  0000000  00000000  00000000
    +0180  00000000  00000000
```

Figure 17-1 TCP/IP CTRACE example

18

CICS Transaction Gateway on z/OS

This chapter describes the gateway daemon and CICS Transaction Gateway on z/OS. The following topics are discussed:

- The gateway daemon
- CTG tracing
- CICS Transaction Gateway application trace
- Gateway daemon trace
- JNI tracing
- EXCI trace

18.1 Gateway daemon

The Gateway daemon listens on protocols defined in CTG.INI for gateway requests from remote Java client applications. It issues these requests to the Client daemon on distributed platforms, and directly to CICS over the EXCI on z/OS. The Gateway daemon runs the protocol listener threads, the worker threads, and the connection manager threads. It uses the GATEWAY section of CTG.INI (and on z/OS the ctgenvvar script) for its configuration.

The client daemon process, cclclnt, exists only on distributed platforms. It manages network connections to CICS servers. It processes ECI, EPI, and ESI requests, sending and receiving the appropriate flows from the CICS server to satisfy the application requests. It uses the CLIENT section of CTG.INI for its configuration.

The Gateway classes are the interface for Java Client applications to connect to the Gateway daemon. The Gateway classes, which are supplied with the CICS Transaction Gateway, must be in the classpath for Java Client applications to run.

The Java Client application is a Java application, servlet, or applet that communicates with the Gateway classes.

18.1.1 The Gateway daemon components

The Gateway daemon consists of two parts:

- The first is the listeners component that accepts connections (TCP, HTTP) from the remote Java client application and Gateway classes. In the CICS TG file system this is file ctgserver.jar in the classes subdirectory and it is written in Java.
- The second component makes the connection to the CICS Server using the External CICS Interface (EXCI). This component is the libCTGJNI.so file in the bin subdirectory and is written in C.

The two components communicate using the Java Native Interface (JNI), which allows Java code to interact with components written in C.

The Gateway daemon listener accepts the connections from the Gateway classes and passes them through to the JNI component of the Gateway daemon, which makes EXCI connections into the CICS server.

18.2 CTG trace file allocation

You control the size of trace files by specifying the **-tfilesize=<size>** option in conjunction with **-tfile=<filename>** on the **ctgstart** command. The <size> parameter is a value in Kbytes that should be greater than 4 and indicates the maximum size of the trace file. This will create a wrap-around file that will never exceed the maximum file size specified. A review of the timestamps is required to check for the newest and oldest entries in the trace. The top of the trace file will not necessarily be the start, or oldest entries.

You can also limit the size of the hex dumps and can obtain certain sections of the hex dump. Specify **–truncationsize=<size>** where size is the maximum size of HEX dump you want, measured in bytes (*not* Kbytes as for tfilesize). Use **–dumpoffset** to start the HEX dumps at a certain offset into the byte array. For example:

```
ctgstart –x –tfile=myctg.trc –tfilesize=100 –truncationsize=100 –dumpoffset=15
```

This will create a file called myctg.trc that contains a wrap-around gateway trace with a maximum file size of 100 Kbytes. Every HEX dump will be displayed starting at the 15th byte and will show only the first 100 bytes of the dump.

When using the tfilesize limit, it is crucial to have the timestamp entries or the file will be impossible to interpret. However, if you have not set the tfilesize limit then you may wish to trace without timestamps. This can be achieved by passing a **–notime** parameter to ctgstart:

```
ctgstart -x -tfile=gway.trc -notime
```

This is not something we would recommend. The timestamp is crucial when attempting to match client and server events. Not having timestamps makes that almost impossible.

18.3 CICS Transaction Gateway application trace

Application trace will typically be used where problems are arising with some component of the Java client application or its interaction with the gateway daemon. It can be used to determine the requests and data areas the client application is creating. This is useful to analyze client application and CICS server interaction. The timestamps are useful because they can be used to measure accurately the overall time required to interact with CICS. This helps to isolate the CICS interaction component of any performance measurements.

Performance problems can be investigated from here first to check that there is no problem with the sending of data, for example, incorrect COMMAREA sizes.

The first way to enable Application trace is to add additional code to a Java client application to interact with the Tracing API. This API is provided through the T class, part of the Gateway classes component of the CICS TG.

The T class allows the application itself to activate and deactivate trace at any time and to control the levels of trace.

Java application programmers can control the tracing from their application by importing the com.ibm.ctg.client.T class and then make calls to the static methods on the T class. For example:

```
import com.ibm.ctg.client.T;
T.setDebugOn(true);
```

This will activate full tracing. Alternatively, you can enable Application tracing by using Java system directives. Using system directives is a quick and easy way of activating application trace. For example, it can be used to quickly see additional information about the attempted interactions with the Gateway daemon.

The disadvantage is that you do not have as much control of what and when to trace as you have by using the tracing API (T class) in an application. For example, in a WebSphere Application Server environment, you must specify the -D option on the server's Java Virtual Machine (JVM™). However, this JVM may be shared by multiple threads of your application and perhaps by additional applications. If the system property has been activated, then all of the threads and applications will be affected and will all write trace output at the same time. If this is a problem then consider using the API control to only activate trace for the required parts of an application server environment.

The system directives and their equivalent API calls are as follows:

```
gateway.T = on/off => setDebugOn()
gateway.T.stack = on/off => setStackOn()
gateway.T.trace = on/off => setTraceOn()
gateway.T.timing = on/off => setTimingOn()
gateway.T.fullDataDump = on/off => setfullDataDump()
gateway.T.setTruncationSize = integer => setTruncationSize()
gateway.T.setDumpOffset = integer => setDumpOffset()
gateway.T.setTFile = String => setTFile(true,String)
gateway.T.setJNITFile = String => setJNITFile(true,String)
```

18.4 Gateway daemon trace

The Gateway daemon trace is useful for identifying exactly what requests have reached the gateway. The trace outputs all of the request parameters and can be used to ensure that the client applications are successfully passing their requests to the Gateway daemon with the correct parameters.

The gateway daemon trace can be started automatically at CICS start using the appropriate ctgstart option or dynamically via TCPAdmin.

TCPAdmin is a new protocol handler that is similar in configuration and operation to the standard TCP protocol handler. It is also configured through the CTG.INI configuration file and uses the same TCP protocol as its basis. There is a GUI front end that can be used to dynamically start tracing as well as changing the CTG.INI file, as follows:

- The **ctgstart –stack** option will turn on exception stack tracing only, and most Java exceptions are traced including expected exceptions during CTG normal operation. No other tracing is performed.
- **ctgstart –trace** enables standard tracing. The trace includes the first 80 bytes of the COMMAREA by default.
- **ctgstart –x** enables full debug tracing that includes everything traced by the -trace option with additional internal information. This includes the entire COMMAREA by default. This can produces large amounts of data and the HFS trace file size must be considered as well as the significant negative impact on performance.

Specifying the ctgstart trace options causes the gateway daemon to make calls to the T class at startup, in the same way that a Java client application does to control application tracing.

18.5 JNI tracing

JNI trace is useful in determining what caused an ECI_ERR_XXX condition. The JNI trace captures the interaction of the Gateway daemon and the EXCI (External CICS Interface) to enable CTG and host CICS communication/interaction problem diagnosis. The JNI trace can also be used to assist with RACF® authentication problems where the return from RACF is shown in the trace, and this can be used to determine reasons for security failures. The JNI trace will also be helpful in identifying system-related configuration problems on CTG for z/OS. It is less suited to application-related errors, where Application trace or Gateway daemon trace can more effectively provide detailed information about each request.

JNI tracing is enabled for the lifetime of the Gateway daemon by specifying a Java directive of **gateway.T.setJNITFile=<filename>**. This is done by using the **–j** option on the **ctgstart** command to pass a parameter to the JVM. Here is an example:

```
ctgstart –j-Dgateway.T.setJNITFile=jni.trc
```

You can also start the JNI trace from within the Java client application's JVM. This means that you can switch on JNI tracing from within your Java client application by using the T class static method, setJNITFile(boolean, String). You can also pass a Java directive to the Java client application's JVM to switch on JNI tracing at startup, without writing any application code, for example:

```
java -Dgateway.T.setJNITFile=jni.trc com.my.Application
```

Finally, if you are using a gateway daemon (remote mode) then you can use the TCPAdmin function to allow dynamic activation and deactivation of JNI trace.

JNI logs all error interactions with the EXCI automatically and these are written to:

```
$HOME/ibm/ctg/ctgjnilog.<process id>
```

$HOME is an environment variable, typically /u/<ctguser>, and <ctguser> is the userid that the Gateway daemon runs under. If $HOME does not exist then the user's current directory is used. <process id> is the process ID of the Gateway daemon. You can issue the following command under USS to determine the location of the files:

```
ps -ef | grep JGate
```

The log may already have sufficient data to assist with problem determination, thus avoiding the need to use JNI trace.

18.6 EXCI trace

The EXCI trace shows activity on the interface between CTG and CICS Server and is stored in the Gateway region address space, or GTF.

EXCI produces SYSMDUMPs for some error conditions and SVC dumps for more serious conditions. These dumps contain all the external CICS interface control blocks as well as trace entries. You can use the z/OS Interactive Problem Control System (IPCS) to format these dumps. You can, of course, request a DUMP to be captured manually.

To use GTF for EXCI tracing, GTF user tracing must be active. GTF must be started in the z/OS image where the CTG address space you are going to trace resides, and you must specify GTF=ON in the DFHXCOPT options table. If you use GTF trace for both the CICS server region and the EXCI region, the trace entries are interleaved, which can help you with problem determination in the CICS-EXCI environment. The external CICS interface does not support any form of auxiliary trace.

18.6.1 Enable a GTF trace

To enable GTF tracing in the DFHXCOPT module you must set the following:

```
DFHXCO TYPE={CSECT|DSECT}
[,CICSSVC={0|number}]
[,CONFDATA={SHOWHIDETC}]
[,DURETRY={30|number-of-seconds}]
[,GTF={OFF|ON}]
[,MSGCASE={MIXED|UPPER}]
[,SURROGCHK={YESNO}]
[,TIMEOUT={0|number}]
[,TRACE={OFF|1|2}]
[,TRACESZE={16|number-of-kilobytes}]
[,TRAP={OFF|ON}]
END DFHXCOPT
```

ECXI options

Specify the EXCI options table in ctgenvvar, as follows:

```
EXCI_OPTIONS='YOUR.EXCI.LOADLIB'
```

In this example, 'YOUR.EXCI.LOADLIB' contains the assembled DFHXCOPT.

Specify the trace level you require in DFHXCOPT and then assemble the table into a data set such as YOUR.EXCI.LOADLIB. ctgenvvar builds the STEPLIB environment variable automatically from any components that must be on the STEPLIB.

By specifying EXCI_OPTIONS='YOUR.EXCI.LOADLIB' you are ensuring that your assembled DFHXCOPT table will be added to the CICS TG's STEPLIB value because ctgenvvar contains the following line:

```
export STEPLIB=${STEPLIB}:${EXCI_OPTIONS}:${EXCI_LOADLIB}
```

If you decide to specify environment variables in an alternative way then make sure that STEPLIB contains your EXCI options data set.

Also ensure that the CICS job has your assembled DFHXCOPT table in the STEPLIB. If you do not do this then you may find that the CICS Transaction Gateway trace points (numbered 800x) will not be seen in the trace.

19

WebSphere MQSeries z/OS diagnostic procedures

This chapter describes WebSphere MQSeries for z/OS diagnostic procedures and provides unique messages that, together with the output of dumps, are aimed at providing sufficient data to allow diagnosis of the problem without having to try to reproduce it.

19.1 WebSphere MQSeries for z/OS

WebSphere MQSeries for z/OS tries to produce an error message when a problem is detected, and all diagnostic messages all begin with the prefix CSQ. Each error message generated by MQSeries is unique; it is generated for one and only one error. Information about the error can be found in WebSphere *MQSeries for z/OS Messages and Codes*, GC33-0819. The first three characters of the names of WebSphere MQSeries for z/OS modules are also CSQ. The fourth character uniquely identifies the component. Characters five through eight are unique within the group identified by the first four characters. There may be some instances when no message is produced, or, if one is produced, it cannot be communicated. In these circumstances, you might have to analyze a dump to isolate the error to a particular module.

19.2 Dumping MQ MSTR, MQ CHIN and CHIN data space

When capturing diagnostic data to analyze MQ problems ensure that you dump both the MQ main (MSTR), the channel initiator (CHIN) address spaces as well as the CHIN data space, CSQXTRDS. Figure 19-1 shows the procedure to dump the MQ address spaces.

```
DUMP COMM=(MQSERIES MAIN DUMP)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=(CSQ1MSTR,CSQ1CHIN),CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
IEE600I REPLY TO 01 IS;JOBNAME=(CSQ1MSTR,CSQ1CHIN),CONT
R 02,DSPNAME=('ssidCHIN'.CSQXTRDS),CONT
IEE600I REPLY TO 02 IS;DSPNAME=(‘ssidCHIN,CSQXTRDS),CONT
R 03,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEE600I REPLY TO 03 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
```

Figure 19-1 Dumping the WebSphere MQ MSTR and CHIN address spaces

19.3 MQ tracing using GTF

You can obtain information about API calls and user parameters passed by some MQSeries calls on entry to, and exit from, MQSeries. To do this, you should use the global trace in conjunction with the MVS generalized trace facility (GTF).

You must start GTF if you want to use it for problem determination. Figure 19-2 provides and example of the GTF start procedure.

```
START GTF.procname,,,(MODE=INT)
HASP100 GTF.procname ON STCINRDR
HASP373 GTF.procname STARTED
*01 AHL100A SPECIFY TRACE OPTIONS
R 01,TRACE=JOBNAMEP,USRP
TRACE=JOBNAMEP,USRP
IEE600I REPLY TO 12 IS;TRACE=JOBNAMEP,USRP
*02 ALH101A SPECIFY TRACE EVENT KEYWORDS - JOBNAME=,USR=
R 02,JOBNAME=(ssidMSTR,ssidCHIN,jobname),USR=(5E9,5EA,5EB,5EE,F6C)
JOBNAME=(ssidMSTR,ssidCHIN,jobname),USR=(5E9,5EA,5EB,5EE,F6C)
IEE600I REPLY TO 13 IS;JOBNAME=(ssidMSTR,ssidCHIN,jobname),USR=(5E9,5EA,5EB,5EE,F6C)
*03 ALH102A CONTINUE TRACE DEFINITION OR REPLY END
R 03,END
END
IEE600I REPLY TO 14 IS;END
AHL103I TRACE OPTIONS SELECTED-USR=(5EA,5E9,5EB,5EE,F6C)
AHL103I JOBNAME=(ssidMSTR,ssidCHIN,jobname)
*04 AHL125A RESPECIFY TRACE OPTIONS OR REPLY U
R 04,U
```

Figure 19-2 GTF procedure for WebSphere MQ

The *jobname* included on the GTF JOBNAME parameter can be used for batch jobs, or other subsystems, for example, CICS that may be using MQ services for which MQ trace data might also be required.

If the problem that is being traced is batch related and not using MQ remote queues that require MQ CHIN address space, you can remove the MQ CHIN address space from the GTF JOBNAME parameter. There is no point collecting trace data that is not relevant to the problem being reviewed.

Note: If in doubt about what MQ address spaces to trace, always include both the MSTR and CHIN address spaces on the JOBNAME parameter and also include the CHIN data space on the DSPNAME parameter. The MQSeries trace is for each queue manager subsystem for which you want to collect data.

19.3.1 Starting GTF

When you start the GTF, you should specify the USRP option. You will be prompted to enter a list of event identifiers (EIDs). The EIDs used by MQSeries are:

- 5E9 - To collect information about control blocks on entry to MQSeries
- 5EA - To collect information about control blocks on exit from MQSeries
- 5EB - To collect information on entry to internal MQ functions
- 5EE - To collect information on exit from internal MQ functions
- F6C - To collect information about CICS/MQ functions

You can also use the JOBNAMEP option, specifying the batch, CICS, IMS, or TSO job name, to limit the trace output to certain jobs.

Once started, you can display information about, alter the properties of, and stop the trace with the **`DISPLAY TRACE`**, **`ALTER TRACE`**, and **`STOP TRACE`** commands.

To use any of the trace commands, you must have one of the following:

- Authority to issue start/stop trace commands (trace authority)
- Authority to issue the display trace command (display authority)

To format the user parameter data collected by the global trace you can use the IPCS **GTFTRACE USR(xxx)** command, where **xxx** is:

5E9	To format information about control blocks on entry to MQSeries MQI calls
5EA	To format information about control blocks on exit from MQSeries MQI calls
5E9,5EA	To format information about control blocks on entry to and exit from MQSeries MQI calls
5EB,5EE	To format information about control blocks on entry to and exit from MQSeries Internal functions

The key pieces of diagnostic data required for diagnosing WebSphere MQ on z/OS problems are the MQ MSTR and CHIN (Channel Initiator) joblogs, the CHIN trace for remote messaging activity and the MSTR trace which is processed via GTF (Generalized Trace Facility). The CHIN trace data is always written to the CHIN data space and must be captured during the DUMP process.

Key parameters

Some key parameters must be set up to ensure sufficient space is allocated within the WebSphere MQ address spaces to hold the required data. These can be set up as follows:

- Set the CHIN trace table size to 5Mb via **CSQ6CHIP**.
- Set the trace table size to 250 via **CSQ6SYSP**.

MQ needs to be recycled to pick up these changes.

Starting GTF trace

Start GTF Trace with the following parameters:

- `PARM=MODE(INT)`
- `REGION=2880K`
- The GTF parameters SADMP, SDUMP, ABDUMP, and BLOK should all be set to at least 10Mb.
- `TRACE=JOBNAMEP,USRP`
- `JOBNAME=(ssidMSTR,jobname),USR=(5E9,5EA,5EB,5EE,F6C)`

Now that the GTF trace is running, start the internal MQ trace to output the MQ trace data to the GTF. Start MQ tracing as follows:

```
+cpfSTART TRACE(GLOBAL) DEST(GTF) CLASS(*) RMID(*)
```

For problems that are perceived to be MQ code defect issues, CLASS(*) is required because this will contain all internal code tracing as well as API tracing. If you want to trace application-related problems, CLASS(02:03) should be sufficient.

The CPU overhead when running the GLOBAL trace can be high, but as we generally only require trace data for a very short period of time, for example, 20 seconds, the cost is small if the captured data will assist with resolving the problem.

Although you have specified DEST(GTF), CHIN trace data will still be routed to DEST(RES). Everything else will go to GTF, which will be dumped when the MSTR, CHIN and CHIN data space are dumped, due to the MODE(INT) GTF parm.

It might be a good idea to SLIP on the CSQXxxxE message if this is the common point of failure. Figure 19-3 shows how this can be done.

```
SLIP SET,IF,LPAMOD=(IGC0003E,0),
DATA=(1R?+4,EQ,C3E2D8E7,1R?+8,EQ,FxFxFxC5),
JOBNAME=ssidCHIN,
JOBLIST=(ssidMSTR,ssidCHIN),
DSPNAME=('ssidCHIN'.CSQXTRDS),
SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),
MATCHLIM=1,END
```

Figure 19-3 SLIP on a WebSphere MQ message

Figure 19-4 shows the more streamlined MSG SLIP approach.

```
SLIP SET,MSGID=CSQXxxxE,
JOBNAME=ssidCHIN,
JOBLIST=(ssidMSTR,ssidCHIN),
DSPNAME=('ssidCHIN'.CSQXTRDS),
SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),
MATCHLIM=1,END
```

Figure 19-4 SLIP for WebSphere MQ using the MSGID parameter

This will dump the QMGR and CHIN address spaces and the associated data spaces when msgid CSQXxxxE is generated by the MQ CHIN address space. Naturally, you have to substitute a valid error message.

- To stop MQ Internal trace:

```
+cpfSTOP TRACE(*)   **** +cpf = ssid prefix ****
```

- Stop GTF.

Figure 19-5 shows a sample MQ 5EA trace entry.

```
USRD9 5EA ASCB 00F3120O            JOBN MQ69CHIN
CSQW073I EXIT:  MQSeries user parameter trace
OPEN
Thread... 008D7838  Userid... COAKLEY   pObjDesc. 0EBB2F38
RSV1..... 00000000  RSV2..... 00000000  RSV3..... 00000000
CompCode. 00000000  Reason... 00000000
D4D8F6F9  C3C8C9D5  00000000  00000000  | MQ69CHIN........ |
E3F0F0F8  C4F7F8F3  F8404040  40404040  | T008D7838       |
40404040  40404040  40404040  40404040  |                 |
40404040  40404040  40404040  40404040  |                 |
USRD9 5EA ASCB 00F3120O            JOBN MQ69CHIN
CSQW073I EXIT:  MQSeries user parameter trace
+0000  D6C44040  00000001  00000005  00000000  | OD  ............ |
+0010  00000000  00000000  00000000  00000000  | ................ |
+0020  00000000  00000000  00000000  00000000  | ................ |
+0030  00000000  00000000  00000000  D4D8F6F9  | ............MQ69 |
+0040  40404040  40404040  40404040  40404040  |                  |
+0050  40404040  40404040  40404040  40404040  |                  |
+0060  40404040  40404040  40404040  C3E2D84B  |             CSQ. |
+0070  5C000000  00000000  00000000  00000000  | *............... |
+0080  00000000  00000000  00000000  00000000  | ................ |
```

Figure 19-5 WebSphere MQ GTF trace example

A good search string when reviewing MQ trace data is MQRC. This will locate the occurrence of the MQ Return Codes in the trace. MQ functions will also be shown, for example, OPEN, CLOSE, GET, and PUT, and associated MQ modules.

19.4 WebSphere MQSeries z/OS channel trace

It is worth noting that a wrap-around line trace is always active for each channel and the trace data is stored in a 4K buffer (one for each channel) in the CHIN address space.

This is a good source of data for reviewing problems with LU6.2 and TCP/IP channels.

To enable the trace data to be reviewed, you must have a CSQSNAP DD statement in your CHIN JCL. To display the trace data (to write it to the data set specified in the CSQSNAP DD, for example SYSOUT=*), you must start a CHIN trace as follows:

```
START TRACE(GLOBAL) RMID(231) CLASS(4) IFCID(202)
DISPLAY CHSTATUS(channel) CURRENT
```

In this example, `(channel)` is the channel that you need to trace.

19.5 IPCS and WebSphere MQSeries z/OS

WebSphere MQSeries for z/OS provides a set of panels to help you process dumps using IPCS.

1. From the IPCS Primary Option menu select ANALYSIS (Analyze dump contents). The IPCS MVS Analysis Of Dump Contents panel appears.

2. Select COMPONENT → MVS component data (Option **6**). The IPCS MVS Dump Component Data Analysis panel appears. The appearance depends on the products installed at your installation, but will be similar to that shown in Figure 19-6.

```
------- IPCS MVS DUMP COMPONENT DATA ANALYSIS -------------------
OPTION ===> SCROLL ===> CSR
To display information, specify  S option name   or enter S to the
left of the Option desired. Enter ? to the left of an Option to
display help regarding the component support.
Name Abstract
ALCWAIT Allocation wait summary
AOMDATA AOM analysis
ASMCHECK Auxiliary storage paging activity
ASMDATA ASM control block analysis
AVMDATA AVM control block analysis
COMCHECK Operator communications data
CSQMAIN MQSeries dump formatter panel interface
CSQWDMP MQSeries dump formatter
CTRACE Component trace summary
DAEDATA DAE header data
DIVDATA Data-in-virtual storage
```

Figure 19-6 IPCS dump component data analysis panel

3. Select **CSQMAIN MQSeries dump formatter panel interface**.
4. Figure 19-7 on page 173 shows the IBM MQSeries for z/OS - Dump Analysis menu.

```
-------IBM MQSeries for MVS/ESA - DUMP ANALYSIS----------------
COMMAND ===>
1 Display all dump titles 00 through 99
2 Manage the dump inventory
3 Select a dump
4 Display address spaces active at time of dump
5 Display the symptom string
6 Display the symptom string and other related data
7 Display LOGREC data from the buffer in the dump
8 Format and display the dump
9 Issue IPCS command or CLIST
```

Figure 19-7 WebSphere MQ dump analysis panel

19.5.1 Using IPCS for WebSphere MQSeries

You can also use IPCS to interrogate the dump using the WebSphere MQSeries **VERBX CSQWDMP** command or the channel initiator **VERBX CSQXDPRD**. For example:

- For default formatting of all address spaces, using information from the summary portion of the dump use:

  ```
  VERBX CSQWDMP
  ```

- To display the trace table from a dump of subsystem named MQMT, which was initiated by an operator (and so does not have a summary portion) use:

  ```
  VERBX CSQWDMP 'TT,SUBSYS=MQMT'
  ```

- To display all the control blocks and the trace table from a dump produced by a subsystem abend, for an address space with ASID (address space identifier) 1F, use:

```
VERBX CSQWDMP 'TT,LG,SA=1F'
```

- To display the portion of the trace table from a dump associated with a particular EB thread, use:

```
VERBX CSQWDMP 'TT,EB=nnnnnnnn'
```

- To display message manager 1 report for local non-shared queue objects whose name begins with 'ABC' use:

```
VERBX CSQWDMP 'MMC=1,ONAM=ABC,Obj=MQLO'
```

- The IPCS VERBEXIT CSQXDPRD enables you to format a channel initiator dump. You can select the data that is formatted by specifying keywords. For example:

```
VERBX CSQXDPRD ‘SUBSYS=MQMT,CHST=3’
```

  This displays all channel information, a program trace, line trace, and formatted semaphore table print of all channels in the dump.

19.6 WebSphere MQ JAVA tracing

WebSphere MQ JMS applications normally invoke trace by using command line arguments to the **JAVA** command. For example:

```
java -DMQJMS_TRACE_LEVEL=base   myJMSApplication
```

Sometimes this is not possible, for example when the JMS application is started as a servlet within an application server. In this case it may be appropriate to invoke trace from within the source code.

This can be done as follows:

- `com.ibm.mq.jms.services.ConfigEnvironment.start();`

  This uses MQJMS TRACE_LEVEL and _DIR from the system properties.

- `com.ibm.mq.jms.services.ConfigEnvironment.start(String level);`

  This uses MQJMS_TRACE_DIR; the level parameter should “on,” or “base.”

- `com.ibm.mq.jms.services.ConfigEnvironment.stop();`

The value of MQJMS_TRACE_DIR can be set programmatically with something like the following:

```
java.util.Properties.props = System.getProperties();
props.put("MQJMS_TRACE_DIR", "my/directory/name";
System.setProperties(props);
```

The filename is fixed to mqjms.trc: only the directory can be modified in this manner.

19.7 Taking JMS traces within WebSphere

From the WebSphere Administrator's Console select the following:

Default server → Service Tab → Trace services

Then set the following:

```
com.ibm.ejs.jms.*=all=enabled
```

Note: This provides Websphere JMS Wrapper trace only. If you need a full JMS trace, use the programmatic method shown previously.

20

WebSphere Business Integration Message Broker on z/OS

In this chapter we discuss the components that make up the WebSphere Business Integration (WBI) Message Broker environment, then we look at the sources of diagnostic data.

20.1 Components of WBI message broker on z/OS

The WBI message broker architecture manages the modelling of the application connectivity and integration operations using message flows and nodes, and the modelling of message structure to enable processing within these nodes.

The *Broker* is the run-time environment that assigns units of execution (threads) to message flows to process messages. It takes the models and realizes them as concrete implementations. The primary objective of the broker is to achieve consistency at the application layer. This means that a message flow or set modelled in the Control Center and deployed via the Configuration Manager must operate without reference to the platform of execution. So, for example, we don't want to have ESQL that has distinct processing for z/OS.

The modelling of the message flows and messages is performed using a graphical user interface called the *Control Center*. It also allows the modelling of broker topologies and assignment of resources (message flows and sets) to these topologies. There is support for management of Publish Subscribe topics and their Access Control Lists (ACLs) governing their security.

Resources created and manipulated by the Control Center are stored in databases controlled by the *Configuration Manager* (DB2 on the z/OS platform). This is the master repository for WBIMB resources. An authorized Control Center operator can deploy resources to the broker for run time use.

Finally, the *User Name Server* is an optional component used to support Publish Subscribe ACL processing. It needs to operate on any platform where it can gather relevant principals (users and groups) and report them to the Broker and Configuration Manager.

The broker run-time environment is a collection of address spaces (ASIDs), which allows natural isolation, recovery, and scalability. Each address space contains at least two Language Environment (LE) processes. The first, or *infrastructure* process is started as an authorized process so that it can create z/OS components (for example, Program Calls (PCs) for SVC dumps, and so forth), and then returns to problem state. This process only exists on z/OS.

After initialization, it creates and monitors a second process, which performs the main brokering function. The main process in each ASID runs platform-independent code using C++ and Java (publish/subscribe) to implement brokering function.

Control process

This is the broker started task address space. It is small, being a monitor for failures of the Administration Agent (AA) address spaces. On z/OS, a console listener thread enables z/OS console interactions with users through the MODIFY interface.

Administration agent

This serves as the administration to the configuration manager and, by extension, command center. It manages the deployment of message flows and message sets, and manages the life cycle and command reporting of execution groups (EG).

Execution group

This is where the message flows deployed from the Configuration Manager execute. The DataFlowEngine process itself contains a number of threads and predefined flows

(Configuration, PubSub Control) to support the various brokering functions. Multiple EG address spaces remove any concern about Virtual Storage Constraint Relief (VSCR).

User name server

This address space retrieves all valid principals (users, groups) from z/OS and reports them to the Configuration Manager (CM) and broker to support pub/sub topic ACL processing. It has its own control process.

User process

As well as from the console, user commands can be issued from JCL and directly through a UNIX shell.

20.2 Address spaces that interact with the broker

There are a large number of associated address spaces with which the broker interacts.

OMVS

This address space provides several industry standard interfaces (XPG4) that allow the WMQI processing model and code to be largely platform-independent.

WebSphere MQ

This is one of the primary transports for data flows, and the WBI Message Broker uses it for inter-process and inter-platform communication. For example, the AA communicates with the EGs and CM using XML messages flowed over WMQ.

DB2

Again, this is heavily used by data flows for data warehousing and augmentation, but it is also used to store the deployed data flows and message dictionaries. The broker also keeps some of its internal state in DB2.

RRS

As the broker runs within regular z/OS address spaces, Resource Recovery Services (RRS) is the transaction manager that enables the coordination of resources (message queues, database tables) accessed by a dataflow.

20.3 Dumps captured by WBI message broker

A WebSphere MQ Integrator for z/OS broker or User Name Server produces different types of dumps depending on where the original error occurs:

SVC dumps These are produced for errors in the WebSphere MQ Integrator for z/OS Infrastructure main program (bipimain).

Core dumps These are produced for errors in the broker or User Name Server executables (bipservice, bipbroker, DataFlowEngine, and bipuns).

Core dumps are SYSMDUMP dumps and are written to the started task's user directory. The name is coredump.pid, where pid is the hexadecimal value of the process ID of the process that encountered the error.

The maximum size of a core dump is defined through MAXCORESIZE in the BPXPRMxx parmlib member. The IBM-supplied default is 4MB. To ensure completeness of a core dump

of any WebSphere MQ Integrator for z/OS address space, this should be changed to 2GB. The started task user's directory must then have at least this size.

When the error recovery routines of WebSphere MQ Integrator for z/OS provide an SVC dump, a core dump is also written to your started task user ID directory. In that case the SVC dump and the entry on the z/OS syslog are of interest for IBM to resolve the problem.

20.4 Reviewing a WBI message broker dump

You can use the **mqsireaddump** command utility to format z/OS dump data before sending it to the IBM Support Center for analysis. The output of the format operation is an XML file.

The **mqsireaddump** command utility for z/OS is a set of REXX procedures, and is executed in batch mode via JCL. The output is written to a separate file in the /component_HFS/output/DumpFormatter directory.

In order to use **mqsireaddump**, you need to customize the JCL named BIPJRDMP as follows:

1. Change the name of the sequential dump data set (the input), for example:

   ```
   //WMQIDUMP DD DISP=SHR,DSN=SYS3.DUMP.#MASTER#.T063524.S00001
   ```

2. Change the name of the XML file in an HFS directory where you want the output written, for example:

   ```
   %RUN COMPONENTHFS/output/DumpFormatter/MyDump.xml
   ```

3. Change the user ID's VOLSER for the IPCS Dump Directory, for example:

   ```
   %BLSCDDIR DSNAME(WMQI210.DUMP.DIRECT) VOLUME(*)
   ```

4. Submit the job.

To format core dumps using the **mqsireaddump** command utility, you must copy them to a partitioned data set using the TSO/E **OGET** command. The data set must be allocated with LRECL(4160) and transferred as binary. For example:

```
oget '/u/user_directory/coredump.pid' 'mvs_dataset_name.pid' bin
```

20.5 Dumping the WBI message broker address spaces

Due to the number of components and related products that make up the WBI Message Broker environment, collecting diagnostic data can be a complex process. In most cases you will be dumping multiple address spaces, which will probably include the control process address space, the administration agent, the execution groups, and possibly the UNIX Systems Services address space and even DB2 or RRS.

The key requirements are to dump the control process (usually having a name that includes the BRK suffix, for example, MQ01BRK), the administration agent (suffix = BRK1, for example, MQ01BRK1), and the failing execution group, which will have a suffix that starts with BRK2 for the first execution group address space and increments for each additional execution group (for example, MQ01BRK2, MQ01BRK3, MQ01BRK4, and so forth.)

It may also be necessary to include the WebSphere MQ Queue Manager address space in the dump.

Figure 20-1 shows an example of the possible WBIMB address space dump requirements.

```
DUMP COMM=(WMQI CP/Broker/Execution Dump)
*01 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
R 01,JOBNAME=(MQ01BRK,MQ01BRK1,MQ01BRK2,USSstc, MQ01MSTR),CONT
*02 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
IEE600I REPLY TO 01 IS;JOBNAME=(MQ01BRK,MQ01BRK1,MQ01BRK2,USSstc,MQ01MSTR),CONT
R 02,SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEE600I REPLY TO 02 IS;SDATA=(CSA,RGN,PSA,SQA,LSQA,TRT,SUM),END
IEA794I SVC DUMP HAS CAPTURED: 869
DUMPID=001 REQUESTED BY JOB (*MASTER*)
```

Figure 20-1 WebSphere Business Integration message broker address space dump

20.6 Displaying the status of a trace

To display the status of a trace, use the report trace command, **reporttrace (RT)**.

Names are case-sensitive and you should include the names in single quotes if they contain mixed case characters. For example, see the following command:

```
F MQP1BRK,RT T=YES,E='FONE2ONE ',F='MFFONE2ONE'
```

This command produces the following output:

```
+BIP8098I MQP1BRK Trace level:none,mode:safe,size:4096 KB.
+BIP8071I MQP1BRK Successful command completion.
```

20.7 WBI message broker user execution group trace

To collect a user execution group trace, use the collect trace command, **changetrace** (**CT** - change trace settings). If your flow name is in mixed case, enclose it in single quotes.

For example, to collect a trace, turn trace on using the command:

```
F broker,CT,u=YES,L=debug,E='default'
```

When you have captured your trace, turn trace off with:

```
F broker,CT,u=YES,L=none,E='default'
```

To format your trace, do the following:

1. Edit your component PDSE, and create a new member, for example SEGTRACE. Copy it into BIPJLOG.
2. Search for `-t -b agent` and change it to `-u -e default`, where default is the name of your execution group.
3. Search for format.log. This is where the output file is stored. Change the name if you want to have trace-specific names.

20.8 WBI message broker execution group trace

IBM might ask you to collect a trace of the internal processing of an execution group. To turn tracing on, use the collect trace command, **changetrace** (**ct** - Change trace settings). If your flow name is in mixed case, enclose it in single quotes.

For example to collect a trace, turn trace on using the command:

```
F broker,ct t=YES,L=debug,E='default'
```

When you have captured your trace, turn trace off with:

```
F broker,ct t=YES,L=none,E='default'
```

To format your trace, do the following:

1. Edit your component PDSE, and create a new member, for example SEGTRACE. Copy it into BIPJLOG.
2. Search for `-t -b agent` and change it to `-t -e default`, where default is the name of your execution group.
3. Search for format.log. This is where the output file is stored. Change the name if you want to have trace-specific names.

20.9 WBI message broker service trace

IBM might ask you to collect a trace of the internal processing, not related to an execution group. To turn trace on, use the **`changetrace`** command. If your flow name is in mixed case, enclose it in single quotes.

For example, to collect a trace, turn trace on using the command:

```
F broker,ct t=YES,L=debug,b=yes
```

When you have captured your trace, use the command:

```
F broker,ct t=YES,L=none,b=yes
```

To format your trace, do the following:

1. Edit your component PDSE, and create a new member, for example STRACE.Copy it into BIPJLOG.
2. Search for format.log. This is where the output file is stored. Change the name if you want to have trace-specific names.

20.10 WBI message broker useful output files

The joblog and SYSLOG contain the necessary messages. SYSLOG (SDSF) contains all BIP messages. In the event of a problem, it's also worth looking for messages from other subsystems that may be a factor in the WMQI problem, for example, DB2, WebSphere MQ or RRS.

Because of the way the WMQI administrative agent and execution group address spaces are started, their messages are not collected under their job logs. Review the SYSLOGD HFS file (if you are using this).

To overcome this difficulty, the SYSLOGD allows you to route joblog messages to a joblog file defined by rules in /etc/syslog.conf.

20.11 Useful HFS files

The following files in the HFS are useful for diagnostic analysis:

- .../output/stderr, .../output/stdout contain minimal output from WQMI, including JVM.
- .../output/cvpplog, .../output/cvpserr, .../output/cvpslog contain the output from the customization verification job.
- .../output/joberr, .../output/jobout contain output from customization jobs, BIP$DB01-05, BIP$MQ01, and BIP$UT01.
- .../output/traceodbc contains ODBC tracing information. Enable this through DSNAOINI.
- .../ENVFILE contains USS settings for broker or user name server.
- .../dsnaoini contains ODBC configuration information.
- .../mqsicompcif contains master source for WMQI configuration.

You should check for dumps and other diagnostic information in the following locations:

1. In the home directory of the started task user ID. An authorized user can issue the TSO command **`LU id OMVS`** to display this information.
2. In the component directory in the /output subdirectory.
   For example, for a broker called MQP1BRK check in the following file:
   /var/wmqi/MQP1BRK/output

20.12 WBI Message Broker for z/OS trace files

Examples of the file name format for various WMQI traces are as follows:

- Administration agent service trace:
  .../log/<broker name>.agent.trace.bin.0
- Administration agent user trace:
  .../log/<broker name>.agent.userTrace.bin.0
- WMQI command service trace:
  .../log/<broker name>.mqsistop.trace.bin.0
- WMQI command user trace:
  .../log/<broker name>.mqsistop.userTrace.bin.0
- Control process service trace:
  .../log/<broker name>.service.trace.bin.0
- Control process user trace:
  .../log/<broker name>.service.userTrace.bin.0
- Execution group/DataFlow service trace:
  .../log/<broker name>.02345678-1234-1234-1234-123456789003.trace.bin.0
- Execution group/DataFlow user trace:
  .../log/<broker name>.02345678-1234-1234-1234-123456789003.userTrace.bin.0

Trace files should be formatted at the customer site using BIPJLOG JCL in component PDS.

21

WebSphere Application Server for z/OS

This chapter describes some of the key areas of diagnostic data that can be used for problem determination in WebSphere Application Server on z/OS. As with many of the products that are built using numerous components and utilizing a broad range of system functions, the requirement to collect diagnostic data from all the related components in many problem scenarios is not mandatory. Many problems will require only specific diagnostic data to be collected for the individual component that is experiencing the problem, whereas the more complex problems may require a larger subset of data to be collected, or even diagnostic data collection for all related components.

WebSphere Application Server applications are deployed within a WebSphere Application Server generic server. One or more server instances must be defined within the WebSphere Application Server node and each server instance consists of a controller and one or more servants. The controllers are started by MVS as started tasks, and servants are started by WLM, as they are needed. WebSphere Application Server for z/OS servers are similar to IMS or CICS regions and may be self-contained or dependent on other servers.

This chapter describes:

- WebSphere on z/OS diagnostic data
- Dumping WebSphere Application Server CTRACE
- LDAP trace
- JVM debugging tools for z/OS

21.1 WebSphere on z/OS diagnostic data

The WebSphere for z/OS error log is a log stream data set managed by the z/OS System Logger. This log stream resides on either a staging data set on DASD or in the Coupling Facility.

The data stored in the WebSphere Application Server log stream must be formatted using the BBORBLOG Rexx Exec routine. By default, BBORBLOG formats the error records to fit a 3270 display.

To view the error log stream output you need to invoke the BBORBLOG browser via ISPF option **6** command line. For example:

```
ex 'BBO.SBBOEXEC(BBORBLOG)' 'WAS_LogStream_Name'
```

This creates a browse data set named userid.stream_name, which contains the contents of the log stream.

During WebSphere Application Server initialization, an attempt is made to connect to the appropriate logstream data set. If this connection is successful, you will see the following message, which indicates the name of the data set being used:

```
BBOOOO24I ERRORS WILL BE WRITTEN TO <logstream name> LOG STREAM FOR JOB <server
name>
```

If, however, the server cannot connect to the logstream, the message is instead written to CERR, which puts it in the SYSOUT of the job output. This would be indicated by the message:

```
BBOOOO24I ERRORS WILL BE WRITTEN TO CERR FOR JOB <server name>
```

The log stream records error information when WebSphere for z/OS detects an unexpected condition or failure within its own code. Use the error log stream in conjunction with other facilities available to capture error or status information, such as an activity log, trace data, system logrec, and job log.

Figure 21-1 shows some sample data from the WebSphere Application Server for z/OS log.

```
2002/09/05 20:26:10.233 01 SYSTEM=SC49 SERVER=NAMING01 JobName=BBONMS
ASID=0X0060 PID=0X01060042 TID=0X23AE32E0 0X000008 c=1.19
./bboi3pli.cpp+3712 ... BBOU0011W The function
IBOIM390PrivateLocalToServer_IMContainer_Impl::beforeMethodDispatch(::
ByteString*,CORBA::Object_LocalProxy_ptr,const
char*,CORBA::Long)+3712 raised CORBA system exception
CORBA::INV_OBJREF.  Error code is C9C21444
```

Figure 21-1 WebSphere Application Server log example

21.1.1 WebSphere Application Server joblog and syslog

Each MVS address space maintains a number of JES (Job Entry Subsystem) output files that contain information related to the state of the control and server regions.

The SYSLOG (System Log) stores messages that are written to the master console. Most serious messages related to WebSphere Application Server on z/OS will be written to both the syslog and also to the WebSphere Application Server job logs.

The output from both the joblog and SYSLOG can be reviewed using the Spool Display and Search Facility (SDSF). The key JES spool DD names associated with WAS address space problem diagnosis are:

- **JESMSGLG** - This section contains start-up messages, including a list of environment variable values and server settings, and the service level of WebSphere:

  ```
  BBOM0007I CURRENT CB SERVICE LEVEL IS build level o0511.05 release
  date 03/14/05 20:48:15.
  ```

  It also lists the Java service level when in a J2EE™ server region:

  ```
  BBOJ0011I JVM Build is J2RE 1.4.2 IBM z/OS Persistent Reusable VM
  build cm142sr1a-20050209 (JIT enabled: jitc).
  ```

- **JESYSMSG** - This section may list more messages, dump information and, again, a list of environment variables and server settings. Figure 21-2 shows a sample of the output written to the JESYSMSG log.

```
BBOO0237I WEBSPHERE FOR Z/OS DAEMON cl6481/nd6481/WS6481D IS STARTING.
BBOM0007I CURRENT CB SERVICE LEVEL IS build level o0511.05 release
date 03/14/05 20:48:15.
BBOM0001I adjunct_region_dynapplenv_jclparms: NOT SET.
BBOM0001I adjunct_region_dynapplenv_jclproc: NOT SET.
BBOM0001I adjunct_region_jvm_properties_file: NOT SET.
BBOM0001I adjunct_region_server_configured_to_bus: NOT SET, DEFAULT=0.
BBOM0001I adjunct_region_start_default_adjunct: NOT SET, DEFAULT=0.
BBOM0001I              cell_name: cl6481.
BBOM0001I        cell_short_name: CL6481.
BBOM0001I client_protocol_password: NOT SET.
BBOM0001I    client_protocol_user: NOT SET.
BBOM0001I client_ras_logstreamname: NOT SET.
BBOM0001I       clustered_server: NOT SET, DEFAULT=0.
BBOM0001I com_ibm_authMechanisms_type_OID: No OID for this mechanism.
BBOM0001I com_ibm_security_SAF_unauthenticated: NOT SET,
DEFAULT=WSGUEST.
BBOM0001I com_ibm_security_SAF_EJBROLE_Audit_Messages_Suppress: NOT
SET, DEFAULT=0.
BBOM0001I com_ibm_userRegistries_type: security:LocalOSUserRegistry.
BBOM0001I com_ibm_userRegistries_CustomUserRegistry_realm: NOT SET,
DEFAULT=CustomRealm.
BBOM0001I com_ibm_userRegistries_LDAPUserRegistry_realm: NOT SET,
DEFAULT=LDAPRealm.
BBOM0001I com_ibm_ws_logging_zos_errorlog_format_cbe: NOT SET,
DEFAULT=0.
BBOM0001I com_ibm_CSI_claim_ssl_sys_v2_timeout: NOT SET, DEFAULT=100.
BBOM0001I com_ibm_CSI_claim_ssl_sys_v3_timeout: NOT SET, DEFAULT=600.
BBOM0001I com_ibm_CSI_claimClientAuthenticationtype: NOT SET,
DEFAULT=SAFUSERIDPASSWORD.
```

Figure 21-2 WebSphere Application Server JESYSMSG output

- **CEEDUMP** - An exception in the address space may cause this section to be generated. It lists failure information including trace backs. (A *trace back* shows which functions were last called prior to the program failure.)
- **SYSOUT** - During normal processing, the SYSOUT should be empty, but there are situations that cause output to be written to this section. If the error log logstream cannot

connect, then the messages set to be written to the error log will be written to CERR, which goes to SYSOUT. Trace from the JVM when you set the environment variable JVM_DEBUG=1 and jvm_logfile is not set.

- **SYSPRINT** - The WebSphere for z/OS trace output can be written to SYSPRINT if the environment variable *ras_trace_outputLocation=SYSPRINT* is set. Figure 21-3 shows some sample data from the SYSPRINT file.

```
Processing Trace Settings File:
config/cells/cl6481/nodes/nd6481/servers/ws6481/trace.dat
Trace: 2005/04/22 07:08:07.453 01 t=7E3E88 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws390.orb.CommonBridge
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0011I JVM Build is J2RE 1.4.2 IBM z/OS Persistent Reusable VM
build cm142sr1a-20050209 (JIT enabled: jitc)
Trace: 2005/04/22 07:08:07.464 01 t=7E3E88 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: com.ibm.ws390.orb.CommonBridge
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0051I PROCESS INFORMATION:STC26724/WS6481S, ASID=103(0x67),
PID=68157916(0x41001dc)
Trace: 2005/04/22 07:08:07.489 01 t=7E3E88 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I java.vendor = IBMCorporation
Trace: 2005/04/22 07:08:07.492 01 t=7E3E88 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I os.name = z/OS
Trace: 2005/04/22 07:08:07.496 01 t=7E3E88 c=UNK key=P8 (13007002)
  ThreadId: 0000000a
  FunctionName: printProperties
  SourceId: com.ibm.ws390.orb.CommonBridge
  Category: AUDIT
  ExtendedMessage: BBOJ0077I sun.boot.class.path =
/WebSphereAL1/V6R0/BS01/AppServer/java/lib/core
```

Figure 21-3 WebSphere Application Server SYSPRINT trace data

JES writes the information to the job logs of the address spaces for the different regions. These include:

- For each server instance, there is one control region and 0 or more server regions (except for the DAEMON, for which there is only a control region).
- Control regions, to put it simply, handle communication, receiving requests from clients and sending back responses.
- Server regions are given the requests to process, so they do the actual work.
- There is also the base set of server instances, as well as the J2EE server instances.
- You may also have address spaces from local clients, the LDAP server, and your HTTP server.

21.1.2 Dumping the WebSphere Application Server address spaces

The dump requirements to assist with WebSphere Application Server problem diagnosis may require not only the dump to contain the specific WAS address spaces, but may also require that the associated OMVS, TCP/IP, and HTTP address spaces be dumped.

For example, the following dump command could be used to dump the related address spaces and associated data spaces:

```
DUMP COMM=(description of problem)
JOBNAME=(OMVS,WAS Serverproc,HTTP server,TCP/IP address
space),DSPNAME=('OMVS'.*),
SDATA=(CSA,GRSQ,LPA,NUC,PSA,RGN,SQA,TRT,SUM)
```

The WebSphere Application Server address space structure can comprise a server control region, a server daemon region, and multiple servant regions that may all be required to be dumped to ensure a thorough review of the problem.

You must ensure that the DUMP MAXSPACE system parameter is set to at least 2500M, as previously discussed, and that sufficient disk space is available to contain these large dumps.

The dump can be formatted using the IPCS VERBEXIT CBDATA verb name to display diagnostic data for the Component Broker element in WebSphere Application Server.

CBDATA includes the following:

- Display of the Component Broker Global control blocks
- Display of Component Broker address space control blocks
- Display of Component Broker address space control blocks with only one Component Broker TCB
- Display of ORB control block information

Other useful IPCS format commands to assist with WAS SVC dump analysis include:

- `ip verbx ledata 'nthread(*)'` - Lists the threads and their tracebacks.
- `ip ctrace comp(sysomvs) full` - Formats the ctrace for the OMVS component.
- `ip ctrace comp(sysomvs) options((sccounts))` - Produces a list of the syscalls together with their names

21.2 WebSphere Application Server CTRACE (SYSBBOSS)

WebSphere for z/OS uses the CTRACE component SYSBBOSS to capture and display trace data in trace data sets. You can send the CTRACE output to the spool (SYSPRINT) or to a data set. It is configured by environment variable parameter ras_trace_outputLocation.

To view or set the environment variables, use the WebSphere Application Server administrative console by selecting **Select Environment** → **Manage WebSphere Variables**.

Consider this example:

```
ras_trace_outputLocation=SYSPRINT | BUFFER | TRCFILE
```

It specifies where you want trace records to be sent: to SYSPRINT, or a memory buffer (BUFFER) whose contents will be later written to a CTRACE data set, or to a trace data set (TRCFILE) specified on the TRCFILE DD statement in the server's start procedure.

For servers, you can specify one or more values, separated by a space. For clients, you can specify SYSPRINT only.

Some other variables that might need to be set include:

- `ras_trace_BufferCount= n` - Specifies the number of trace buffers to allocate
- `ras_trace_BufferSize= n` - Specifies the size of a single trace buffer in bytes

The default values allocate 4 trace buffers with a size of 1MB for each buffer.

21.2.1 Executing the CTRACE for WebSphere

This section describes the procedures to follow if you want to obtain a formatted CTRACE that is prepared for debugging.

Planning for component tracing

- Create CTIBBOxx PARMLIB members for WebSphere for z/OS tracing. This is used only when ras_trace_outputLocation-BUFFER was specified.
- WebSphere for z/OS provides a default CTRACE PARMLIB member, in CTIBBO00. Following is the format of this member:

```
TRACEOPTS
/* Start a ctrace writer.  Remove comments to start the PROC */
/* during CB Series address space initialization. */
WTRSTART(BBOWTR)
/* Indicate that tracing is active for CB Series: */
ON
/* Connect to ctrace external writer (BBOWTR): */
WTR(BBOWTR)
```

> **Note:** You still have to start the BBOWTR started task yourself; WebSphere will not start it.

- Specify buffers for WebSphere for z/OS tracing.

Trace options

WebSphere Application Server for z/OS also allows you to set the trace options via the MVS **`MODIFY`** command. For example:

```
MODIFY was_server_name,TRACETOSYSPRINT=YES
```

```
MODIFY was_server_name,TRACEALL=n  (where n is the level of tracing)
```

These trace levels are 0 (none), 1 (exception), 2 (basic), and 3 (detailed tracing). Under normal conditions and in production, use 1 (exception). The level can also be set via the WebSphere Application Server administrative console in the *ras_trace_defaultTracingLevel* variable.

The *ras_trace_ctraceParms* variable specifies the PARMLIB member that contains the WebSphere Application Server for z/OS CTRACE connection and startup information. The options are SUFFIX which is a two-character suffix to be added to CTIBBO to form CTIBBOxx, or it is a fully specified name of a member of PARMLIB. The default is 00. This parameter is valid only for the daemon address space, and must be specified as a program environment variable.

Obtaining the trace data

- You first need to allocate the trace data set (DCB: PS, VB, 32756, 32760). Start the CTRACE writer address space. This is automatically done in the default WebSphere for z/OS PARMLIB member CTIBBO00. If the CTRACE writer was not started, you can do it now with the command:

  ```
  TRACE CT,WTRSTART=BBOWTR
  ```

  You will see this message:

  ```
  ITT110I INITIALIZATION OF TRACE WRITER BBOWTR COMPLETE.
  ```

- Start the daemon address space with the desired trace specifications.
- Start the WebSphere for z/OS servers.
- If you need to collect trace data for problem analysis, do the following:
  - Disconnect WebSphere for z/OS from CTRACE by using the operator command:

    ```
    TRACE CT,ON,COMP=SYSBBOSS
    REPLY x,WTR=DISCONNECT,END
    ```

  - Stop the CTRACE address space by using the operator command:

    ```
    TRACE CT,WTRSTOP=BBOWTR
    ```

  Figure 21-4 shows the syslog with the commands to disconnect WebSphere for z/OS from CTRACE, and to stop the CTRACE address space.

```
SY1      TRACE CT,ON,COMP=SYSBBOSS
SY1     *04 ITT006A SPECIFY OPERAND(S) FOR TRACE CT COMMAND.
SY1      REPLY 4,WTR=DISCONNECT,END
SY1      IEE600I REPLY TO 04 IS;WTR=DISCONNECT,END
SY1      ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT
 COMMAND WERE SUCCESSFULLY EXECUTED.
SY1      IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,016K)
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
SY1      TRACE CT,WTRSTOP=BBOWTR
SY1      ITT038I ALL OF THE TRANSACTIONS REQUESTED VIA THE TRACE CT
 COMMAND WERE SUCCESSFULLY EXECUTED.
SY1      IEE839I ST=(ON,0064K,00128K) AS=ON  BR=OFF EX=ON  MT=(ON,016K)
        ISSUE DISPLAY TRACE CMD FOR SYSTEM AND COMPONENT TRACE STATUS
        ISSUE DISPLAY TRACE,TT CMD FOR TRANSACTION TRACE STATUS
SY1      AHL904I THE FOLLOWING TRACE DATASETS CONTAIN TRACE DATA :
          ITSOWAS.BOSS.CTRACE
SY1      ITT111I CTRACE WRITER BBOWTR TERMINATED BECAUSE OF A WTRSTOP
 REQUEST
```

Figure 21-4 WAS CTRACE Syslog output

Note: We recommend that you disable CTRACE for better performance. If IBM Support asks for a CTRACE, you can enable it at that time.

See *WebSphere Application Server for z/OS V4.0.1: Messages and Diagnosis,* GA22-7837 for more information.

Viewing the CTRACE with the IPCS

From the Interactive Problem Control Facility (IPCS) Primary Option Menu panel, select option **0** (DEFAULTS) and enter the name of the trace data set, then press Enter. Issue the following IPCS command from option **6** (COMMAND):

```
CTRACE COMP(SYSBBOSS)FULL
```

If you are interested in only JRas data, enter the following IPCS command:

```
CTRACE COMP(SYSBBOSS)USEREXIT(JRAS)
```

MVS Interactive Problem Control System (IPCS) Commands, SA22-7594, describes how to use IPCS CTRACE, and contains a complete list of CTRACE command parameters.

Creating output data set

To send this output to a data set, you can use IPCS commands. The following steps describe the process:

- From any IPCS command line, execute these commands:

  ```
  TSO ALLOC FI(IPCSPRNT) SYSOUT(H)
  TSO ALLOC FI(IPCSTOC) SYSOUT(H)
  ```

- You need to have PRINT specified as one of the defaults:

  ```
  IP SETEDF PRINT NOTERM
  ```

  This causes output to only go to PRINT, and not to the screen. (You can use TERM instead, but then you have to scroll to the bottom of the output display to get all the output in the PRINT file.)

- When IPCSPRNT is used as the DDNAME (as in this case), the first command issued that causes output will OPEN the print file:

  ```
  IP CBF CVT
  ```

  This IPCS command is used to format the Communications Vector Table.

- Now you can format things and run execs, then close the print file:

  ```
  IP CLOSE PRINT
  ```

- You can now go to another ISPF window, get into SDSF, and put a question mark (?) next to your active userid. IPCSPRNT should be one of the DDNAMES present. Your output (for example, the preceding formatted CVT) should be there.

  You can now use the XDC command (put `xdc` next to your active userid) to copy this output to a data set.

- Free the files you allocated with the following IPCS commands:

  ```
  TSO FREE FI(IPCSPRNT)
  TSO FREE FI(IPCSTOC)
  ```

Note: If, after these steps, you leave IPCS, then come back and try to issue an IPCS command, you will receive the message: `Unable to open PRINT FILE(IPCSPRNT)`. If this occurs, go into IPCS option **0** for DEFAULTS and change PRINT to NOPRINT.

21.3 LDAP trace

For WebSphere Application Server for z/OS, the Lightweight Directory Access Protocol (LDAP) component of the z/OS Secureway Security Server provides the directory services for

the Java Naming and Directory Interface™ (JNDI), CORBA (MOFW) naming and interface repository services. The contents of the directory are stored in DB2 tables.

Like other z/OS components, the LDAP has a trace that is very useful when you encounter problems related to security and authorization issues.

The LDAP trace is started and stopped dynamically. You can also activate it when the LDAP starts. The output goes directly to the job log of the LDAP started task.

21.3.1 Starting an LDAP trace

When the LDAP server is running as a started task or from the z/OS shell, it is possible to dynamically turn the debugging facility on and off. The following command can be sent to the LDAP server from SDSF, or from the operator console. Note that if the command is entered from SDSF, it must be preceded by a slash (/). The command is:

```
f ldapsrv,appl=debug=nnnnn
```

In this command, nnnnn is the decimal value of the desired debug level, or a debug keyword, ERROR for example, which will trace all error conditions.

Note: To send the same command to the LDAP server in the OS/390 shell, you need to know the job name assigned to the process. To find out this job name, enter the following command from SDSF or from the operator's console:

```
d a,l
```

Once you find out the job name (which includes the user ID under which the LDAP server is running, and a suffix), use it to replace **ldapsrv** in the preceding command.

Prior to starting the server, the LDAP_DEBUG environment variable can be set. The server uses this value first. For example,

```
export LDAP_DEBUG='ERROR+TRACE'
```

When starting the server, the -d parameter can be specified. The debug level specified on this parameter either replaces, adds to, or deletes from the preceding debug level. For example,

```
s ldapsrv,parms='-d ERROR'
```

This replaces the current debug level that is either off or has been set by the LDAP_DEBUG environment variable with the new debug level of only ERROR. A list of all debug levels is provided in the *SecureWay Security Server LDAP Server Administration and Usage Guide,* SC24-5923.

Debug information will be added to the job log output associated with the LDAP server.

When you turn on the trace, the following message should appear in the console:

```
GLD0091I Successfully set debug level to xxxxx from console command.
```

To turn debug tracing off, enter the same command, but providing the value zero (0) for nnnnn; this message will appear:

```
GLD0091I Successfully set debug level to 0 from console command.
```

21.3.2 IBM HTTP Server logs and trace

The server error log includes errors encountered by the server's clients. The information can include timeout and access errors. Once you have eliminated an actual communications problem, this log should be reviewed.

The server saves the error log in the HFS. The file is specified in the httpd.conf file with the ErrorLog directive. For example:

```
ErrorLog /web/logs/errorlog
```

The server records activity in the access log files and stores them each day. At midnight the server closes the current access log and creates a new access log file. The access log contains entries for page request mode to the server.

The logging directive in httpd.conf called AccessLog points to the file where you want the access log to be saved. For example:

```
AccessLog /web/logs/accesslog
```

Figure 21-5 shows some sample data from the HTTP server job SYSOUT.

```
TCP......... Full local host name is wtsc61.itso.ibm.com
HTHostName gave "wtsc61.itso.ibm.com"; sc.hostname is "wtsc61.itso.ibm.com".
Welcome.....  Adding default welcome names
Timer....... table created.
Server..... will set SO_REUSEADDR for fast Bounce.
Error log... "/web/ds611/logs/httpd-errors.Apr232005" opened
AccessLog......... "/web/ds611/logs/httpd-log.Apr232005" opened
AgentLog......... "/web/ds611/logs/agent-log.Apr232005" opened
RefererLog....... "/web/ds611/logs/referer-log.Apr232005" opened
.
Writer...... 268-byte block arriving.
Writer...... converting 0-byte header.
Writer...... converted 268 characters.
level3write...socket=11, bufptr=22F12840, size=268, socketype=1
HTHandle.... Beginning SKWRITE
skWRITE..... requested to write 301 bytes on socket=11.
skWRITE..... wrote 301 bytes on socket=11.
level3write...SSL write socket=11,location=22F12840,size=268,status=268
HTTimer... off set->off socket 11.
Logging..... updating SNMP/PerfMon counters.
Logging....... fastpath off.
Logging....... Server_IP=-=
Logging....... Client_HOST=9.42.171.62=
Logging....... Request_Line=HEAD / HTTP/1.1=
Logging....... Status_Code=200=
Logging....... Method=HEAD=
Logging....... Type=text/html=
Logging....... Cookie==
Logging....... Referer==
Logging....... Agent==
V     9.42.171.62   443 HEAD        877 200 1114437268.818951 1114439069.721142
439069.769456 1114439069.803782 0000000000.000000 0000000000.000000
```

Figure 21-5 HTTP Server log data from SYSOUT

HTTP -vv tracing

The server trace has several levels of debugging (verbose, very verbose, verbose cache and debug), but the most used is the -vv trace.

There are two ways to start the -vv trace, as follows:

1. Use the -vv parameter in the started procedure of the server as follows:

```
//IMWPROC PROC LEPARM=,ICSPARM='-vv -r /web/httpd.conf'
//*****************************************************************
//WEBSRV EXEC PGM=IMWHTTPD,REGION=0K,TIME=NOLIMIT,
// PARM=(©&LEPARM/&ICSPARM©)
//*****************************************************************
//SYSIN DD DUMMY
//OUTDSC OUTPUT DEST=HOLD
//SYSPRINT DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSERR DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//STDOUT DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//STDERR DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//SYSOUT DD SYSOUT=*,OUTPUT=(*.OUTDSC)
//CEEDUMP DD SYSOUT=*,OUTPUT=(*.OUTDSC)
```

   When the server is started, it is in very verbose mode.

2. Alternatively, you can start the server with the following console command:

   ```
   f imwebsrv,appl=-vv
   ```

   In this command, imwebsrv is the name of your IBM HTTP Server. The following message appears: `IMW3518I Second level tracing (-vv) enabled.`

To stop the trace, issue this command:

```
f imwebsrv,appl=-nodebug
```

The following message is displayed: `IMW3508I Debug has been disabled for all modules.`

21.4 JVM debugging tools for z/OS

This section describes three problem analysis tools for the WebSphere for z/OS production environment:

- Svcdump.jar
- FindRoots/HeapRoots
- Dumpviewer GUI and jformat

The JVM sits at the core of the execution environment for WebSphere Application Server and most products that run under the z/OS USS run-time environment. These tools provide a non-IPCS interface to assist with SVC dump analysis, and while not providing the complex granularity of IPCS, will enable you to resolve many issues from the generated data. This can be performed under the z/OS JVM, or alternatively, by transferring the dump to a distributed JVM (either UNIX or Windows) and running the PD tools on these platforms.

The svcdump.jar file enables access directly to the binary SVC dump or transaction dumps created on z/OS without the need for intermediate software such as IPCS. There are three packages shipped in svcdump.jar:

- Dump utility: com.ibm.jvm.svcdump. The dump package will format out native and Java stacks for threads in dumped processes that include an instantiated JVM. The dump utility includes a function to print out other useful information, for example, in core trace buffers maintained by the JVM and the system trace, that mimic or extend the information that can be obtained with IPCS.
- FindRoots utility: com.ibm.jvm.ras.findroots.* This package provides multiple ways of formatting out the object graphs present in the Java-managed heap. This is critical for the sometimes difficult tasks of pinning down object leaks and in other ways making sense of heap occupancy.
- Java API can be used to write specific applications to perform some customized analysis at the business application level.

The svcdump.jar file is available for download from the following Web site, after completing the required IBM registration:

```
https://www14.software.ibm.com/webapp/iwm/web/preLogin.do?source=diagjava
```

This tool is under active development, so enhancements are being made to expand the functionality. It should be noted that this utility is provided “as-is”, and is *not* covered for defect support under your product support contract.

The svcdump utility is comprised of three files. They are:

- The svcdump.jar file itself.
- The doc.jar file, which contains documentation for the exposed API.
- libsvcdump.so, a DLL that allows the Java code to access an unformatted dump in an MVS data set rather than in the HFS. This avoids the requirement to provide a large HFS data set to copy the dump to, meaning that on z/OS, the original dump can be analyzed instead.

To use the dump utility, copy the three files in binary format to a suitable location in the HFS, or another JVM platform.

The following example was extracted from a SNAP dump of a CICS region with 2 JVMs active. The following command was issued and the output routed to a file with the inclusion of a > filename.txt directive.

```
java -Xmx512M -jar svcdump.jar DUMP.MV2C.CICSSC1.D050427.T185201.S00695
```

Figure 21-6 shows a sample of the data returned from a format of an SVC dump using the svcdump.jar utility.

```
Dump title: CICS DUMP: SYSTEM=IYK3ZSC1 CODE=MT0001   ID=1/0001
Time of dump: Wed Apr 27 14:52:03 EDT 2005
Jvm fullversion: J2RE 1.4.2 IBM z/OS Persistent Reusable VM build cm142-20040917
.
TCB 8aa0b0 tid 24c008e8 pthread id 1bfc0e0000000001 tid type 0x00000000 tid state
0x00000015 tid singled 0x00000040 jvmp 24a3a740 ee 24c00708 caa 2d72aac0 (non-xplink
waitstat)

Dsa        Entry     Offset    Function                      Module
---        -----     ------    --------                      ------
2d72be08   22e3c9b8  0004aa24  CEEOPCW
2d72bc58   22e85018  0000006e  CEEVH2OS
2d7cd220   24902970  00000040  pthread_cond_wait
2d7cd2a0   7bd2cd08  000002f6  ThreadUtils_BlockingSection
                        /u/sovbld/cm142/cm142-20040917/src/hpi/pfm/threads_utils.c
2d7cd320   7bd15280  00000372  sysSignalWait
                        /u/sovbld/cm142/cm142-20040917/src/hpi/pfm/interrupt_md.c
2d7cd3c0   7c118cd8  00000084  signalDispatcherThread
                        /u/sovbld/cm142/cm142-20040917/src/jvm/sov/xm/signals.c
2d7cd440   7c11a6f0  000002ee  xmExecuteThread
                       /u/sovbld/cm142/cm142-20040917/src/jvm/sov/xm/thr.c
2d7cd600   7c10d5a8  0000001c  threadStart
                     /u/sovbld/cm142/cm142-20040917/src/jvm/pfm/xe/common/xe_thread_md.c
2d7cd680   7bd2d0a0  000002fe  ThreadUtils_Shell
                     /u/sovbld/cm142/cm142-20040917/src/hpi/pfm/threads_utils.c
2d7cd720   22e85758  0000094e  CEEVROND
7f351e38   0001ebb8  00000932  CEEOPCMM
2d72b438   00000000  0ed03198  (unknown)
7f3507d0   0001ebb8  fffe1446  CEEOPCMM
7f350000   00000000  0ed03198  (unknown)
Java stack:

Method                                        Location
------                                        --------
java/lang/ref/Reference.process               Native Method
java/lang/ref/Reference.access$300            Reference.java:50
java/lang/ref/Reference$ReferenceHandler.run  Reference.java:130
```

Figure 21-6 SVC Dump format using the svcdump.jar utility

The dumpviewer GUI utility is another tool that can be useful in analyzing JVM problem data. It is supplied as part of the scvdump.jar utility. This can be invoked via the following command:

```
java -Xmx512m -cp svcdump.jar com.ibm.jvm.ras.dump.format.DvConsole -g
```

The svcdump.jar and dumpviewer utility, for those who are not comfortable with dump analysis via IPCS, are worth reviewing. As stated earlier, these are provided as-is, with no defect support through your normal software support channels.

22

Distributed platform problem determination

This chapter provides some guidelines to assist with the collection of relevant diagnostic data to be used for problem diagnosis and resolution for products running on UNIX and Windows.

This is not a definitive list of techniques, but covers most of the key diagnostic tools that are available for capturing and analyzing data in a WebSphere Application Server, WebSphere Business Integrator, and WebSphere MQSeries environment. It does not imply that the techniques outlined will guarantee that the data collected will be sufficient to solve the reported problem and does not exclude the fact that additional data may be requested to further the problem diagnosis process.

The diagnostic data collection process is complex. Keep in mind one of the fundamentals of efficient problem diagnosis that is sometimes called a *process of elimination* methodology. This implies that you may require additional diagnostic data to be captured using more refined and granular techniques as the analysis progresses.

This chapter describes some basic approaches to diagnose system problems. There are times, however, when your diagnosis will require the assistance of the IBM Support Center.

This chapter describes:

- AIX® tracing and core dumps
- WebSphere Application Server tracing
- WebSphere MQ on UNIX and Windows tracing
- WebSphere Business Integration Message Broker tracing
- LDAP tracing
- DB2 UDB on UNIX and Windows
- JAVA tracing
- WebSphere TXSeries®
- SYMREC file
- Encina® trace messages

22.1 What release am I running?

Different platforms use different commands to show you product information. With many environments now comprising combinations of different platforms, operating systems and products that all interact with the z/OS operating system in a distributed topology, we have included some procedures to enable you to find out the release and maintenance level for IBM products that could be an influencing factor on the z/OS problem you are investigating.

Important: Do not overlook the most obvious source of release information, and that is often recorded in the console or job log messages generated during startup of the operating system or product.

In **Windows**, the drop-down HELP menu will show you the product release, whereas other platforms require specific commands.

- **AIX:**

```
lslpp -l all   or  lslpp -l l grep mqseries
```

- **Solaris™:**

```
pkginfo | grep  -i mqseries
```

- **HP/UX:**

```
swlist -R
```

- **Compaq TRU64:**

```
setld -I
```

In **WebSphere MQ** the FDC (First Data Capture) file that is generated when a serious error is detected contains the release and maintenance level information in the **LVLS** and **CMVC** level field. For example, p521-CSD03H.

To find the JAVA version issue **`java -fullversion`** from the command line.

To find out the release of DB2 on UNIX and Windows issue the db2level command.

22.2 AIX tracing and core dumps

AIX has a system-call tracing facility, but it's more difficult to use since it can't be directed at a single process: turning on the tracing will cause the system to trace all calls made by all processes on the machine, and you can't tell which process made which call from the trace output. The relevant commands are **`trace`**, **`trcstop`**, and **`trcrpt`**; see the AIX documentation for usage information.

22.2.1 tcpdump and iptrace

The AIX **`tcpdump`** command is similar to the Solaris **`snoop`** command. Alternatively, the combination of **`iptrace`** and **`ipreport`** can be used on AIX. See the AIX documentation for details; in particular, try “man tcpdump” and “man iptrace.”

To trace all packets going between machines box1 and box2 using **`tcpdump`**, use the following command:

```
tcpdump -x host box1 and box2
```

22.2.2 UNIX systems core dump analysis

UNIX processes (including JVM process) will produce a system core dump as well as Java stack trace information in a process's working directory if it crashes. The system core dump can provide useful information as to why the process crashed, giving you a system view of a failing JVM process. However, the system core dump will not provide Java class information. Everything in the dump is C library oriented. The information provided for JVM process refers to Java's C libraries and not the reference Java class files.

22.2.3 Generating a core dump

It is often a requirement that if a core dump has not been automatically generated, one must manually trigger the core dump process. This can be achieved by issuing a **kill -6** command for the related pid.

If the current file and core ulimits have not been set to unlimited, it is recommended that the following be set to ensure successful core dump processing:

```
ulimit -c unlimited
ulimit -f unlimited
```

22.2.4 Looking at a system core dump

The core file on UNIX systems can be inspected using the dbx and gdb (GNU debugger) tools. The dbx tool is part of the AIX install (it might not be installed by default). On Sun™, dbx can be installed for an additional expense. The gdb is freeware and can be downloaded.

You can issue the **dbx** command with the Java binary executable file, normally <WAS_HOME>/java/bin/java, as the parameter. Figure 22-1 shows the commands to find the binary executable and invoke dbx.

```
# pwd
/usr/WebSphere/AppServer
# ls -alF core
-rw-r--r--   1 root system 425440811 Nov 07 19:40 core
# dbx /usr/WebSphere/AppServer/java/bin/java
Type 'help' for help.
reading symbolic information ...warning: no source compiled with -g
[using memory image in core]
Segmentation fault in strncpy.strncpy [/usr/lib/libbadjni.so] at 0xd32a3318
0xd32a3318 (strncpy+0x118) 9cc50001       stbu   r6,0x1(r5)
(dbx)
```

Figure 22-1 dbx invocation example

The sample was taken from an AIX 5.1 system and shows a segmentation fault occurred (SIGSEGV, signal # 11).

Either of the following command can be used to locate the true Java executable name of the core:

```
# strings core | grep COMMAND_LINE
```

or,

```
# strings core | more
```

Figure 22-2 shows how the dbx **where** command provides a stack trace of where the error occurred.

```
(dbx) where
strncpy.strncpy() at 0xd32a3318
bad_native_method() at 0xd32a315c
Java_itso_BadJni_badJniMethod(0x44b1fe4c, 0x4574c2e0) at 0xd32a31ac
mmisInvoke_V_VHelper(0x32c13ef0, 0x44b235fc, 0x1, 0x44b1fe4c, 0x4574c358) at
0xd2eb6fe4
mmipInvoke_V_V(??, ??) at 0xd2ed5e6c
```

Figure 22-2 dbx where command example

Figure 22-3 on page 203 shows the failure is to be found in the bad_native_method() method of the native JNI library module. The **registers** and **listi** commands are useful to view the system registers information and the instructions.

Other useful dbx commands include:

- **map**
- **thread**
- **thread info**
- **thread <thread_no>**
- **thread current <thread_no>**

Type `help` for help on a command or topic and `quit` to exit dbx.

```
(dbx) unset $noflregs
(dbx) registers
  $r0:0x00000000  $stkp:0x4574c1c8   $toc:0x494a13c4    $r3:0x00000000
  $r4:0x4574c20b    $r5:0xffffffff    $r6:0x00000045    $r7:0x00000074
  $r8:0x000000c2    $r9:0x00000058   $r10:0x40b55a74   $r11:0x000034e0
 $r12:0x00000000   $r13:0x00000000   $r14:0x456cf800   $r15:0x4574c350
 $r16:0x44b1fe4c   $r17:0x40f0b1ec   $r18:0x00000009   $r19:0x00000000
 $r20:0x00000041   $r21:0x5604a124   $r22:0x000000c1   $r23:0x00000001
 $r24:0x32451780   $r25:0x40b20600   $r26:0x44b235fc   $r27:0x30213b08
 $r28:0x3021c118   $r29:0x44b222b4   $r30:0x44b1fe4c   $r31:0x30212418
 $iar:0xd32a3318   $msr:0x0000d0b2    $cr:0x44824844 $link:0xd32a3160
 $ctr:0x0000000a   $xer:0x00000000
          Condition status = 0:g 1:g 2:l 3:e 4:g 5:l 6:g 7:g
  $fr0:0x0000000000000000   $fr1:0x3fe8000000000000   $fr2: 0x0000000000000000
  $fr3:0x0000000000000000   $fr4:0x0000000000000000   $fr5: 0x0000000000000000
  $fr6:0x0000000000000000   $fr7:0x0000000000000000   $fr8: 0x0000000000000000
  $fr9:0x0000000000000000  $fr10:0x0000000000000000  $fr11: 0x0000000000000000
 $fr12:0x0000000000000000  $fr13:0x3fe8000000000000  $fr14: 0x0000000000000000
 $fr15:0x0000000000000000  $fr16:0x0000000000000000  $fr17: 0x0000000000000000
 $fr18:0x0000000000000000  $fr19:0x0000000000000000  $fr20: 0x0000000000000000
 $fr21:0x0000000000000000  $fr22:0x0000000000000000  $fr23: 0x0000000000000000
 $fr24:0x0000000000000000  $fr25:0x0000000000000000  $fr26: 0x0000000000000000
 $fr27:0x0000000000000000  $fr28:0x0000000000000000  $fr29: 0x0000000000000000
 $fr30:0x0000000000000000  $fr31:0x0000000000000000 $fpscr: 0xa6100000
in strncpy.strncpy [/usr/lib/libbadjni.so] at 0xd32a3318
0xd32a3318 (strncpy+0x118) 9cc50001      stbu   r6,0x1(r5)

(dbx) listi .-40,.+40
0xd32a32f4 (strncpy+0xf4) 4182009c        beq   0xd32a3390 (strncpy+0x190)
0xd32a32f8 (strncpy+0xf8) 8cc40001       lbzu   r6,0x1(r4)
0xd32a32fc (strncpy+0xfc) 8ce40001       lbzu   r7,0x1(r4)
0xd32a3300 (strncpy+0x100) 8d040001       lbzu   r8,0x1(r4)
0xd32a3304 (strncpy+0x104) 8d240001       lbzu   r9,0x1(r4)
0xd32a3308 (strncpy+0x108) 2c060000       cmpi   cr0,0x0,r6,0x0
0xd32a330c (strncpy+0x10c) 2c870000       cmpi   cr1,0x0,r7,0x0
0xd32a3310 (strncpy+0x110) 2f080000       cmpi   cr6,0x0,r8,0x0
0xd32a3314 (strncpy+0x114) 2f890000       cmpi   cr7,0x0,r9,0x0
0xd32a3318 (strncpy+0x118) 9cc50001       stbu   r6,0x1(r5)
0xd32a331c (strncpy+0x11c) 4e400020    bdzgelr
0xd32a3320 (strncpy+0x120) 4182004c        beq   0xd32a336c (strncpy+0x16c)
0xd32a3324 (strncpy+0x124) 9ce50001       stbu   r7,0x1(r5)
0xd32a3328 (strncpy+0x128) 4e400020    bdzgelr
0xd32a332c (strncpy+0x12c) 4186004c        beq   cr1,0xd32a3378 (strncpy+0x178)
0xd32a3330 (strncpy+0x130) 9d050001       stbu   r8,0x1(r5)
0xd32a3334 (strncpy+0x134) 4e400020    bdzgelr
0xd32a3338 (strncpy+0x138) 419a004c        beq   cr6,0xd32a3384 (strncpy+0x184)
0xd32a333c (strncpy+0x13c) 9d250001       stbu   r9,0x1(r5)
0xd32a3340 (strncpy+0x140) 4e400020    bdzgelr
0xd32a3344 (strncpy+0x144) 419e004c        beq   cr7,0xd32a3390 (strncpy+0x190)
(dbx) quit
```

Figure 22-3 dbx registers and listi command example

22.2.5 Ensuring that a good core file is generated

After a crash, if there is no core file or the output from dbx shows that the core file is truncated, ensure that:

1. The file system containing the core file has enough free space. The size required depends on your WebSphere configuration environment, but it is recommended that you have over 500 MB.
2. On AIX, ensure that Enable full CORE dump is set to “true” in the system environment. You can do this via **smitty** by issuing the following command, which will then present the System Management Interface Tool display:

   ```
   # smitty chgsys
   ```

   Alternatively, you can use the following commands:

   ```
   # lsattr -El sys0 | grep fullcore
   fullcore     false         Enable full CORE dump  True
   # chdev -a fullcore=true -l sys0
   sys0 changed
   # lsattr -El sys0 | grep fullcore
   fullcore     true          Enable full CORE dump  True
   ```

3. The owner of the running process must have write permission to the directory to which the process dumps the core.
4. Both the maximum file and coredump size specifications must be large enough to process the dump.

   ```
   # ulimit -a
   time(seconds)        unlimited
   file(blocks)         2097151 <-- !
   data(kbytes)         131072
   stack(kbytes)        32768
   memory(kbytes)       32768
   coredump(blocks)     2097151 <-- !
   nofiles(descriptors) 2000
   ```

   You can change the file and coredump maximum value using the following commands:

   ```
   # ulimit -f nnnnnnn
   # ulimit -c nnnnnnn
   ```

   Note: For the purpose of capturing a specific instance of a core dump, we recommend setting the file and core ulimit to “unlimited.”

   The directory where the process dumps the core is the current working directory of the process that is running. In WebSphere Application Server V5, it is normally the <WAS_HOME> directory.

Monitoring a running process with dbx

Another use of dbx is to monitor a running process. The **-a** parameter allows the debug program to be attached to a process that is running. To attach the debug program, you need authority to use the **kill** command on this process. Use the **ps** command to determine the process ID. If you have permission, the **dbx** command interrupts the process, determines the full name of the object file, reads in the symbolic information, and prompts for commands.

Figure 22-4 shows the procedure that can be used to monitor a running process using dbx.

```
# ps -ef|grep java
  root 18088     1 0 Nov 07  pts/2  9:06 /usr/WebSphere/... dmgr
  root 20648     1 6 Nov 07  pts/2 39:58 /usr/WebSphere/... m10df51f
  root 22234 20648 2 Nov 08  pts/2 32:19 /usr/WebSphere/... server1

# dbx -a 22234
Waiting to attach to process 22234 ...
Successfully attached to java.
Type 'help' for help.
reading symbolic information ...warning: no source compiled with -g

stopped in _event_sleep at 0xd00549dc
0xd00549dc (_event_sleep+0x90) 80410014        lwz   r2,0x14(r1)
(dbx) where
_event_sleep(??, ??, ??, ??, ??) at 0xd00549dc
_event_wait(??) at 0xd0054ec8
_cond_wait_local(??, ??, ??) at 0xd0060e04
_cond_wait(??, ??, ??) at 0xd0061298
pthread_cond_wait(??, ??) at 0xd0061fbc
condvarWait(??, ??, ??) at 0xd3056160
sysMonitorWait(??, ??, ??, ??) at 0xd305511c
lkMonitorWait(??, ??, ??, ??) at 0xd2f39724
JVM_MonitorWait(??, ??, ??, ??) at 0xd2eab14c
(dbx) detach
#
```

Figure 22-4 dbx -a procedure to monitor a running process

To continue execution of the application and exit dbx, type `detach` instead of quit. (If you typed `quit` to exit, the process would stop running.

Note: Do not use the command **`dbx -a <pid>`** on heavily loaded systems because it could be temporarily blocked.

22.2.6 errpt command

On AIX, the **`errpt`** command generates an error report from entries in a system error log. If you have a system core dump, the event would be logged in the system error log. Figure 22-5 on page 206 shows an example of the errpt output.

```
# errpt -a > /tmp/errpt.txt
# vi /tmp/errpt.txt
....
---------------------------------------------------------------------------
LABEL:          CORE_DUMP
IDENTIFIER:     C60BB505

Date/Time:       Thu Nov  7 19:40:49 EST
...
Detail Data
SIGNAL NUMBER
          11
USER'S PROCESS ID:
       22226
...
PROGRAM NAME
java
ADDITIONAL INFORMATION
strncpy 118
bad_nativ 20
Java_itso 18
mmisInvok 2A4
entryCmp FFFFE688
??
...
SYMPTOM CODE
PCSS/SPI2 FLDS/java SIG/11 FLDS/strncpy VALU/118 FLDS/bad_nativ
```

Figure 22-5 AIX errpt -a example

A portion of the name of the native methods processing at the time of the JVM crash is shown in the error report. Even though the **errpt** command does not provide the fully qualified name, it is enough to start tracing the problem.

22.3 WebSphere Application Server

WebSphere Application Server writes system messages to several general purpose logs. These include the JVM logs, the process logs, and the IBM service log.

The JVM logs are created by redirecting the System.out and System.err streams of the JVM to independent log files. WebSphere Application Server writes formatted messages to the System.out stream. In addition, applications and other code can write to these streams using the print() and println() methods defined by the streams. Some JDK™ built-ins such as the printStackTrace() method on the Throwable class can also write to these streams. Typically, the System.out log is used to monitor the health of the running application server. The System.out log can be used for problem determination, but we recommend using the IBM Service log and the advanced capabilities of the Log Analyzer instead. The System.err log contains exception stack trace information that is useful when performing problem analysis.

Since each application server represents a JVM, there is one set of JVM logs for each application server and all of its applications, located by default in the installation_root/logs/server_name directory. In the case of a WebSphere Application Server

Network Deployment configuration, JVM logs are also created for the deployment manager and each node manager, since they also represent JVMs.

The process logs are created by redirecting the stdout and stderr streams of the process to independent log files. Native code, including the Java Virtual Machine itself, writes to these files. As a general rule, WebSphere Application Server does not write to these files. However, these logs can contain information relating to problems in native code or diagnostic information written by the JVM.

As with JVM logs, there is a set of process logs for each application server since each JVM is an operating system process, and in the case of a WebSphere Application Server Network Deployment configuration, a set of process logs for the deployment manager and each node manager.

The IBM service log contains both the WebSphere Application Server messages that are written to the System.out stream and some special messages that contain extended service information that is normally not of interest, but can be important when analyzing problems. There is one service log for all WebSphere Application Server JVMs on a node, including all application servers. The IBM Service log is maintained in a binary format and requires a special tool to view. This viewer, the Log Analyzer, provides additional diagnostic capabilities. In addition, the binary format provides capabilities that are utilized by IBM support organizations.

In addition to these general purpose logs, WebSphere Application Server contains other specialized logs that are very specific in nature and are scoped to a particular component or activity. For example, the HTTP server plug-in maintains a special log. Normally, these logs are not of interest, but you might be instructed to examine one or more of these logs while performing specific problem determination procedures.

22.3.1 Reviewing the JVM logs

The JVM logs are written as plain text files. Therefore there are no special requirements to view these logs. They are located in the installation_directory/logs/applicationServerName directory, and by default are named SystemOut.log and SystemErr.log.

There are two techniques that you can use to view the JVM logs for an application server:

1. Use the administrative console. This supports viewing the JVM logs from a remote machine.

   To view the JVM logs from the administrative console:

   a. Start the administrative console. Click **Troubleshooting** → **Logging and Tracing** in the console navigation tree, then click **Server** → **JVM Logs**.

   b. Select the **Runtime** tab.

   c. Click **View** corresponding to the log you want to view.

2. Use a text editor on the machine where the logs are stored. To do this:

   a. Go to the machine where the logs are stored.

   b. Open the file in a text editor or drag and drop the file into an editing and viewing program.

22.3.2 Interpreting the JVM log data

The JVM logs contain print data written by applications. The application can write this data directly in the form of System.out.print(), System.err.print(), or other method calls. The

application can also write data indirectly by calling a JVM function, such as an Exception.printStackTrace(). In addition, the System.out JVM log contains system messages written by the WebSphere Application Server.

You can format application data to look like WebSphere Application Server system messages using the Installed Application Output field of the JVM Logs properties panel, or as plain text with no additional formatting. WebSphere Application Server system messages are always formatted.

Depending on how the JVM log is configured, formatted messages can be written to the JVM logs in either basic or advanced format.

JVM log message formats

Formatted messages are written to the JVM logs in one of two formats:

- Basic format - The format used in earlier versions of WebSphere Application Server.
- Advanced format - Extends the basic format by adding information about an event, when possible.

Basic and advanced format fields

Basic and Advanced Formats use many of the same fields and formatting techniques. The fields that may be found in these formats are as follows:

- **TimeStamp** - The timestamp is formatted using the locale of the process where it is formatted. It includes a fully qualified date (for example YYMMDD), 24 hour time with millisecond precision, and a time zone.
- **ThreadId** - Eight character hexadecimal value generated from the hash code of the thread that issued the message.
- **ShortName** - Abbreviated name of the logging component that issued the message or trace event. This is typically the class name for WebSphere Application Server internal components, but can be some other identifier for user applications.
- **LongName** - The full name of the logging component that issued the message or trace event. This is typically the fully qualified class name for WebSphere Application Server internal components, but can be some other identifier for user applications.
- **EventType** - A one character field that indicates the type of the message or trace event. Message types are in upper case. Possible values include:

  A Audit message
  I Informational message
  W Warning message
  E Error message
  F Fatal message
  O Message that was written directly to System.out by the user application or internal components
  R Message that was written directly to System.err by the user application or internal components
  U Special message type used by the message logging component of the WebSphere Application Server run time
  Z A placeholder to indicate the type was not recognized

- **ClassName** - The class that issued the message or trace event.
- **MethodName** - The method that issued the message or trace event.
- **Organization** - The organization that owns the application that issued the message or trace event.

- **Product** - The product that issued the message or trace event.
- **Component** - The component within the product that issued the message or trace event.

Viewing the JVM logs

To view the JVM logs, click **Troubleshooting → Logs and Trace → Server → JVM Logs.**

File Name specifies the name of one of the log files. The first file name field specifies the name of the System.out log. The second file name field specifies the name of the System.err file.

Click the **View** button on the Runtime tab to view the contents of a selected log file.

Process logs

WebSphere Application Server processes contain two output streams that are accessible to native code running in the process. These streams are the stdout and stderr streams. Native code, including the JVM, may write data to these process streams. In addition, the JVM-provided System.out and System.err streams can be configured to write their data to these streams also.

By default, the stdout and stderr streams are redirected, at application server startup, to log files that contain text written to the stdout and stderr streams by native modules (dlls,.exes, UNIX libraries, and other modules). By default, these files are stored as installation_root/logs/applicationServerName/native_stderr.log and native_stdout.log.

This is a change from previous versions of WebSphere Application Server, which by default had one log file for both JVM standard output and native standard output, and one log file for both JVM standard error and native error output.

Viewing the service log

The service log is a special log written in a binary format. You cannot view the log directly using a text editor. You should never directly edit the service log, as doing so will corrupt the log. To move the service log from one machine to another, you must use a mechanism like FTP, which supports binary file transfer.

You can view the service log in either of the following ways:

1. Using the Log Analyzer tool to view the service log. This tool provides interactive viewing and analysis capability that is helpful in identifying problems. This is the recommended method.
2. If you are unable to use the Log Analyzer tool, you can use the Showlog tool to convert the contents of the service log to a text format that you can then write to a file or dump to the command shell window. The steps for using the Showlog tool are the following:
   a. Open a shell window on the machine where the service log resides.
   b. Change directory to install_directory/bin where install_directory is the fully qualified path where the WebSphere Application Server product is installed.
   c. (Optional) Run the showlog script with no parameters to display usage instructions.

      On Windows systems, the script is named <samp>showlog.bat. On UNIX systems, the script is named <samp>showlog.sh.
   d. Run the showlog script with the appropriate parameters. For example:
      i. To display the service log contents to the shell window, use the invocation showlog service_log_filename. If the service log is not in the default location, you must fully qualify the service_log_filename.

ii. To format and write the service log contents to a file use the invocation showlog service_log_filename output_filename. If the service log is not in the default location, you must fully qualify the service_log_filename.

Log analyzer

To take advantage of the Log Analyzer's browsing and analysis capabilities, start the Log Analyzer tool as follows:

- On UNIX systems: installation_root/bin/waslogbr
- On Windows systems: installation_root/bin/waslogbr.bat

Select **File** → **Open**, and select the file /logs/activity.log. You can also browse to activity logs from other WebSphere Application Server installations, and even other versions of the product. Expand the tree of administrative and application server logging sessions. Uncolored records are normal, yellow records are warnings, and pink records are errors. If you select a record, you will see its contents, including the basic error or warning message, date, time, which component logged the record, and which process (the administrative server or an application server) it came from, in the upper right-hand pane.

The activity log does not analyze any other log files, such as default_stderr.log or tracefile. To analyze these records, right-click on a record in the tree on the left (click on the UnitOfWorkView at the top to get them all), and select Analyze. Any records with a green check mark next to them match a record in the symptom database. When you select a checked record, you will see an explanation of the problem in the lower right-hand pane.

Updating the symptom database

The database of known problems and resolutions used when you click the Analyze menu item is periodically enhanced as new problems come to light and new versions of the product are introduced. To ensure that you have the latest version of the database, select the **File** → **Update Database** → **WebSphere Application Server Symptom Database** menu item from within the log analyzer tool at least once a month. Users who have just installed the product and have never run the update should do so immediately, since updates may have occurred since the version was first released.

The knowledge base used for analysis is built gradually from problems reported by users. For a recently released version of the product, you might not find any analysis hits. However, the Log Analyzer tool still provides a way to see all error messages and warnings from all components in a convenient, user-friendly way.

22.3.3 Collector tool

The *collector tool* gathers information about your WebSphere Application Server installation and packages it in a Java archive (JAR) file that you can send to IBM Customer Support to assist in determining and analyzing your problem. Information in the JAR file includes logs, property files, configuration files, operating system and Java data, and the presence and level of each software prerequisite.

WebSphere Application Server 5.0.2 introduced the `-Summary` option for the collector tool to produce a smaller text file and console version of some of the information in the Java archive (JAR) file that the tool produces without the -Summary parameter. You can use the collector summary option to retrieve basic configuration and prerequisite software level information.

22.4 Debugging with the Application Server toolkit

The Application Server Toolkit, included with the WebSphere Application Server on a separately-installable CD, includes debugging functionality that is built on the Eclipse workbench. It includes the *WebSphere Application Server debug adapter* that allows you to debug Web objects that are running on WebSphere Application Server and that you have launched in a browser. These objects include EJBs, JSPs, and servlets, as follows:

- The *JavaScript™ debug adapter* enables server-side JavaScript debugging.
- The *Compiled language debugger* allows you to detect and diagnose errors in compiled-language applications.
- The *Java development tools (JDT) debugger* allows you to debug Java.

All of the debug components in the Application Server Toolkit can be used for debugging locally and for remote debugging.

22.5 WebSphere Application Server tracing

WebSphere Applicatrion Server tracing can be used to obtain detailed information about the execution of WebSphere Application Server components, including application servers, clients, and other processes in the environment.

Trace files show the time and sequence of methods called by WebSphere Application Server base classes, and you can use these files to pinpoint the failure.

Collecting a trace is often requested by IBM technical support personnel. If you are not familiar with the internal structure of WebSphere Application Server, the trace output might not be meaningful to you.

22.5.1 Enabling tracing

Trace for an application server process is enabled while the server process runs by using the administrative console. You can configure the application server to start in a trace-enabled state by setting the appropriate configuration properties. You can only enable trace for an application client or stand-alone process at process startup.

Trace strings

By default, the trace service for all WebSphere Application Server components is disabled. To change the current state of the trace service, you must modify the trace string that is passed to the trace service. This trace string encodes the information detailing which level of trace to enable or disable and for which components.

The Trace strings can be manually typed, or constructed with the administrative console GUI. Trace strings must be formatted as follows:

```
TRACESTRING=COMPONENT_TRACE_STRING [:COMPONENT_TRACE_STRING ] *
COMPONENT_TRACE_STRING=COMPONENT_NAME=LEVEL=STATE [,LEVEL=STATE ] *
```

Where:

- `LEVEL` = all | entryExit | debug | event
- `STATE` = enabled | disabled
- `COMPONENT_NAME` = COMPONENT | GROUP

The COMPONENT_NAME is the name of a component or group registered with the trace service. Typically, WebSphere Application Server components register using a fully qualified Java classname, for example com.ibm.servlet.engine.ServletEngine. In addition, you can use a wildcard character of asterisk (*) to terminate a component name and indicate multiple classes or packages. For example, use a component name of com.ibm.servlet.* to specify all components whose names begin with com.ibm.servlet.

Figure 22-6 shows some examples of valid trace strings.

```
com.ibm.ejs.ras.ManagerAdmin=debug=enabled
com.ibm.ejs.ras.ManagerAdmin=all=enabled,event=disabled
com.ibm.ejs.ras.*=all=enabled
com.ibm.ejs.ras.*=all=enabled:com.ibm.ws.ras=debug=enabled,entryexit=enabled
```

Figure 22-6 Trace string examples

Trace string examples

Several format rules apply to trace strings:

- Trace strings cannot contain blanks.
- Trace strings are processed from left to right.

The following example enables all trace for all components whose names start with abc, then disables event tracing for those same components.

```
abc.*=all=enabled,event=disabled
```

This is equivalent to:

```
abc.*=debug=enabled,entryexit=enabled
```

22.5.2 Enabling trace at server startup

The diagnostic trace configuration settings for a server process determine the initial trace state for the server process. The configuration settings are read at server startup and used to configure the trace service. You can also change many of the trace service properties or settings while the server process is running.

The following process can be used to enable trace at server startup:

1. Start the administrative console.
2. Click **Troubleshooting** → **Logs and Trace** in the console navigation tree, then click **Server** → **Diagnostic Trace**.
3. Click **Configuration**.
4. Select the Enable Trace check box to enable trace, clear the check box to disable trace.
5. Set the trace specification to the desired state by entering the proper TraceString.
6. Select whether to direct trace output to either a file or an in-memory circular buffer.
7. (Optional) If the in-memory circular buffer is selected for the trace output, set the size of the buffer, specified in thousands of entries. This is the maximum number of entries that will be retained in the buffer at any given time.
8. (Optional) If a file is selected for trace output, set the maximum size in megabytes to which the file should be allowed to grow. When the file reaches this size, the existing file will be closed, renamed, and a new file with the original name reopened. The new name of the

file will be based upon the original name with a timestamp qualifier added to the name. In addition, specify the number of history files to keep.

9. Select the desired format for the generated trace.
10. Save the changed configuration.
11. Start the server.

22.5.3 Enabling trace on a running server

You can modify the trace service state that determines which components are being actively traced for a running server by using the following procedure.

1. Start the administrative console.
2. Click **Troubleshooting** → **Logging and Tracing** in the console navigation tree, then click **Server** → **Diagnostic Trace**.
3. Select the **Runtime** tab.
4. (Optional) Select the Save Trace check box if you want to write your changes back to the server configuration.
5. Change the existing trace state by changing the trace specification to the desired state.
6. (Optional) Configure the trace output if a change from the existing one is desired.
7. Click **Apply**.

22.5.4 Enabling trace on an application client or stand-alone process

Many stand-alone processes and the WebSphere J2EE application client define a process-specific mechanism for enabling trace. To enable trace and message logging for application clients or processes that have not defined such a mechanism, add the `-DtraceSettingsFile=filename` system property to the startup script or command.

For example, to trace the stand-alone client application program named com.ibm.sample.MyClientProgram, enter the command:

```
java -DtraceSettingsFile=MyTraceSettings.properties com.ibm.samples.MyClientProgram
```

The file identified by filename must be a properties file placed in the classpath of the application client or stand-alone process. An example file is provided, as follows:

```
ininstall_root/properties/TraceSettings.properties
```

You can optionally set the following parameters to change the output defaults:

- Configure the MyTraceSettings.properties file to send trace output to a file. The traceFileName property in the trace settings file specifies one of two options:
  - The fully qualified name of an output file. For example, traceFileName=c:\\MyTraceFile.log. You must specify this property to generate visible output.
  - stdout. When specified, output is written to system.out.
- Specify a trace string for writing messages. The trace string property specifies a startup trace specification, similar to that available on the server.

22.5.5 JMS tracing within WebSphere

To enable JMS tracing within WebSphere:

1. From the WebSphere administrator's console select:

   Default server → Service → Trace services

2. Set:

   ```
   com.ibm.ejs.jms.*=all=enabled
   ```

Note: This provides WebSphere JMS Wrapper trace only. If you need full JMS trace use the programmatic method.

22.6 WebSphere MQ on UNIX and Windows

WebSphere MQSeries in most installations runs as a client/server application that incorporates many different platforms and operating systems that interact with each other. You will often need to review data from many of the components that make up the WebSphere MQSeries environment. These could include other z/OS systems, UNIX systems from many other vendors, or Windows.

The most important diagnostic data to review is written to the MQ error log and the associated FDC files. These diagnostic files may be created on both the Server (that is, where the Queue Manager resides) and on the Client, so it is important to review both Client and Server for the relevant files.

22.6.1 WebSphere MQSeries error logs

WebSphere MQSeries uses a number of error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use.

The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

In WebSPhere MQSeries for UNIX systems:

The queue manager can be available or not available as follows:

- If the queue manager name is known and the queue manager is available, error logs are located in:

  /var/mqm/qmgrs/qmname/errors

- If the queue manager is not available, error logs are located in:

  /var/mqm/qmgrs/@SYSTEM/errors

- If an error has occurred with a client application, error logs are located on the client's root drive in:

  /var/mqm/errors

WebSphere MQSeries for Windows

Assuming that MQSeries has been installed on the C drive in the MQM directory, the following will help you locate the required logs:

- If the queue manager name is known and the queue manager is available, error logs are located in:

  c:\mqm\qmgrs\qmname\errors

- If the queue manager is not available, error logs are located in:

  c:\mqm\qmgrs\@SYSTEM\errors

- If an error has occurred with a client application, error logs are located on the client's root drive in:

  c:\mqm\errors

Windows NT, 2000, and XP

In WebSphere MQSeries for Windows NT/2000/XP, an indication of the error is also added to the Application Log, which can be examined with the Event Viewer application provided with Windows NT®.

You can also examine the Registry to help resolve any errors. The Registry Editor supplied with Windows NT allows you to filter errors that are placed in the Event Log by placing the code in the following Registry entry:

HKEY_LOCAL_MACHINE\SOFTWARE\IBM\MQSeries\CurrentVersion\IgnoredErrorCodes

For example, to ignore error 5000, add AMQ5000 to the list.

22.6.2 WebSphere MQ JAVA tracing

WebSphere MQ JMS applications normally invoke trace by using command line arguments to the **java** command. For example:

```
java -DMQJMS_TRACE_LEVEL=base   myJMSApplication
```

Sometimes this is not possible, for example, when the JMS application is started as a servlet within an application server. In this case it may be appropriate to invoke trace from within the source code.

This can be done as follows:

```
com.ibm.mq.jms.services.ConfigEnvironment.start();
     //uses MQJMS TRACE_LEVEL and _DIR from the system properties
com.ibm.mq.jms.services.ConfigEnvironment.start(String level);
     //uses MQJMS_TRACE_DIR; the level parameter should "on", or "base"
com.ibm.mq.jms.services.ConfigEnvironment.stop();
```

The value of MQJMS_TRACE_DIR can be set programmatically with something like:

```
java.util.Properties.props = System.getProperties();
props.put("MQJMS_TRACE_DIR", "my/directory/name";
System.setProperties(props);
```

The filename is fixed to mqjms.trc; only the directory can be modified in this manner.

22.6.3 AIX MQ tracing

Unlike the other WebSphere MQSeries distributed platforms that use MQ-supplied internal tracing utilities, MQ on the AIX platform uses AIX tracing to capture the MQ trace data. The procedure is as follows:

1. Issue **`ps -ef`** to display parent processes. This will assist with correlating the active MQ processes with the trace data since all processes will be traced.
2. There are two ways to run MQ tracing on AIX: Interactive and Asynchronous. The commands to invoke them on the program myprog and then end the trace are as follows:
   a. Interactive trace:

```
trace -j30D,30E -o trace file
->!myprog
->q
```

   b. Asynchronous trace

```
trace -a -j30D,30E -o trace file
myprog
trcstop
```

22.6.4 Formatting the MQ trace file

You can format the trace file with the command:

```
trcrpt -t mqmtop/lib/amqtrc.fmt trace.file > report.file
```

The report.file is the name of the file where you want to put the formatted trace output.

Note: All MQSeries activity on the machine is traced while the trace is active.

Figure 22-7 on page 217 shows a sample of the data contained in the MQ for AIX trace.

```
  30E    0.250173285       0.915095   MQS Thread stack
  30E    0.250661751       0.488466   pid(10004)
  30E    0.251111529       0.449778   MQS -> zlaMainThread
  30E    0.251151575       0.040046   MQS -> zlaProcessMessage
  30E    0.251152096       0.000521   MQS -> zlaProcessMQIRequest
  30E    0.251152532       0.000436   MQS -> zlaMQGET
  30E    0.251153015       0.000483   MQS -> zsqMQGET
  30E    0.251191184       0.038169   MQS -> kpiMQGET
  30E    0.251253521       0.062337   MQS -> kqiWaitForMessage
  30E    0.251254039       0.000518   MQS -> kqiWaitForABit
  30E    0.251291476       0.037437   MQS -> apiLockExclusive
  30E    0.251940443       0.648967   MQS Returning an error to the AI Layer:
CompCode 2 Reason 7f1 MQRC_NO_MSG_AVAILABLE
  30E    0.252257186       0.316743   hev=1::0::0-275688
  30E    0.252258418       0.001232   hev=1::0::0-275604
  30E    0.252263620       0.005202   hev=1::0::0-275688 TimeOut(-1)
  30E    0.252353122       0.089502   MQS Thread stack
  30E    0.252366827       0.013705   pid(10004)
  30E    0.253368728       1.001901   MQS -> ccxTcpResponder
  30E    0.253369247       0.000519   MQS -> ccxResponder
  30E    0.253369732       0.000485   MQS -> rrxResponder
  30E    0.253370165       0.000433   MQS -> rriMQIServer
  30E    0.253370603       0.000438   MQS -> MQGET
  30E    0.253371040       0.000437   MQS -> zstMQGET
  30E    0.253371499       0.000459   MQS -> ziiMQGET
  30E    0.253371949       0.000450   MQS -> zcpDeleteMessage
```

Figure 22-7 WebSphere AIX Trace Data

Tip: A good starting point is to search for the string MQRC which identifies MQ reason codes. While many of these do not indicate a serious error, as in the illustrated case, the key to your problem might be as simple as finding the reason code. The reason code is displayed in HEX, for example, 7f1. The messages are documented in the manual in hex, X'07F1' and decimal, 2033.

22.6.5 MQ Tracing on UNIX and Windows (excluding AIX)

To start WebSphere MQ tracing issue the **strmqtrc** command.

The output files in UNIX are always created in the directory /var/mqm/trace. The filename takes the format AMQxxxxx.TRC. One file is created for each active MQ process.

In Windows the trace files are located in the c:\mqm\errors directory (assuming drive C has been used).

To end the WebSphere MQ trace issue the **endmqtrc** command.

To format the WebSphere MQ trace file issue the **dspmqtrc** command. You must specify the input (unformatted trace file, that is, AMQxxxxx.TRC) and the output filename.

Figure 22-8 shows some MQ trace data from a Solaris system.

```
14:26:34.48031     23652.1 MQOPEN <<
14:26:34.48045     23652.1 Hconn         : Input  Parm
14:26:34.48059     23652.1 Objdesc:
14:26:34.48073     23652.1 0x0000:  4f442020 00000001 00000001 00000000  |OD............|
14:26:34.48073     23652.1 0x0010:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0020:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0030:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0040:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0050:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0060:  00000000 00000000 00000000 414d512e  |............AMQ.|
14:26:34.48073     23652.1 0x0070:  2a000000 00000000 00000000 00000000  |*...............|
14:26:34.48073     23652.1 0x0080:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x0090:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x00a0:  00000000 00000000 00000000 ff3a0240  |............:.@.|
14:26:34.48073     23652.1 0x00b0:  000000d4 00000000 fedae740 00050fc0  |... .......@....|
14:26:34.48073     23652.1 0x00c0:  ffbec3f0 fed89c70 00000000 00000000  |.....  p........|
14:26:34.48073     23652.1 0x00d0:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x00e0:  00000000 00000000 00000000 00000000  |................|
14:26:34.48073     23652.1 0x00f0:  0001d9b0 00000000 0220002a 0020002a  |.. ...... .*. .*|
14:26:34.48073     23652.1 0x0100:  00000000 00005c64 00000001 00000001  |......\d........|
14:26:34.48073     23652.1 0x0110:  ffbec460 fee4d96c 0004ca5c 00000008  |...`.. l.. \....|
14:26:34.48073     23652.1 0x0120:  00000058 00fefeff feabc6a0 0000ff00  |...X............|
14:26:34.48073     23652.1 0x0130:  ffbec468 0002a314 30de0018 0220002a  |...h....0.... .*|
14:26:34.48073     23652.1 0x0140:  3f97587a 0000b438 00000000 00005c64  |?.Xz...8......\d|
14:26:34.48090     23652.1 Options       : Input  Parm
14:26:34.48103     23652.1 Hobj:
14:26:34.48117     23652.1 0x0000:  00000000                           |....            |
14:26:34.48130     23652.1 Compcode:
14:26:34.48144     23652.1 0x0000:  00000002                           |....            |
14:26:34.48157     23652.1 Reason:
14:26:34.48171     23652.1 0x0000:  00000825                           |...%            |
14:26:34.48185     23652.1 --}! zstMQOPEN rc=MQRC_UNKNOWN_OBJECT_NAME
14:26:34.48282     23652.1 ObjHandle=0 ObjType=1 ObjName=
14:26:34.48303     23652.1 -}! MQOPEN rc=MQRC_UNKNOWN_OBJECT_NAME
14:26:34.48402     23652.1 -{  xcsCheckPointer
14:26:34.48514     23652.1 -}  xcsCheckPointer rc=OK
14:26:34.49019     23652.1 ImqObject::open (error): reason code 2085 on MQOPEN for , unknown
                           object
14:26:34.49104     23652.1       }! ImqObject::open rc=MQRC_UNKNOWN_OBJECT_NAME
```

Figure 22-8 WebSphere MQ trace data from a non-AIX system

From this example we can see that no Object Name is included in the MQOD (Object Descriptor), so MQ does not know what needs to be opened.

22.7 WebSphere Business Integration Message Broker

The following information is required to perform WebSphere Business Integration Message Broker problem diagnosis on distributed platforms:

1. The failing *message flow*
2. The input message to drive message flow (and message sets if using MRM)

3. Database definition of user tables (DDL)
4. Clear description of the data being inserted

Recreate the error condition and provide:

1. WBI service trace
2. A service-level trace of the broker admin agent during a failed deploy that causes the BIPxxxxE message
3. A service-level trace of the brokers' *default* execution group that is experiencing the problem
4. Your environment details, meaning the operating system and product release level and maintenance details
5. The abend file that is generated during tracing

These two traces should be obtained from the time the broker and execution were started, and then extracted after the failure has occurred.

On the Windows platform, do not ignore the importance of the System log, Application log and Security log, which can be accessed via the Administrative Tools Event Viewer option. These files are located in the C:\WINNT\System32\Config directory and are identified with the .evt suffix. The files are AppEvent, SysEvent and SecEvent. Review these for any related BIPxxxxx messages.

22.7.1 WBI Message Broker command-level tracing

If you want to trace the command executables themselves, for example, when creating a configuration manager, you must set the environment variables MQSI_UTILITY_TRACE and MQSI_UTILITY_TRACESIZE before you initiate trace.

Be sure to reset these variables when the command you are tracing has completed. If you do not do so, all subsequent commands are also traced, and performance will therefore be degraded.

For example:

1. Set the following environment variable:

```
MQSI_UTILITY_TRACE=debug
```

2. Run the following command until you get the failure:

```
mqsicreateconfigmgr
```

3. Issue the following command to create an XML file from the trace data:

```
mqsireadlog configmgr -t -b mqsicreateconfigmgr -f -o config.xml
```

4. Issue the following command to format the previously created XML file:

```
mqsiformatlog -i config.xml -o config.txt
```

5. Set the following environment variable to reset the trace flag:

```
MQSI_UTILITY_TRACE=none
```

22.7.2 Tracing the WBI Message Broker and execution group at startup

Use the following procedure to trace the WebSphere Business Integrator Message Broker is as follows:

1. Stop the Message Broker by issuing the following command if you want to trace activity from a Broker startup:

   ```
   mqsistop <brokerName>
   ```

2. Enable the Broker agent trace as follows:

   ```
   mqsiservice <brokername> -r Trace=debug
   ```

3. Enable execution group tracing as follows:

   ```
   mqsiservice <brokerName> -r executionGroupTraceLevel=debug
   ```

4. Start the Broker

   ```
   mqsistart <brokerName>
   ```

5. Recreate the problem.
6. Stop the Broker.
7. Read the trace log files and output this data to XML files as follows:

   ```
   mqsireadlog brokerName -t -b agent   -f -o agent.xml
   mqsireadlog brokerName -t -b service -f -o service.xml
   mqsireadlog brokerName -t -b <egroup> -f -o <egroup>.xml
   ```

8. Format the XML files created in the previous step as follows:

   ```
   mqsiformatlog -i agent.xml -o agent.txt
   mqsiformatlog -i service.xml -o service.txt
   mqsiformatlog -i <egroup>.xml -o <egroup.txt
   ```

9. Turn off tracing by issuing:

   ```
   mqsiservice <brokerName> -r Trace=""
   mqsiservice <brokerName> -r executionGroupOverrideLevel=""
   ```

22.7.3 Tracing the WBI Message Broker execution group

To dynamically start a WBIMB execution group mqsi user trace, use the following procedure:

1. Issue the following command to enable the execution group trace:

   ```
   mqsichangetrace <brokerName> -u -e <egroup> -l debug -r
   ```

2. Wait until the DataFlowEngine process has failed; this should generate trace files in the log directory.
3. Stop the execution group trace by issuing:

   ```
   mqsichangetrace <brokerName> -u -e <egroup> -l none
   ```

4. Read the created trace files and create the XML file for formatting as follows:

   ```
   mqsireadlog <brokerName> -t -e <egroup> -f -o <egroup>.xml
   ```

5. Format the generated XML file as follows:

   ```
   mqsiformatlog -i <egroup>.xml -o <egroup>.fmt
   ```

It is advisable to clear service trace files (using the **mqsichangetrace –r** option) and ensure that it does not wrap by assigning a large enough size (use the **-c** option). Additionaly, clear the ODBC trace file prior to running failure test and if possible, recycle the broker (using

mqsistop/mqsistart) since the ODBC trace will then show the database connection being re-established.

22.7.4 WBI Message Broker Configuration Manager tracing

If you need to trace the Configuration Manager startup process you will need to modify the ConfigMgr Windows registry entry using the following steps:

1. End the configuration manager.
2. Run **regedit** and go to:

   HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphereMQIntegrator\2\ConfigMgr
3. Right-click **configmgr**, add a new String Value, and rename it to `Trace`. Modify the Data value to `debug`.
4. Start the Configuration Manager by issuing **mqsistart**.
5. Recreate the problem.
6. Stop the Configuration Manager.
7. Return to the registry and delete the Trace entry created earlier.

To trace the Configuration Manager during “normal” operation perform the following:

1. Issue the following command to set the trace flag on:

   ```
   mqsichangetrace ConfigMgr -u -e -l debug -r
   ```
2. Recreate the problem.
3. Make sure the configuration manager trace is stopped by issuing:

   ```
   mqsichangetrace ConfigMgr -u -e -l none
   ```
4. Convert the trace data to XML files by issuing the following commands:

   ```
   mqsireadlog configmgr -t -b agent   -f -o agent.xml
   mqsireadlog configmgr -t -b service -f -o service.xml
   ```
5. Format the generated XML files by using the **mqsiformatlog** command.

   ```
   mqsiformatlog -i agent.xml -o agent.txt
   mqsiformatlog -i service.xml -o service.txt
   ```

22.8 Lightweight Directory Access Protocol (LDAP)

Lightweight Directory Access Protocol (LDAP) is a user registry in which authentication is performed using an LDAP binding.

WebSphere Application Server security provides and supports implementation of most major LDAP directory servers, which can act as the repository for user and group information. These LDAP servers are called by the product processes (servers) for authenticating a user and other security-related tasks (for example, getting user or group information).

If the error logs do not provide enough information to resolve a problem, you can run the IBM Directory Server in a special debug mode that generates very detailed information. The server executable **ibmslapd** must be run from a command prompt to enable debug output. The syntax is as follows:

```
ldtrc on
ibmslapd -h bitmask
```

The specified bitmask value determines which categories of debug output are generated. Figure 22-9 shows the LDAP debug categories.

```
Hex        Decimal Value Description
0x0001           1 LDAP_DEBUG_TRACE Entry and exit from routines
0x0002           2 LDAP_DEBUG_PACKETS Packet activity
0x0004           4 LDAP_DEBUG_ARGS Data arguments from requests
0x0008           8 LDAP_DEBUG_CONNS Connection activity
0x0010          16 LDAP_DEBUG_BER Encoding and decoding of data
0x0020          32 LDAP_DEBUG_FILTER Search filters
0x0040          64 LDAP_DEBUG_MESSAGE Messaging subsystem activities and events
0x0080         128 LDAP_DEBUG_ACL Access Control List activities
0x0100         256 LDAP_DEBUG_STATS Operational statistics
0x0200         512 LDAP_DEBUG_THREAD Threading statistics
0x0400        1024 LDAP_DEBUG_REPL Replication statistics
0x0800        2048 LDAP_DEBUG_PARSE Parsing activities
0x1000        4096 LDAP_DEBUG_PERFORMANCERelational backend performance
statistics
0x1000        8192 LDAP_DEBUG_RDBM Relational backend activities (RDBM)
0x4000       16384 LDAP_DEBUG_REFERRAL Referral activities
0x8000       32768 LDAP_DEBUG_ERROR Error conditions
0xffff       65535 ALL
0x7fffffff 2147483647 LDAP_DEBUG_ANY All levels of debug
```

Figure 22-9 LDAP debug categories

22.9 IBM DB2 UDB on UNIX and Windows

The key files required to assist with problem diagnosis of DB2-related problems are:

- db2diag.log file
- Trap files
- Dump files
- Messages files

These files are generated or updated when different events or problems occur.

22.9.1 db2diag.log file

The db2diag.log contains most of the key information used for DB2 problem diagnosis. This file is located in the DB2 diagnostic directory, defined by the DIAGPATH variable in the Database Manager Configuration.

By default the directory is defined as follows:

- UNIX: $HOME/sqllib/db2dump - where $HOME is the DB2 instance owner's home directory
- Windows or OS/2: INSTALL PATH\SQLLIB\<DB2INSTANCE> - where INSTALL PATH is the directory where DB2 is installed

The Database Manager Configuration also controls how much information is logged to the db2diag.log through the use of the diagnostic level, or DIAGLEVEL variable. The

DIAGLEVEL can be set from 0 to 4, but this setting by default is 3, which is usually sufficient for most problems.

Trap files

Whenever a DB2 process receives a signal or exception (raised by the operating system as a result of a system event), a trap file is generated in the DB2 diagnostic directory.

The files are created using the following naming convention:

- UNIX: t*pppppp.nnn* - where *pppppp* is the process ID (PID) and *nnn* is the node where the trap occurred
- Intel®: DB*ppptttt*.TRP - where *ppp* is the process ID (PID) and *ttt* is the thread ID (TID)

Dump files

When DB2 determines that a serious problem has been detected (often related to the internal function of DB2), a dump will often be taken and the files will be located in the DIAGPATH directory. The filenames will either be pppppp.nnn or lpppppp.nnn for UNIX, or ppptttt.nnn or lppptttt.nnn for Windows.

Messages files

Some DB2 utilities like BIND, LOAD, EXPORT, and IMPORT provide an option to dump out messages files to a user-defined location. These files contain useful information to report the progress, success, or failure of the utility that was run and can assist with problem determination.

db2support utility

The db2support utility is designed to automatically collect all DB2 and system diagnostic information available (including information described in previous sections). It has an optional interactive *Question and Answer* session available to help collect information for problems that you may want additional assistance investigating.

The db2support utility was made available in DB2 V7 (FP4) and is invoked as follows:

```
db2support <output path> -d <database name> -c
```

The output is collected and stored in db2support.zip.

22.9.2 JAVA Database Connector tracing

JDBC™ is the connector between the WebSphere for z/OS and the DB2 UDB for z/OS. This connector allows communication between the two products. Requests to DB2 go through this connector, and this trace details what happens during this communication.

The JDBC trace is useful for diagnosing problems in the DB2 Structured Query Language for Java and Java Database Connector (SQLJ/JDBC). The output will go to an HFS file specified in the JDBC properties file.

JDBC trace information shows Java methods, database names, plannames, usernames, or connection pools.

22.9.3 Running a JDBC trace

Perform the following steps to obtain the JDBC trace:

1. Set up the environmental variable parameter in the current.env file:

   ```
   DB2SQLJPROPERTIES=/usr/lpp/DB2/DB2710/classes/wsccb_DB2sqljjdbc.properties
   ```

2. In this properties file, called wsccb_DB2sqljjdbc.properties, set up the variable DB2SQLJ_TRACE_FILENAME to enable the SQLJ/JDBC trace and specify the name of the file to which the trace is written:

   ```
   DB2SQLJ_TRACE_FILENAME=/tmp/IVP2_jdbctrace
   ```

   Note: You can specify the path and file name that you want; we used /tmp/IVP2_jdbctrace as an example.

3. The JDBC trace produces two HFS files:

   /tmp/IVP2_jdbctrace
   This file will be in binary format. You need to format it using the **DB2sqljtrace** command described in the next step.

   /tmp/IVP2_jdbctrace.JTRACE
   This file contains readable text.

4. To format the binary trace data, use the following **DB2sqljtrace** command in the USS environment (OMVS or telnet):

   ```
   DB2sqljtrace fmt|flw TRACE_FILENAME > OUTPUT_FILENAME
   ```

   You can use the fmt subcommand or the flw subcommand, as follows:

   – **fmt** - specifies that the output trace is to contain a record of each time a function is entered or exited before the failure occurs.

   – **flw** - specifies that the output trace is to contain the function flow before the failure occurs.

   OUTPUT_FILENAME is the name of the file where you want the new formatted trace.

   Note: Be sure the PATH and LIBPATH environmental variables are defined with the JDBC path and libraries to run this command correctly.

   You can change them with the following commands:

   ```
   export PATH=$PATH:/usr/lpp/DB2/DB2710/bin
   export LIBPATH=$LIBPATH:/usr/lpp/DB2/DB2710/lib
   ```

   You can verify that they are correct with the following commands:

   ```
   echo $PATH
   echo $LIBPATH
   ```

22.10 WebSphere TXSeries (CICS for UNIX and Windows)

The key pieces of diagnostic information required for diagnosing TXSeries/CICS problems include not only CICS specific data, but also data related to the Encina and DCE components of TXSeries.

22.11 The SYMREC file

When the CICS region writes a symptom record, the record is appended to the file:

- /var/cics_regions/regionName/symrecs.nnnnnn (on CICS for Open Systems)
- c:\var\cics_regions\regionName\symrecs.nnnnnn (on CICS for Windows)

This file does not exist until a symptom record is written. CICS appends to the symrecs.nnnnnn file and never truncates it.

CSMT is a transient data queue that contains messages about transactions. Messages written to CSMT include the date, time, region name, and principal facility, when there is one.

CSMT is in:

- /var/cics_regions/region/data (on Open Systems)
- c:\var\cics_regions\region\data (on Windows)

The console.nnnnnn is located in:

- /var/cics_regions/regionName/data (on Open Systems)
- c:\var\cics_regions\regionName\data (on Windows)

Every time a new CICS region starts, the number of the console.nnnnnn file is incremented by one.

Note: When a CICS region is cold-started, console.nnnnnn and CSMT are recreated. Any information previously stored in these files is lost. Make a copy of console.nnnnnn and CSMT before you restart a region after an error.

On the Windows platform, do not ignore the importance of the System log, Application log and Security log, which can be accessed via the Administrative Tools Event Viewer option. These files are located in the C:\WINNT\System32\Config directory and are identified with the .evt suffix. The files are AppEvent, SysEvent and SecEvent.

22.12 Encina trace messages

Encina trace messages can be located in the following subdirectories:

- /var/cics_servers/ (on Open Systems)
- c:\var \cics_servers\ (on Windows)

These messages are generated by the server for fatal, nonfatal, and audit messages.

22.13 DCE diagnostic data

DCE messages are written to:

- /opt/dcelocal/var/svc (on Open Systems)
- c:\opt\cics\dcelocal\var\svc (on Windows NT)

The files that store this data are: error.log, fatal.log and warning.log. DCE Endpoint mapper messages are written to /opt/dcelocal/var/dced/dced.log.

DCE Security messages are written to either /opt/dcelocal/var/security/secd.log or /opt/dcelocal/var/security/adm/secd/secd.log.

Database-related error data can be found in DB2diag.log

The message output is classified into five categories: fatal, error, warning, notice, and verbose notice. DCE itself is capable of logging messages to files that are stored by default in /opt/dcelocal/var/svc, and whose disposition is most easily controlled via the /opt/dcelocal/var/svc/routing file.

By default, only fatal, error, and warning messages are actually kept; messages of type notice and verbose notice are discarded. The default messages are stored in files named fatal.log, error.log, and warning.log in the /opt/dcelocal/var/svc directory. You can cause DCE to retain notice and verbose notice messages too, by editing the /opt/dcelocal/var/svc/routing and adding lines like these:

```
NOTICE:FILE:/opt/dcelocal/var/svc/notice.log
NOTICE_VERBOSE:FILE:/opt/dcelocal/var/svc/verbose.log
```

The DCE endpoint mapper process, dced, keeps a log file in /opt/dcelocal/var/dced/dced.log. This logfile is created anew each time dced is started; old messages from the previous instance of dced are not saved.

Similarly, on security server machines, secd keeps a log file in either /opt/dcelocal/var/security/secd.log or in /opt/dcelocal/var/security/adm/secd/secd.log (depending on the platform). This logfile is also created anew whenever secd is started, and old messages are not retained.

You will almost always be asked for output from the show_conf script when you report a problem. The show_conf script gathers all sorts of information about a machine that includes: OS version and configuration data, DCE/DFS version and PTF information, logfiles, and other related data. You will usually be requested to run **show_conf** on your DCE/DFS server machines and perhaps on a few selected client machines. For example, if you have a problem that occurs on some clients but not on others, you may be asked to run show_conf on one or two of the *bad* clients and also on one or two of the *good* clients.

The output of show_conf will be 20 to 40 pages of text for each machine.

The show_conf utility (along with other DCE debug tools) can be downloaded from:

```
http://www.ibm.com/software/network/dce/support/index.debugtools.html
```

22.14 DCE/DFS core files

If a process related to DCE/DFS drops core, you will need to send the IBM Support Center the corefile. But, the core alone is not enough, since it will depend on shared libraries on your system that may differ from the libraries on IBM systems. The solution is to use an IBM debug tool to package up the core and all the other binaries that it depends on. If the core occurred on Solaris, then you would use **grab_core**; if on AIX, you would use **senddata**.

On AIX, the default limit on core file size may be relatively small, and that may cause the core to be truncated. A truncated core will probably be useless for debugging; if your AIX limits cause your core to be truncated, you will have to raise the limit and wait for another core before the support staff can attempt to diagnose your problem. Also, note that senddata requires Perl on the system where it is run.

You may not want to send the entire core when first reporting a problem. The tools described in the next section (showProcInfo and dumpthreads) can be run against a core file on your system, and will yield a set of thread stacks from the core. When first reporting a core, you may want to just run one of these tools and send IBM the output; but be sure to save the core in case it turns out that support staff need the whole thing.

22.15 DCE/DFS process hangs or loops

If a process is hanging and appears to be unresponsive, or if it is spinning (looping) and consuming large amounts of CPU time, then you need to see what the process is doing to enable you to figure out what's wrong. There are a couple of tools that allow you to see stack traces for each thread in the process. By running one of these tools a few times (waiting a minute or so between runs), you can see which threads are moving and which are stuck. You can use showProcInfo on either Solaris or AIX; or you can use dumpthreads on AIX.

Both of these tools require dbx on the system where they are run, and showProcInfo also requires Perl.

If you don't have the prerequisite tools (dbx, Perl, or both), then you could force the process to drop core, using **`gcore`** on Solaris (which does not kill the process), or **`kill -6`** on Solaris or AIX (but this will kill the process). You would then have to package the cores using either grab_core or sendatat.

Another alternative if you want a set of stack traces from a Solaris process, but you can't run showProcInfo because you lack dbx and/or Perl on the system, is to use /usr/proc/bin/pstack.

22.16 TXSeries CICS dump format utility (cicsdfmt)

The **`cicsdfmt`** utility should be used to format dumps. CICS names the dump file as aaaannnn.dmpmm, where aaaa indicates why the dump was taken. For example:

- ASRA as a result of an ASRA abnormal termination
- ASRB as a result of an ASRB abnormal termination
- SYSA as a result of a SYSA abnormal termination
- SHUT from a shutdown request
- SNAP from a CEMT PERFORM SNAP DUMP request
- A four letter dumpcode
- From an EXEC CICS DUMP command
- A four letter abnormal termination code

From an EXEC CICS ABEND command or from a transaction abnormal termination initiated by CICS.

The other parts of the name have the following meanings:

nnnn The dump sequence number, which CICS increments each time a dump is performed. CICS saves this number between runs. CICS retrieves this number when it autostarts the region. When CICS performs a region shutdown, it saves the current dump sequence number for the next autostart of the region.

dmp The dump ending string to identify the file as a dump file.

mm A number to indicate if the dump data from one invocation to dump was split over a number of files or not. If the dump data is in one file, the file is named aaaannnn.dmp01. If the dump data is spread over two files, the dump data will be in the files named aaaannnn.dmp01 and aaaannnn.dmp02, which are usually in different dump directories.

22.16.1 Dump directories

CICS uses a number of directories to write the dump. These directories are subdirectories of the dump directory.

On the DumpName attribute of the Region Definitions (RD), specify the name of the directory (containing the subdirectories) to which CICS dumps are written.

On the CoreDumpName attribute of the Region Definitions (RD), specify the name of a subdirectory of the DumpName directory. CICS uses this subdirectory for a core dump in the event of a nonrecoverable CICS abnormal termination.

22.17 TXSeries CICS trace format utility (cicstfmt)

The **`cicstfmt`** utility should be used to format trace files.

The **`cicsservice`** utility, found in the cics/utils directory, can be used to package up the diagnostic data for transmission, but if you have failed to cleanup your obsolete dumps and logs, this utility will package up a large amount of irrelevant data that has nothing to do with the current problem. It is probably best avoided if you do not manage your dump and log files.

22.18 WebSphere TXSeries tracing

Tracing in WebSphere TXSeries can require concurrent traces to be run for some, or all of the key components (CICS, Encina, and DCE).

In WebSphere TXSeries the CICS component basically provides a CICS API interface. The transaction coordination and management process runs as a global task under Encina. Encina is the transaction manager, the two-phase commit coordinator, and controls access to the Structured File Server (SFS). Even if DB2 is used as the file system, Encina will be the two-phase commit coordinator.

Most installations that run WebSPhere TXSeries CICS use Remote Procedure Calls (RPC). This does not use DCE for authorization. DCE in this case is merely the portal through which connections are passed via RPC to CICS.

On the other hand, sites that use Encina for transaction processing tend to use full DCE functionality.

We recommend, in the early stages of problem diagnosis, that you capture a full CICS trace, and if the problem also indicates a problem with the SFS, then Encina tracing should also be active. Should more granular trace data be required, then you will be advised of the required options.

22.19 TXSeries CICS auxiliary trace facility

Trace in WebSphere TXSeries is written per thread per process, so if the cicsas process has 26 threads then 26 files are recreated.

The suggested cicstrace environment variable settings are:

```
CICSTRACE= -A on -d /tmp -B off -M on -S on -t all=5
```

The variables are:

- A = TraceFlagAux
- B = TraceFlag Buffer
- M = TraceMasterFlag
- S = TraceSystemFlag
- t = TraceSystemSpec.
- d = TraceDirectorySystem

The blank before the -A *must* be there.

The CICSTRACE environment variable can be used to override the values from the RD stanza in the master trace area for any of the trace-related attributes.

This variable can be set in the region environment file, as follows:

```
/var/cics_regions/regionName/environment
```

It is read when the region starts

Alternatively, it can be set on the command line, where it is read dynamically.

22.19.1 Starting TXSeries CICS system tracing

To start collecting system trace from a running region, issue the following commands:

```
CECI TRACE ON
CECI TRACE SYSTEM ON
CEMT SET AUXTRACE ON
```

22.19.2 Stopping TXSeries CICS system tracing

To stop collecting system trace from a running region, issue the following commands:

```
CEMT SET AUXTRACE OFF
CECI TRACE SYSTEM OFF
CECI TRACE OFF
```

22.19.3 TXSeries CICS trace files

The location of all system trace files, including those created by dynamic trace commands, is determined by the value of the TraceDirectorySystem attribute.

System trace records are stored directly to files, called auxiliary trace files, when the TraceFlagAux RD attribute is switched on. The location of all system trace files, including auxiliary trace files, is determined by the value of the TraceDirectorySystem RD attribute. Auxiliary trace files are named according to the following pattern:

regionName.processName.nprocessNumber.pprocessID.tthreadID.format

Under this naming convention, processName is the name of the CICS process from which the trace is generated; the processNumber is the identifier assigned by CICS, not the operating system; processID is the operating-system identifier. The threadID is omitted for single-threaded processes like standalone programs; in multi-threaded processes, trace from each thread goes into separate files. The format field indicates if the file is formatted or not: the value is either cicsfmt for a formatted file or cicstrc for an unformatted file.

The unformatted files need to be formatted by using the **cicstfmt** utility before they can be read.

22.20 Encina tracing for CICS application server processes

The following procedures will allow you to find the server name used by the cicsas so that tkadmin tracing can be turned on for a cicsas process. This is an alternative to issuing the trace at startup in the /var/cics_regions/<region>/environment file.

This is a more dynamic form of tracing since you will start tracing a specific process for a specific server. It can be difficult to determine the server name that the cicsas process is using, and which is required to set the correct trace value.

To find out the server name in relation to a specific CICS Application Server process, use the following procedure:

1. List the PIDs of the cicsas processes:

   ```
   $ps -ef | grep cicsas
   ```

2. Keep track of the PIDs listed. You will need them later.
3. You need to determine the encina server_id of these processes so that you can trace them. The best way we know of to do this is to issue a log format command so the cicsas server names are displayed. For example:

   ```
   $cicslogfmt -r <region_name> -a
   ```

   You will see a long listing that will include the cicsas server names, for example:

   ```
   "ncadg_ip_udp:9.38.201.19[36441]"
   ```

   You will see many names in the log format output - not all of them cicsas names. To verify you have the right ones, pick a likely server name, such as CICS_AS_101_KMPREG1, and query its pid, like this:

   ```
   $tkadmin query process -server 'ncadg_ip_udp:9.38.201.19[36441]'
   ```

   The output will be the PID of the cicsas process. Once you have matched all of the PIDs from step 2, you can start the traces.

4) To set the trace spec, do the following:

```
$tkadmin trace spec -server 'ncadg_ip_udp:9.38.201.19[36441]' tmxa=all
```

5) To verify that it is set correctly, do the following:

```
$tkadmin list trace -server 'ncadg_ip_udp:9.38.201.19[36441]'
```

You will see tmxa set to "trace_all" in the output.

6) Redirect the trace to some file system:

```
$tkadmin redirect trace -s 'ncadg_ip_udp:9.38.201.19[36441]' trace -dest
/tmp/somefile.tmp
```

You will now be sending all XA events to /tmp/somefile.tmp.

7) Once you have completed tracing, turn tracing off using the following procedure.

```
$tkadmin redirect trace -s 'ncadg_ip_udp:9.38.201.19[36441]' trace
$tkadmin trace specification -s 'ncadg_ip_udp:9.38.201.19[36441]' all=default
```

This command will reset everything back to the default values.

22.21 Writing trace data to in-storage buffers

System trace records are stored in the trace buffer of each process when the TraceFlagBuffer is switched on. The trace buffer is a ring; trace records are written sequentially into the buffer, and when the end of the buffer is reached, storage continues at the beginning of the buffer, overwriting older records. The size of the ring buffer is determined by the value of the TraceMaxSizeBuffer attribute. The default value is 131,072 bytes. The attribute can be set to any positive integer, but changes do not affect processes already running.

The ring buffer is the default destination for system trace records. Storing trace records in the ring buffer puts the least load on the process being traced, making this the best way to trace production systems.

Records in the ring buffer must be retrieved before they can be read. This is achieved by dumping the ring buffer to a file. CICS processes dump their ring buffers automatically depending on the following environmental variable settings:

```
CICSTRACE_DUMP_ON_ABNORMAL_EXIT
CICSTRACE_DUMP_ON_EXIT
CICSTRACE_DUMP_ON_SYMREC
CICSTRACE_DUMP_ON_ABEND
CICSTRACE_DUMP_ON_MSN
```

The location of all system trace files, including those dumped from the buffer, is determined by the value of the TraceDirectorySystem environment variable.

For Encina the **tkadmin dump ringbuffer** command dumps trace output contained in a main memory ring buffer to a file. The **tkadmin set ringbuffer size** command changes the size of a ring buffer.

Some helpful Encina diagnostic utilities can be downloaded from:

http://www.ibm.com/software/webservers/appserv/txseries/support/encina_faq.html

22.22 CICS universal client

There are two types of messages in the client daemon: messages displayed to the user and errors written to the Client daemon error log and trace file.

CICS Transaction Gateway for z/OS Administration, SC34-5528 explains all of these messages.

22.22.1 Error log messages

Any errors on the client workstation that are not caused by incorrect use of the API are written to the Client daemon error log.

The error log (which has a default name of CICSCLI.LOG) is an ASCII text file that you can browse using a standard text editor.

Help information for all messages is provided in two ASCII text files in the <install_path>\bin subdirectory. You can view these using any standard text editor.

- CCLMSG.HLP provides help for the end user messages, in the language you chose at installation time.
- CCLLOG.HLP provides help for the trace and log messages. This is provided in English only.

Errors resulting from incorrect use of the APIs simply result in error return codes to the application. It is the responsibility of the application to notify the end user of the error and provide guidance on corrective actions.

22.22.2 Pop-up messages

Errors generated from within the Client daemon are displayed in a pop-up window. You must click OK in the pop-up window before the Client daemon can continue processing. There may be times when you do not want messages to appear in pop-up windows, for example, if you leave the Client daemon running unattended overnight. To disable the display of pop-up messages, enter the following command:

```
CICSCLI  -n
```

When the display of pop-up messages is disabled, the messages are still written to the Client daemon error log. To re-enable the display of pop-up messages, enter the following command.

```
CICSCLI  -e
```

You can specify the **-n** parameter together with the **-s** parameter. The display of pop-up messages is enabled by default.

22.22.3 CICS universal client tracing

The client daemon tracing is a very useful problem determination tool for communication problems. You can use the trace functions to collect detailed information on the execution of a certain function or transaction. A trace can show you how the execution of a particular activity is affected by, for example, the execution of other tasks in a CICS system. Each trace entry has a time stamp, which provides information on the time taken to perform certain activities.

You can specify which components of the client daemon you want to trace. You control this with the **CICSCLI -m** command, (see the CICSCLI command reference for details), or by specifying a list of components using the configuration tool.

The output from the trace function is a binary trace file called, by default, CICSCLI.BIN in the <install_path>\bin subdirectory. You can specify a different name for this file, using the configuration tool. However, you cannot change the .BIN extension. Using the Maximum Client wrap size configuration setting, you can specify that the binary trace file should wrap into a second trace file, and you can also specify the maximum size of these files.

To read the trace, run the CICSFTRC utility to convert the binary file or files into a text file. This text file is called CICSCLI.TRC by default. The files exist as follows:

- The default trace files are:

  CICSCLI.BIN

- The binary trace file produced by running the Client daemon trace is:

  CICSCLI.WRP

- The second binary trace file if wrapping of client trace is enabled is:

  CICSCLI.TRC

- The name of the text trace file produced when the binary trace file is converted to a text file using the CICSFTRC utility is:

  CICSCLI.BAK

  This is the backup file of the binary trace file. A backup file is produced from any existing .BIN file when you turn tracing on.

22.23 Starting and stopping client daemon tracing

To start client daemon tracing, enter the **CICSCLI** command with the **-d** option, for example:

```
CICSCLI -d=nnn
```

Here, *nnn* is optional, and is the maximum size in bytes of the data areas to be traced. The default value is 512.

We recommend that you set at least -d=1000 to ensure that all relevant data is included in the trace before sending it to your support organization.

As a general rule the following is recommended for CICS client tracing:

```
CICSCLI -d=9999 -m=all
```

If you need to trace the Client daemon immediately from startup, you can specify the **-s** and **-d** parameters together in the same CICSCLI command. For example, the following command starts the connection to a CICS server with the name CICSTCP, enables the trace function, and sets the maximum data area size to be traced to 128 bytes:

```
CICSCLI  -s=CICSTCP -d=128
```

You can specify which components are to be traced when you start tracing. See the CICSCLI command reference for details.

To stop Client daemon tracing, enter the CICSCLI command with the **-o** option, for example:

```
CICSCLI -o
```

The trace is also automatically stopped if you stop the Client daemon by using the **CICSCLI -x** command.

Choosing which components to trace is specified using the following options:

ALL This option traces everything. It is the preferred option; use it if performance allows, and consider using the binary formatting tool to filter information.

API.1 and **API.2** These trace the boundary between the user application or Java classes and the Client daemon. Switching on API.2 automatically switches on API.1 as well.

DRV.1 and **DRV.2** The protocol drivers trace the boundary between the Client daemon and the network protocol. Specify the DRV.1 and API components if you are not sure whether a problem is inside the Client daemon, and you want to minimize the impact on performance, for example when trying to determine a performance problem.

You can also specify these components to help determine which product is causing the system to lock up.

CCL This component traces the main Client daemon process. Specify the API and CCL components if you believe that the problem is within the Client daemon.

TRN The TRN component traces the internal interprocess transport between Client processes. Use it if entries in the Client log refer to functions such as FaarqGetMsg, FaarqPutMsg, or FaarqStart. TRN is the most verbose tracing component.

22.24 Wrapping the client daemon trace

You can control the size of the binary trace file by specifying that it wraps into a second trace file. You turn on wrapping of trace using the Maximum Client wrap size configuration setting, where you specify the maximum size of the wrapping trace (in kilobytes). If this value is 0 (the default), wrapping trace is not turned on.

When wrapping trace is turned on, two files (called CICSCLI.BIN and CICSCLI.WRP) are used. Each file can be up to half the size of the Maximum Client wrap size value.

22.25 Formatting the binary trace file (CICSFTRC)

You use the binary trace formatter utility **CICSFTRC** to convert the binary trace file CICSCLI.TRC to ASCII text. The utility has the following parameters:

-m=list Specifies that only trace points from the listed components are written to the text file. The components you can specify are the same as for CICSCLI -m. If -m is not specified, all trace points in the binary trace are written to the text file.

-w[=filename] Indicates that there are two binary trace files to format and then concatenate (that is, the binary files were created with a wrapping trace). If no filename is specified with the -w parameter, CICSFTRC assumes that the name of the second trace file is CICSCLI.WRP.

The following parameters can also be used:

-n Indents entry and exit points in the test trace file to make it more readable. By default, indentation is turned off.

-d Specifies detailed trace formatting. If you are using EPI calls, CICSTERM or CICSPRINT, an approximation of the screen layout will be included in the trace.

-i=filename Specifies the name of the input (binary) trace file, which is CICSCLI.BIN by default.

-o=filename Specifies the name of the output (text) trace file. If no -o parameter is specified, the name of the text trace file is assumed to be CICSCLI.TRC.

-f Overwrite any existing files.

-s Do summary trace formatting. Summary trace formatting is driven from a text file (CCLSUMTR.TXT) which is read in at initialization time. This defines the set of trace points for which you want summary tracing, and the type of trace point. As DetailFormat reaches each trace point, if it is one of the ones read in from this file, a line is generated in the summary file. Use as requested by your IBM support organization; see Program support.

22.25.1 Summary of API calls produced by the formatter

The formatter can produce a summary of API calls that the user program makes, and show the progress of the calls through the Client daemon. Specify the API.2 component to produce a summary trace. Figure 22-11 shows an example of the API summary trace taken with the API.2 and DRV options. (The layout of your trace may be different, depending on the contents of CCLSUMTR.TXT.)

```
[Process ,Thread  ] Time             (API Summary)            (CCLCLNT) Summary (Comms Summary)
=============================================================================================================
[000000bf,0000017c] 12:08:32.190  --->[7315] CCL3310 ECI Call type ECI_SYNC, UOW=0
[00000089,000000a4] 12:08:32.290 (-S->)[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=89
[000000bf,0000017c] 12:08:32.330     [7391] CCL1040   using slot = Slotp->Ctrl.ConvId = 6
[000000bf,0000017c] 12:08:32.350     [7099] CCL1037   Sync ECI call, so waiting for a reply...
[00000089,00000063] 12:08:32.400 (<-R-)[4418] CCL4412 TCP/IP (to ITSOTEST) receive data: Length=12
[00000089,0000018b] 12:08:32.511 <-R-[4418] CCL4412 TCP/IP (to ITSOTEST) receive data: Length=29
[00000089,000000a4] 12:08:32.521 -S->[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=94
[00000089,000000a4] 12:08:32.531 -S->[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=94
[00000089,000000a4] 12:08:32.541 -S->[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=94
[00000089,000000a4] 12:08:32.541 -S->[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=94
[00000089,000000a4] 12:08:32.551 -S->[4410] CCL4411 TCP/IP (to ITSOTEST) send data: Length=94
[00000089,00000168] 12:08:32.581 <-R-[4418] CCL4412 TCP/IP (to ITSOTEST) receive data: Length=12
[00000089,0000017e] 12:08:32.601 <-R-[4418] CCL4412 TCP/IP (to ITSOTEST) receive data: Length=31
[000000bf,0000018e] 12:08:32.621     [7364] CCL3350   Event Service Thread got a request REQUEST_TYPE_ECI_1
[000000bf,0000008a]  12:08:32.671 <---[7316] CCL3311 ECI Call type ECI_SYNC, UOW=0 got rc=ECI_NO_ERROR ({Time
                                   in API = 0.821 seconds})
```

Figure 22-10 API summary trace

Summary trace description

The *Time in API* field shows the amount of time that the client API call took to complete. This can help when investigating performance problems.

The API Summary column refers to client API code inside the user application process. It tracks when user requests enter and leave the client API code. ---> and <--- show the program entering and leaving the Client daemon API.

CCLCLNT is the background Client daemon process. You only get entries here if you specify the CCL tracepoint.

The comms summary tracks when client daemon calls enter and leave the network. -S-> shows a request being sent to the network; <-R- shows a reply being received.

22.25.2 Diagnosing application errors

If you just want to diagnose an application error, and are not interested in the client, specify only API.1 and API.2 tracepoints. The trace contains less information, and is easier to understand.

If a user application is making EPI calls, or using CICSTERM or CICSPRNT, the trace formatter puts an approximation of the screen into the trace. Figure 22-12 shows a 3270 screen capture from a formatted trace file, taken from the CECI transaction. It is an aid to problem determination, not a completely accurate representation of the screen.

The formatter lists the commands which built the screen, and shows an approximation of the screen.

.

```
Command = Erase/Write, so clearing main screen
 Command2 = Read Modified
 WCC = 0x32 ( Free Kbd,80 char)
 Set Buffer Address to (1,2)
 Insert Cursor @ (1,2)
 Set Buffer Address to (1,1)
 Start Field Extended (Unprotected,Alphanumeric,Display,not-pen-detectable,Foreground Colour Green)
 Data : ' '
 Insert Cursor @ (1,3)
 Set Buffer Address to (2,1)
  Data : 'User        '
  .....
  .....
  .....
  .....
 Set Buffer Address to (24,49)
 Start Field Extended (Autoskip (Prot+Num),Display,not-pen-detectable,Foreground Colour Turquoise)
 Data : '9'
 Set Buffer Address to (24,51)
 Start Field Extended (Unprotected,Alphanumeric,Intense,pen-detectable,Foreground Colour Red)
 Data : 'Messages    '
            1         2         3         4         5         6         7         8
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
  >+--------------------------------------------------------------------------------+
01| -                                                                              |
02| ´STATUS. . :´Enter one of the following:                                       |
03|     ´                                                                          |
04| ´ABend       EXtract     READPrev    WAit                                      |
05| ´ADdress     FEpi        READQ       WRITE                                     |
06| ´ALlocate    FOrmattime  RECeive     WRITEQ                                    |
07| ´ASKtime     FREE        RELease     Xctl                                      |
08| ´ASSign      FREEMain    RESetbr                                               |
09| ´Bif         Getmain     RETRieve                                              |
10| ´CAncel      Handle      RETUrn                                                |
11| ´CHange      IGnore      REWrite                                               |
12| ´CONNect     INquire     SENd                                                  |
13| ´CONVerse    ISsue       SET                                                   |
14| ´DELAy       LInk        SIGNOFf                                               |
15| ´DELETE      LOad        SIGNON                                                |
16| ´DELETEQ     PErform     START                                                 |
17| ´DEQ         POP         STARTBr                                               |
18| ´DUmp        POSt        SUspend                                               |
19| ´ENDbr       PUsh        SYncpoint                                             |
20| ´ENQ         READ        Unlock                                                |
21| ´ENTer       READNext    Verify                                                |
22|     ´                                                                          |
23| ´PF´1-Help       ´2-HEX         ´3-End         ´4-EIB         ´5-Variables    |
24|    ´6-User                                     ´9-Messages                     |
  +--------------------------------------------------------------------------------+
  | 1BþC000                   ITSOTEST                                             |
  +--------------------------------------------------------------------------------+
    12345678901234567890123456789012345678901234567890123456789012345678901234567890
            1         2         3         4         5         6         7         8
```

Figure 22-11 Formatted trace file screen capture

22.26 Client daemon trace analysis

The client daemon trace file records detailed information on all actions taken during the execution of a particular activity. You can use this information in your problem determination

activities, and to help understand how the Client daemon performs a particular function, for example, establishing a connection to a CICS server.

If you cannot interpret the trace yourself, you should contact your IBM support organization and forward the unformatted binary trace file.

Some sample traces are shown in sample client daemon trace.

Format of trace entries

The entries in the client daemon trace file have the following format:

```
time [process id,thread id] [number] component trace message data
```

Where:

time = The time the entry was written, to millisecond accuracy.

[process id, thread id] = Process id is a unique number that the operating system uses to identify a process. Thread id is a unique number that the operating system uses to identify a thread within a particular process.

[number] = A number to aid your support organization in the diagnosis of serious problems.

[component] = The component of the product to which this entry applies.

trace message = The trace message number and text. These trace messages are explained in *CICS Transaction Gateway for z/OS Administration,* SC34-5528.

data = Some trace entries include a dump of key data blocks in addition to the trace message.

22.26.1 Sample client daemon trace

Figure 22-12 on page 239 shows the trace information recorded during the successful connection of a Client daemon to a CICS server using the TCP/IP protocol.

The trace was generated using the commands:

```
CICSCLI -s=cicstcp -d
CICSCLI -o
```

Client daemon trace description

Following is a description of the trace:

CCL1042 Start of trace message. The trace file is overwritten each time a trace is started. You can delete the file when required. Check the date and time stamp to ensure that you are reading the correct trace.

CCL2048 Maximum trace data size is at the default size of 512 bytes. You can modify this size by specifying the size value in the start command for the client trace (see Starting and stopping Client daemon tracing).

CCL3251 The client sends a CCIN transaction to the server to install its connection definition on the server.

CCL3238 Reply to message CCL3238, includes the conversation ID for this conversation.

CCL3113 The client sends a CCIN transaction to the server with Appl ID set to * to install its application. The Appl ID is specified in the configuration file as Client=*. This

requests the server to dynamically generate an Appl ID that is unique within the CICS server system.

CCL3114 This is the response to message CCL3114 with the dynamically generated Appl ID.

CCL1043 End of trace message.

```
17:03:57.580 [0000080c,00000908] [1007] TRC:(CCL1042) *** CICS Client for AIX v5.0 Service
Level 00 - service trace started ***
17:03:57.590 [0000080c,00000908] [2183] CCL:(CCL2048) Maximum trace data size set to 512
17:03:57.600 [0000080c,00000908] [2114] CCL:CCL2142 GetNextTimeout timeout is 0 seconds
17:03:58.612 [0000080c,00000908] [2030] CCL:CCL2106 Comms Event : LINK-UP
17:03:58.622 [0000080c,00000908] [2019] DRV:CCL2055 Connection with server established
(linkID=1)
17:03:58.622 [0000080c,00000908] [2035] CCL:CCL2109 Send server TCS data
17:03:58.632 [0000080c,00000908] [3214] CCL:(CCL3251) Comms Allocate request (LinkId=1,
Tran='CCIN')
17:03:58.632 [0000080c,00000908] [3217] CCL:(CCL3238) Comms Allocate completed (LinkId=1,
ConvId=1, Rc=0)
17:03:58.632 [0000080c,00000908] [2127] CCL:CCL2143 CommsBegin - OK
17:03:58.632 [0000080c,00000908] [3100] CCL:(CCL3113) CCIN install request: ApplId='*      ',
Codepage=819
...
...
...
17:04:00.625 [0000080c,00000908] [3102] CCL:(CCL3114) CCIN install response:
ApplId='@0Z8AAAA', Codepage=8859-1, Rc=0
17:04:00.625 [0000080c,00000908] [3241] CCL:CCL3255 Comms Complete request (ConvId=1)
17:04:00.635 [0000080c,00000908] [3244] CCL:CCL3246 Comms Complete completed (ConvId=1, Rc=0)
17:04:00.635 [0000080c,00000908] [3218] CCL:CCL3252 Comms Deallocate request (ConvId=1)
17:04:00.645 [0000080c,00000908] [3221] CCL:CCL3239 Comms Deallocate completed (ConvId=1,
Reason=0, Rc=0)
17:04:00.645 [0000080c,00000908] [2042] CCL:CCL2114 Processed TCS Reply - Server connected
17:04:00.645 [0000080c,00000908] [2114] CCL:CCL2142 GetNextTimeout timeout is 0 seconds
17:04:00.655 [0000080c,00000908] [2114] CCL:CCL2142 GetNextTimeout timeout is 0 seconds
17:04:00.664 [0000080c,00000908] [1004] TRC:(CCL1043) *** Service trace ended ***
```

Figure 22-12 Trace showing successful connection request

Figure 22-13 on page 240 shows trace information recorded when we tried to connect to a CICS server over TCP/IP using an invalid port number. The port number specified in the CTG.INI file was not defined in the services file of the server. Hence, the connection could not be established.

```
16:16:41.562 [0000093c,000008ec] [1007] TRC:CCL1042 *** CICS Client for Windows v5.0 Service Level 00 -
service trace started ***
16:16:41.572 [0000093c,000008ec] [2183] CCL:CCL2048 Maximum trace data size set to 512
16:16:41.582 [0000093c,000008ec] [2114] CCL:CCL2142 GetNextTimeout timeout is 0 seconds
16:16:41.612 [0000093c,000008ec] [2114] CCL:CCL2142 GetNextTimeout timeout is -1 seconds
16:16:41.622 [0000093c,000008ec] [3207] CCL:CCL3429 Comms Open request (Server=CICSTCP, Driver=CCLIBMIP)
16:16:41.622 [0000093c,000008ec] [4408] DRV:CCL4413 (CCL4413) TCP/IP (to CICSTCP)
address=192.113.36.200, (port=1089), socket=3
16:16:41.622 [0000093c,000008ec] [3210] CCL:CCL3236 Comms Open completed (Server=CICSTCP, LinkId=1,Rc=0)
16:16:41.633 [0000093c,000008ec] [2114] CCL:CCL2142 GetNextTimeout timeout is 3660 seconds
16:16:41.633 [0000093c,000008ec] [1004] TRC:CCL1043 ***Service trace ended ***
```

Figure 22-13 Trace example showing invalid connection request

CCL4413 shows the port number used for this connection request.

You must check your definitions in the SIT on the server, the configuration file on the workstation, and the services file for the port number specified. You must provide a valid port number or use the default value.

22.27 CICS Transaction Gateway tracing

To start full tracing on the CTG at startup, issue the following command:

```
ctgstart -x -tfile=ctg.trc
```

This will create a trace file called ctg.trc as indicated by the **-tfile** parameter. The **-x** option indicates full tracing.

JNI tracing can also be enabled when you start the CICS Transaction Gateway by issuing the following command

```
ctgstart -j-Dgateway.T.setJNITFile=filename
```

Here, filename is the name of the file to which trace output is to be sent.

22.27.1 JNI tracing

JNI tracing can also be enabled using the environment variable CTG_JNI_TRACE and specifying the path and file where you want the information stored. For example:

```
CTG_JNI_TRACE=c:\temp\ctgjni.trc
```

Starting the CICS Transaction Gateway would activate CTG and JNI tracing, as follows:

```
ctgstart -x -tfile=ctg.trc -jDgateway.T.setJNITFile=filename
```

To turn on full tracing for the java components of CTG, the following lines could also be added to the application code. All output will go to stdout so you will need to pipe it to a file. Add the import to the MAIN of the program. This will load the debug methods.

```
import com.ibm.ctg.client.T
```

Also, within the main() method, insert the following lines:

```
// user code begin {__main()_1}
  T.setDebugOn(true);
 // user code end {__main()_1}
```

This will set the debug trace on. To set it off, use the same block of code substituting **(false)** for **(true)**.

To do exception-only tracing use the same code, but substitute,

```
T.setStackOn(true)
```

Additional data that is required is the CICSCLI.ERR file, the CICSCLI.LOG file, and also the CTG.ini file.

Related publications

The publications listed in this section are considered particularly suitable for a more detailed discussion of the topics covered in this redbook.

IBM Redbooks

For information on ordering these publications, see “How to get IBM Redbooks” on page 244. Note that some of the documents referenced here may be available in softcopy only.

Other publications

These publications are also relevant as further information sources:

- *CICS Transaction Gateway for z/OS Administration,* SC34-5528
- *CICS Transactions for z/OS Trace Entries*, SC34-6242
- *CICS Diagnosis Reference*, LY33-6102
- *CICS Messages and Codes,* GC34-6241
- *CICS Problem Determination Guide*, SC34-6239
- *CICS User's Handbook,* SC34-5986
- *CICSPlex SM Messages and Codes*, GC33-0790
- *CICSPlex SM Problem Determination*, GC34-6472
- *DB2 UDB for z/OS V8 Messages and Codes*, GC18-7422
- *IMS V9 Messages and Codes Vol. 1*, GC18-7827
- *IMS V9 Messages and Codes Vol. 2,* GC18-7828
- *LE Debugging Guide and Run-Time Messages,* SC33-6681
- *WebSphere Application Server for z/OS V5: Messages and Codes*, GA22-7915
- *WebSphere MQ for z/OS Messages and Codes V5.3.1,* GC34-6056
- *WebSphere MQ for z/OS Problem Determination Guide V5.3.1*, GC34-6054
- *z/OS Language Environment Run-time Messages*, SA22-7566
- *z/OS Language Environment Debugging Guide,* GA22-7560
- *z/OS CS: SNA Messages*, SC31-8790
- *z/OS CS: IP Diagnosis Guide,* GC31-8782-05
- *z/OS CS: IP Messages Volume 1 (EZA)*, SC31-8783
- *z/OS CS: IP Messages Volume 2 (EZB, EZD),* SC31-8784
- *z/OS CS: IP Messages Volume 3 (EZY)*, SC31-8785
- *z/OS CS: IP Messages Volume 4 (EZZ, SNM)*, SC31-8786
- *z/OS CS: IP and SNA Codes*, SC31-8791
- *z/OS Communications Server SNA Diagnosis Volume 1: Techniques and Procedures* LY43-0088

- *z/OS Communications Server SNA Diagnosis Volume 2: FFST Dumps and the VIT* LY43-0089
- *z/OS MVS IPCS Commands*, SA22-7594
- *z/OS MVS System Codes*, SA22-7626
- *z/OS MVS System Messages, Vol 1 (ABA-AOM)*, SA22-7631
- *z/OS MVS System Messages, Vol 2 (ARC-ASA),* SA22-7632
- *z/OS MVS System Messages, Vol 3 (ASB-BPX), SA22-7633*
- *z/OS MVS System Messages, Vol 4 (CBD-DMO),* SA22-7634
- *z/OS MVS System Messages, Vol 5 (EDG-GFS)*, SA22-7635
- *z/OS MVS System Messages, Vol 6 (GOS-IEA)*, SA22-7636
- *z/OS MVS System Messages, Vol 7 (IEB-IEE),* SA22-7637
- *z/OS MVS System Messages, Vol 8 (IEF-IGD),* SA22-7638
- *z/OS MVS System Messages, Vol 9 (IGF-IWM)*, SA22-7639
- *z/OS MVS System Messages, Vol 10 (IXC-IZP),* SA22-7640
- *z/OS UNIX System Services Messages and Codes,* SA22-7807

Online resources

These Web sites and URLs are also relevant as further information sources:

- CICS

  `http://www.ibm.com/software/htp/cics/library/`
- DB2 and IMS Information Center

  `http://publib.boulder.ibm.com/infocenter/dzichelp/index.jsp`
- z/OS Internet library

  `http://www-1.ibm.com/servers/eserver/zseries/zos/bkserv/`
- WebSphere Business Integration

  `http://www-306.ibm.com/software/integration/websphere/library/`
- WebSphere Application Server

  `http://www-306.ibm.com/software/webservers/appserv/was/support/`

How to get IBM Redbooks

You can search for, view, or download Redbooks, Redpapers, Hints and Tips, draft publications and Additional materials, as well as order hardcopy Redbooks or CD-ROMs, at this Web site:

ibm.com/redbooks

Help from IBM

IBM Support and downloads

ibm.com/support

IBM Global Services

ibm.com/services

Index

E

F

G

I

J

L

M

O

P

R

S

T

V

W

X

Z

z/OS Diagnostic Data Collection and Analysis

(0.5” spine)
0.475”<->0.873”
250 <-> 459 pages

z/OS Diagnostic Data Collection and Analysis

Redbooks

Diagnostic fundamentals and recognizing common problem types

Obtaining and analyzing dumps and traces

Tools for collecting detailed diagnostic data

This IBM Redbook describes problem diagnosis fundamentals and analysis methodologies for the z/OS system. It provides guidelines for the collection of relevant diagnostic data, tips for analyzing the data, and techniques to assist in identifying and resolving of Language Environment, CICS, CICSPlex/SM, MQSeries, VTAM, and DB2 problems. Also described are some diagnostic procedures that are not purely z/OS, but that are related to the various platforms (UNIX and Windows) where IBM software executes and interacts with z/OS in a Client/Server or distributed framework topology.

This document shows you how to:

- Adopt a systematic and thorough approach to dealing with problems
- Identify the different types of problems
- Determine where to look for diagnostic information and how to obtain it
- Interpret and analyze the diagnostic data collected
- Escalate problems to the IBM Support Center when necessary

Diagnostic data collection and analysis is a dynamic and complex process. This redbook shows you how to identify and document problems, collect and analyze pertinent diagnostic data and obtain help as needed, to speed you on your way to problem resolution.

INTERNATIONAL TECHNICAL SUPPORT ORGANIZATION

BUILDING TECHNICAL INFORMATION BASED ON PRACTICAL EXPERIENCE

IBM Redbooks are developed by the IBM International Technical Support Organization. Experts from IBM, Customers and Partners from around the world create timely technical information based on realistic scenarios. Specific recommendations are provided to help you implement IT solutions more effectively in your environment.

For more information:
ibm.com/redbooks

SG24-7110-00 ISBN 0738493996