CICS/ESA

Problem Determination Guide

Version 4 Release 1

SC33-1176-01

CICS/ESA

Problem Determination Guide

Version 4 Release 1

SC33-1176-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page v.

First edition (October 1994)

This edition applies to Version 4 Release 1 of the IBM licensed program Customer Information Control System/Enterprise Systems Architecture (CICS/ESA), program number 5655-018, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on this product.

This book is based on the Problem Determination Guide for CICS/ESA 3.3, SC33-0678-02. Changes from that edition are marked by vertical lines to the left of the changes.

The CICS/ESA 3.3 edition remains applicable and current for users of CICS/ESA 3.3.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page entitled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories Limited, Information Development,
Mail Point 095, Hursley Park, Winchester, Hampshire, England, SO21 2JN.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

The following paragraph does not apply in any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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 of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter 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 the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A..

Trademarks and service marks

The following terms, used in this publication, are trademarks or service marks of IBM Corporation in the United States or other countries:

CICS, SAA, MVS/XA, CICS/MVS, DB2, System/390, C/370, CICS/ESA, ESA/370, IBM, NetView, MVS/ESA, IMS/ESA, Enterprise Systems Architecture/370, RETAIN, VTAM

Programming interface information

This book is intended to help you diagnose problems in a CICS system. This book documents information that is Diagnosis, Modification or Tuning Information provided by CICS.

Warning: Do not use this Diagnosis, Modification or Tuning Information as a programming interface.

Preface

What this book is about

This book is about methods of determining the causes of problems in a system that uses CICS/ESA. It contains information about resolving CICS application and system problems, dealing with the IBM Support Center, and handling authorized program analysis reports (APARs).

In general, this book does not describe methods of problem determination for the CICS/ESA Front End Programming Interface. This information is included in the *CICS/ESA Front End Programming Interface User's Guide*.

If you need to know where programming interface information is described, or about the definitions of the different types of information in the CICS library, you should read the *CICS Family: Library Guide*.

Who this book is for

This book is for those who are responsible for debugging CICS systems and application programs.

What you need to know to understand this book

This book assumes that you have a good knowledge of CICS. If you are using the book to resolve system problems, you need to be familiar with the books that tell you how to install and use a CICS system.

A note about terms used in this book

Throughout this book, the term *APPC* is used to mean LUTYPE6.2. For example, APPC session is used instead of LUTYPE6.2 session.

Bibliography

CICS/ESA 4.1 library

Evaluation and planning	
Release Guide	GC33-1161
Migration Guide	GC33-1162
General	
CICS Family: Library Guide	GC33-1226
Master Index	SC33-1187
User's Handbook	SX33-1188
Glossary (softcopy only)	GC33-1189
Administration	
Installation Guide	SC33-1163
System Definition Guide	SC33-1164
Customization Guide	SC33-1165
Resource Definition Guide	SC33-1166
Operations and Utilities Guide	SC33-1167
CICS-Supplied Transactions	SC33-1168
Program Directory	GC33-1200
Programming	
Application Programming Guide	SC33-1169
Application Programming Reference	SC33-1170
System Programming Reference	SC33-1171
Sample Applications Guide	SC33-1173
Distributed Transaction Programming Guide	SC33-1174
Front End Programming Interface User's Guide	SC33-1175
Diagnosis	
Problem Determination Guide	SC33-1176
Messages and Codes	SC33-1177
Diagnosis Handbook	LX33-6093
Diagnosis Reference	LY33-6082
Data Areas	LY33-6083
Supplementary Data Areas	LY33-6081
Communication	
Intercommunication Guide	SC33-1181
CICS Family: Inter-product Communication	SC33-0824
CICS Family: Communicating from CICS/ESA and CICS/VSE	SC33-0825
Special topics	
Recovery and Restart Guide	SC33-1182
Performance Guide	SC33-1183
CICS-IMS Database Control Guide	SC33-1184
CICS-RACF Security Guide	SC33-1185
Shared Data Tables Guide	SC33-1186
External CICS Interface	SC33-1390

Other CICS books

- *CICS Application Programming Primer (VS COBOL II)*, SC33-0674
- *CICS Application Migration Aid Guide*, SC33-0768
- *Transaction Processing: Concepts and Products*, GC33-0754
- *CICS Family: API Structure*, SC33-1007
- *CICS/ESA XRF Guide* for CICS/ESA Version 3 Release 3, SC33-0661
- *CICS Family: General Information*, GC33-0155
- *CICS/ESA Facilities and Planning Guide* for CICS/ESA Version 3 Release 3, SC33-0654
- *IBM CICS Transaction Affinities Utility MVS/ESA*, SC33-1159

Part 1. Approach to problem determination

Part 1 contains:

Chapter 1. Introduction to problem determination

This book tells you how to find the reasons for problems with your CICS system. Usually, you start with a symptom, or set of symptoms, and trace them back to their cause. This process is called **problem determination**.

Problem determination and problem solving

Problem determination is not the same as problem solving.

Often, the process of problem determination enables you to solve your problem. For example:

- If the cause of the problem is an error in an application program, you can solve the problem by fixing the error.
- If the cause of the problem is an error in your system programming (such as an error in resource definition), you can solve the problem by correcting the error.

However, you may not always be able to solve a problem after determining its cause. For example:

- A performance problem may be caused by a limitation of your hardware.
- You may find that the cause of a problem is in the CICS code. If this happens, you need to contact your IBM Support Center for a solution.

How this book can help you

In this book, we start with the symptoms of the problem, and try to use these symptoms to classify it. For each class of problem we suggest possible causes, and techniques you can use to establish what the cause is.

Always assume first that the problem has a simple cause (for example, an application programming error). If, as a result of investigation, you find that the cause of the problem is not straightforward, go on to consider possible causes that may be more difficult to determine (and to solve). Having exhausted all other possibilities, consider the possibility that the cause of the problem may be in the CICS code itself (in which case you need help from IBM).

Figure 1 on page 4 shows you which part of the book to read first.

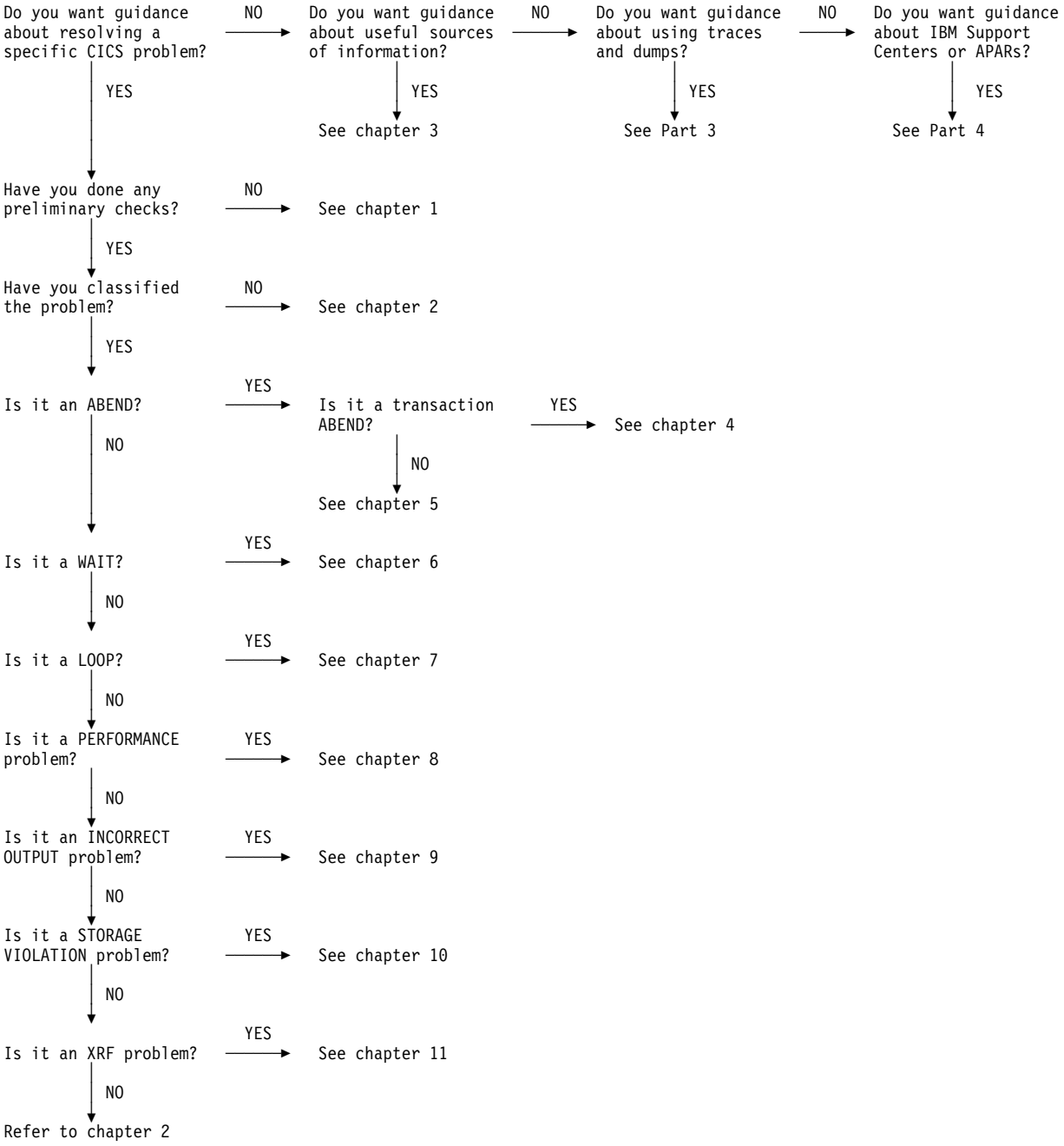


Figure 1. Where to look first

Preliminary checks

Before starting problem determination in detail, it is worth considering the facts to see if there is an obvious cause of the problem, or a likely area in which to start your investigation. This approach to debugging can often save work by highlighting a simple error, or by narrowing down the range of possibilities.

The sections that follow raise some fundamental questions that you need to consider.

As you go through the questions, make a note of anything that might be relevant to the problem. Even if the observations you record do not at first suggest a cause, they could be useful to you later if you need to carry out systematic problem determination.

Has the CICS system run successfully before?

If the CICS system has not run successfully before, it is possible that you have not yet set it up correctly. You need to refer to other books in the CICS/ESA 4.1 library for guidance on doing this.

If you are currently migrating to CICS/ESA 4.1, ensure that you are aware of all the changes that have been made for this release. For details of these, see the *CICS/ESA Migration Guide*.

Are there any messages explaining the failure?

If a transaction abends, and the task terminates abnormally, CICS sends a message reporting the fact to the CSMT log (or your site replacement). If you find a message there, it might immediately suggest a reason for the failure.

Were there any unusual messages associated with CICS start up, or while the system was running before the error occurred? These might indicate some system problem that prevented your transaction from running successfully.

If you see any messages that you don't understand, you can use the CICS messages transaction, CMAC, for online message information. If you don't have access to a CICS system to run the CMAC transaction, look in the *CICS/ESA Messages and Codes* manual for an explanation and, perhaps, a suggested course of action that you can take to resolve the problem.

Can the failure be reproduced?

If the failure is reproducible, consider the conditions under which it can be reproduced:

- Can you identify any application that always seems to be in the system when the problem occurs? If so, examine the application to see if it is in error. Apart from application coding errors, consider whether you have defined sufficient resources. Typically, if you had not defined sufficient resources, you would find that the problem is related to the number of users of the application. For example, there could be too few strings defined for a VSAM file.

- Are you using exit programming interface (XPI) calls? If so, be sure to observe the XPI protocols and restrictions exactly. For programming information about the XPI, see the *CICS/ESA Customization Guide*.

  The exit programming interface enables you to invoke a domain and enter its environment directly; using it incorrectly can cause severe CICS system problems. Here are some particular points for your attention:

  – Are the input parameters correct? If their format is not valid, they are rejected by the called domain, and an exception trace is made. If their values are acceptable to the domain but inappropriate for the system, they could cause unpredictable effects.

  – Be aware that you cannot use some XPI calls within certain user exits. If you do, the results can be unpredictable, and can cause CICS to stall or abend.

- If the problem is not related to any particular application, consider your CICS system definition parameters. Experience has shown that poorly defined parameters are often the cause of problems in CICS systems. You can find guidance about setting up your CICS system in the *CICS/ESA System Definition Guide*.

- Does the problem seem to be related to system loading? If so, the system might be running near its maximum capacity, or it might be in need of tuning. For guidance about dealing with this sort of problem, see the *CICS/ESA Performance Guide*.

Does the failure occur at specific times of day?

If the failure occurs at specific times of day, it could be that it depends on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so those are the times when load-dependent failures are most likely to happen. If your CICS network extends across more than one time zone, peak system loading might seem to you to occur at some other time of day.

Is the failure intermittent?

If you have an intermittent failure, particularly one that does not always show the same symptoms, the problem is likely to be very difficult to resolve. An intermittent failure can be caused by a “random” storage overlay. Furthermore, the transaction that caused the error might have been deleted from the system long before the symptoms are seen.

A method you can use to investigate random overlays is described in Chapter 10, “Dealing with storage violations” on page 189.

Have any changes been made since the last successful run?

When you are considering changes that might recently have been made, think about the CICS system, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you don't yet know about might have been run on the system.

- If an **APAR** or **PTF** has been applied to CICS, check that no error message was obtained. If the installation was successful, check with the IBM Support Center for any APAR or PTF error.

  If an APAR or PTF has been applied to any other program, consider the effect it might have on the way CICS interfaces with it.

- If a **hardware modification** has been carried out, it is possible that your CICS system does not function as it did before the change.

- If a **new or modified application** has just been included, check for error messages in the output from these sources:

  – Translator
  – Compiler
  – Assembler
  – Linkage editor.

- If your **initialization procedure** has been changed, for example by JCL, CICS system initialization or override parameters, or VTAM CONFIG/LIST, consider whether that might be the cause of the problem. In addition, check for error messages sent to the console during initialization.

- If **resource definitions** have been made using CEDA, they must be installed before they are available to the running CICS system.

  If the definitions were made but not installed when CICS was last terminated, they might not have been preserved over the termination and subsequent start up. In general, changes made to the CSD but not installed are not visible when the CICS system is *warm* started. However, if the change was in a group in the GRPLIST specified on a *cold* start, it is effectively installed during startup. (Changes which have been installed are not visible after a cold start unless they were made to a group in the GRPLIST.)

  If START=AUTO was specified in the system initialization table, or as an override, you need to examine the job log to find out how the CICS system last came up.

  For detailed guidance about the ways in which resources can be defined and installed, see the *CICS/ESA Resource Definition Guide*.

Are specific parts of the network affected by the problem?

You might be able to identify specific parts of the network that are affected by the problem. If you can, look for any explanatory message from the access method. Even if no message has been sent to the console, you might find one in the CSNE log.

Have you made any network-related changes that might account for the problem?

If a single terminal is affected, there could be an error in the terminal definition. Consider both the TERMINAL definition, and the TYPETERM definition it uses.

If a number of terminals are affected, try to identify some factor that is common to all of them. For example:

- Do they use the same TYPETERM definition? If so, it is likely that there is an error in that TYPETERM definition.
- Are they non-VTAM terminals all using a single line? If so, the line might have gone out of service, or there could be a problem with the access method.
- Is the whole network affected? If so, CICS has probably stalled. See "What to do if CICS has stalled" on page 137 for advice about dealing with CICS system stalls.

Has the application run successfully before?

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Before you answer **YES** to this question, consider these subsidiary questions:

- **Have any changes been made to the application since it last ran successfully?**

  If so, it is likely that the error lies somewhere in the new or modified part of the application. Examine the changes, and see if you can find an obvious reason for the failure.

  If you used RDO options DEFINE or ALTER to define or alter a transaction, program, or mapset, the changes won't be invoked until you use the INSTALL option.

  If you changed any maps, be sure that you created both a new load-module (TYPE=MAP) and a new DSECT (TYPE=DSECT), and then recompiled every program using the new DSECT. Use the CEMT commands:

  ```
  CEMT SET PROGRAM(mapset) NEWCOPY
  CEMT SET PROGRAM(all programs) NEWCOPY
  ```

  (See the *CICS/ESA CICS-Supplied Transactions* manual for guidance about the CEMT transaction.)

- **Have all the functions of the application been fully exercised before?**

  Could the failure have occurred when part of the application that had never been invoked before was used for the first time? If so, the error probably lies in that part of the application. Try to find out what the application was doing when it failed, and check the source code in that part of the program for errors.

  If a program has been run successfully on many previous occasions, examine the contents of any records, screen data, PF keys, or files that were being processed when the error occurred. They may contain some unusual data value that causes a rarely used path in the program to be invoked.

  Check that the records required at the time of the error can be retrieved successfully. If there is more than one method of accessing the information, make sure you test that the data can be accessed in precisely the way the program attempts to access it.

  Check all fields within the records at the time of the error to see if they contain data in a format acceptable to the program.

The application has not run successfully before

If your application has not yet run successfully, you need to examine it carefully to see if you can find any errors.

Before you look at the code, examine the output from the **translator**, the **compiler** or **assembler**, and the **linkage editor**, to see if any errors have been reported. If your application failed to translate, compile or assemble, or link-edit cleanly into the correct load library, it will also fail to run if you attempt to invoke it.

If the documentation shows that each of these steps was accomplished without error, you must consider the coding logic of the application. Do the symptoms of the failure indicate the function that is failing and, therefore, the piece of code in error? The following is a list of some programming errors commonly found in applications:

- **CICS areas are addressed incorrectly**
- **The rules for quasi-reentrancy are not followed**
- **Transient data is managed incorrectly**
- **File resources are not released**
- **Storage is corrupted by the program**
- **Return codes from CICS requests are ignored**.

What to do next

Perhaps the preliminary checks have enabled you to find the cause of the problem. If so, you should now be able to resolve it, possibly with the help of other books in the CICS library and in the libraries of other licensed programs.

If you have not yet found the cause, you must start to look at the problem in greater detail. Begin by finding the best category for the problem, using the approach described in Chapter 2, “Classifying the problem” on page 11.

Chapter 2. Classifying the problem

The purpose of this chapter is to help you classify your problem into one of the categories used by the IBM Support Center for its service procedures. IBM Support Center staff have found that classifying the problem first is a good approach to problem determination.

Note: For problem determination of the ONC/RPC feature, refer to the ONC RPC
Guide.

Symptom keywords as a basis for classifying problems

IBM keeps records of all known problems with its licensed programs on the RETAIN database. IBM Support Center staff continually update the database as new problems are reported, and they regularly search the database to see if problems they are told about are already known.

If you have the IBM INFORMATION/ACCESS licensed program, 5665-266, you can look on the RETAIN database yourself. Each of the problems there has a classification type.

For software problems, the classifications you will see are:

ABEND
WAIT
LOOP
POOR PERFORMANCE, or **PERFM**
INCORRECT OUTPUT, or **INCORROUT**
MESSAGE.

All but the last of these, MESSAGE, are considered in this book. If you receive a CICS error message, you can use the CICS message transaction, CMAC, for online message information. If you don't have access to a running CICS system, look in the *CICS/ESA Messages and Codes* manual for an explanation. If you get a message from another IBM program, or from the operating system, you need to look in the messages and codes book from the appropriate library for an explanation of what that message means.

The *CICS/ESA Messages and Codes* manual might give you enough information to solve the problem quickly, or it might redirect you to this manual for further guidance. If you are unable to deal with the message, you may eventually need to contact the IBM Support Center for help.

One sort of problem that might give rise to a number of symptoms, usually ill-defined, is that of poor application design. Checking the design of an application is beyond the scope of this book, but one instance is described in "Poor application design" on page 19, where you will find there an example of how bad design can give rise to an application with poor usability.

Some more ways in which this manual classifies problems

In addition to the RETAIN classifications used by the IBM Support Centers, this manual considers the following types of problem to belong in classes of their own:

STORAGE VIOLATIONS
XRF ERRORS.

Extended Recovery Facility (XRF) errors can be classified in a straightforward way, but confirming that you have a storage violation can be difficult. Unless you get a CICS message stating explicitly that you have a storage violation, you could get almost any symptom, depending on what has been overlaid. You might, therefore, classify it initially as one of the RETAIN symptom types described in “Symptom keywords as a basis for classifying problems” on page 11.

Using the symptoms to classify the problem

The paragraphs that follow are to help you to classify the problem on the basis of the symptoms you observe.

The symptoms might enable you to classify the problem correctly at once, but sometimes classification is not so straightforward. You may need to consider the evidence carefully before making your decision. You might need to make a “best guess”, and then be prepared to reconsider later on the basis of further evidence.

Look for the section heading that most nearly describes the symptoms you have, and then follow the advice given there.

CICS has stopped running

There are three main reasons why CICS might unexpectedly stop running:

1. There could be a CICS system abend.
2. CICS could be in a wait state. In other words, it could have stalled.
3. A program could be in a tight loop.

Consider, too, the possibility that CICS might still be running, but only slowly. Be certain that there is no activity at all before carrying out the checks in this section. If CICS *is* running slowly, you probably have a performance problem. If so, read “CICS is running slowly” on page 13 to confirm this before going on to Chapter 8, “Dealing with performance problems” on page 153 for advice about what to do next.

If CICS *has* stopped running, look for any message that might explain the situation. The message might appear in either of the following places:

- **The MVS console**. Look for any message saying that the CICS job has abnormally terminated. If you find one, it means that a CICS system abend has occurred and that CICS is no longer running. In such a case, you need to examine the CSMT log (see below) to see which abend message has been written there.

  If you don’t find any explanatory message on the MVS console, check in the CSMT log to see if anything has been written there.

- **The CSMT log**. CSMT is the transient data destination to which abend messages are written. If you find a message there, use the CMAC transaction

or look in the *CICS/ESA Messages and Codes* manual to make sure there has been a CICS *system* abend.

If you see only a *transaction* abend message in the CSMT log, that won't account for CICS itself not running, and you should not classify the problem as an abend. A faulty transaction could hold CICS up, perhaps indefinitely, but CICS would resume work again if the transaction abended.

Here are two examples of messages that might accompany CICS system abends, and which you would find on the CSMT log:

DFHST0001 *applid* **An abend (code** *aaa/bbbb***) has occurred at offset X'***offset***' in module** *modname*.

DFHSR0601 Program interrupt occurred with system task *taskid* **in control**

If you get either of these messages, or any others for which the system action is to terminate CICS, turn to Chapter 5, "Dealing with CICS system abends" on page 45 for advice on what to do next.

If you can find no message saying that CICS has terminated, it is likely that the CICS system is in a wait state, or that some program is in a tight loop and not returning control to CICS. These two possibilities are dealt with in Chapter 6, "Dealing with waits" on page 57 and Chapter 7, "Dealing with loops" on page 143, respectively.

CICS is running slowly

If CICS is running slowly, it is likely that you have a performance problem. It could be because your system is badly tuned, or because it is operating near the limits of its capacity. You will probably notice that the problem is worst at peak system load times, typically at mid-morning and mid-afternoon. If your network extends across more than one time zone, peak system load might seem to you to occur at some other time.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed transaction could be the cause. You might classify the problem initially as "poor performance", but be prepared to reconsider your classification later.

The following are some individual symptoms that could contribute to your perception that CICS is running slowly:

- **Tasks take a long time to start running**
- **Some low priority tasks won't run at all**
- **Tasks start running, but take a long time to complete**
- **Some tasks start running, but don't complete**
- **No output is obtained**
- **Terminal activity is reduced, or has ceased**.

Some of these symptoms do not, in isolation, necessarily mean that you have got a performance problem. They could indicate that some task is in a loop, or is waiting on a resource that is not available. Only you can judge whether what you see should be classified as "poor performance", in the light of all the evidence you have.

You might be able to gather more detailed evidence by using the tools and techniques that CICS provides for collecting performance data. The following is a summary of what is available:

- CICS statistics. You can use these to gather information about the CICS system as a whole, without regard to tasks.
- CICS monitoring. You can use this facility to collect information about CICS tasks.
- CICS tracing. This is not a specific tool for collecting performance data, but you can use it to gather detailed information about performance problems.

For guidance about using these tools and techniques, and advice about performance and system tuning in general, see the *CICS/ESA Performance Guide*.

You can find guidance about identifying specific performance bottlenecks in your CICS system in Chapter 8, “Dealing with performance problems” on page 153.

A task fails to start

If a task fails to start, look first in the CSMT and CSNE logs for any explanatory message. If you don’t find one, the task is possibly being prevented from starting because either the system is running at the MXT limit, the transaction is queuing for admittance to a transaction class, or for other performance reasons.

Classify the problem tentatively as “poor performance”, and turn to Chapter 8, “Dealing with performance problems” on page 153 for further guidance.

A task is running slowly

If just one task is running slowly, it is likely that the explanation lies with the task itself. It could be in a loop, or it could periodically be entering a wait state. You need to decide which of these possibilities is the most likely before starting systematic problem determination. The ways that you might distinguish between waits and loops are described in “Distinguishing between waits, loops, and poor performance” on page 16.

Note: Don’t overlook the possibility that the task might simply be doing unnecessary work that does not change the final result—for example, starting a skip sequential browse with large gaps between the keys, or failing to finish one because it is holding on to resources.

A task stops running at a terminal

When a task stops running at a terminal, you will notice either or both of these symptoms:

- **No output is obtained at the terminal**
- **The terminal accepts no input**.

First, make sure that the task is still in the system. Use CEMT INQ TASK to check its status, and make sure that it has not simply ended without writing back to the terminal.

If the terminal has a display unit, check to see whether a special symbol has been displayed in the operator information area that could explain the fault. If the

operator information area is clear, next check to see that no message has been sent to any of the transient data destinations used for error messages, for example:

- CDBC, the destination for DBCTL related messages
- CSMT, the destination for terminal error and abend messages
- CSTL, the destination for terminal I/O error messages
- CSNE, the destination for error messages written by DFHZNAC and DFHZNEP.

For details of the destinations used by CICS, see the *CICS/ESA System Definition Guide*. If you can find no explanation for the problem, the fault is probably associated with the task running at the terminal. These are the possibilities:

- The task is in a wait state
- The task is in a loop
- There is a performance problem.

Read “Distinguishing between waits, loops, and poor performance” on page 16 to find out which of these is the most likely explanation. You can then refer to the appropriate chapter for advice about dealing with the problem.

A transaction has abended

If the transaction abended when you ran your application, CICS gives you an error message on your screen as well as a message on the CSMT log.

Use the CMAC transaction or look in the *CICS/ESA Messages and Codes* manual for an explanation of the message, and, perhaps, advice about what you should do to solve the problem. If the code is not there, or the explanation or advice given is not sufficient for you to solve the problem, turn to Chapter 4, “Dealing with transaction abends” on page 31.

You have obtained some incorrect output

Incorrect output might be regarded as any sort of output that you were not expecting. However, use the term with care in the context of problem determination, because it might be a secondary effect of some other type of error. For example, looping could be occurring if you get any sort of repetitive output, even though that output is not what you had expected. Also, CICS responds to many errors that it detects by sending messages. You might regard the messages as “incorrect output”, but they are only symptoms of another type of problem.

If you have received an unexpected message, and its meaning is not at first clear, use the CMAC transaction or look in the *CICS/ESA Messages and Codes* manual for an explanation. It might suggest a simple response that you can make to the message, or it might redirect you to this manual for further guidance.

These are the types of incorrect output that are dealt with in this manual:

- **Incorrect trace or dump data:**
  - Wrong destination
  - Wrong type of data captured
  - Correct type of data captured, but the data values were unexpected.
- **Wrong data displayed on the terminal**.

You can find advice about investigating the cause of any of these types of incorrect output in Chapter 9, “Dealing with incorrect output” on page 163.

A storage violation has occurred

When CICS detects that storage has been corrupted, this message is sent to the console:

DFHSM0102 *applid* **A storage violation (code X'***code***') has been detected by module** *modname*.

If you see this message, or you know (through other means) that a storage violation has occurred, turn to Chapter 10, “Dealing with storage violations” on page 189 for advice about dealing with the problem.

In many cases storage violations go undetected by CICS, and you only find out that they have occurred when something else goes wrong as a result of the overlay. You could, for example, get a program check because code or data has been overlaid. You might suspect some other type of problem at first, and only after starting your investigation find that a storage violation has occurred.

You can avoid many storage violations by enabling transaction isolation, storage protection, and command protection.

An XRF error has occurred

If an XRF error has occurred, turn to Chapter 11, “Dealing with XRF errors” on page 199 for advice on what to do.

Distinguishing between waits, loops, and poor performance

Waits, loops, and poor performance can be quite difficult to distinguish, and in some cases you need to carry out quite a detailed investigation before deciding which classification is the right one for your problem.

Any of the following symptoms could be caused by a wait, or a loop, or by a badly tuned or overloaded system:

- **One or more user tasks in your CICS system fails to start**
- **One or more tasks stays suspended**
- **One or more tasks fails to complete**
- **No output is obtained**
- **Terminal activity is reduced, or has ceased**
- **The performance of your system is poor**.

Because it can be difficult to make a correct classification, consider the evidence carefully before adopting a problem solving strategy.

This section gives you guidance about choosing the best classification. When you have decided on that, read the appropriate chapter later in this manual for further advice. However, note that in some cases your initial classification could be wrong, and you then need to reappraise the problem.

Waits

For the purpose of problem determination, a wait state is regarded as a state in which the execution of a task has been suspended. That is, the task has started to run, but it has been suspended without completing and has subsequently failed to resume.

The task might typically be waiting for a resource that is unavailable, or it might be waiting for an ECB to be posted. A wait might affect just a single task, or a group of tasks that may be related in some way. If none of the tasks in a CICS region is running, CICS is in a wait state. The way to handle that situation is dealt with in "What to do if CICS has stalled" on page 137.

If you are authorized to use the CEMT transaction, you can find out which user tasks or CICS-supplied transactions are currently suspended in a running CICS system using CEMT INQ TASK. Use the transaction several times, perhaps repeating the sequence after a few minutes, to see if any task stays suspended. If you do find such a task, look at the resource type that it is waiting on (the value shown for the HTYPE option). Is it unreasonable that there should be an extended wait on the resource? Does the resource type suggest possible causes of the problem?

You can use EXEC CICS INQUIRE TASK or EXEC CICS INQUIRE TASK LIST as alternatives to the CEMT transaction. You can execute these commands under CECI, or from a transaction.

Use INQUIRE TASK LIST to find the task numbers of all SUSPENDED, READY, and RUNNING user tasks. If you use this command repeatedly, you can see which tasks stay suspended. You may also be able to find some relationship between several suspended tasks, perhaps indicating the cause of the wait.

If it seems fairly certain that your problem is correctly classified as a wait, and the cause is not yet apparent, turn to Chapter 6, "Dealing with waits" on page 57 for guidance about solving the problem.

However, you should allow for the possibility that a task may stay suspended because of an underlying performance problem, or because some other task may be looping.

If you can find no evidence that a task is waiting for a specific resource, you should not regard this as a wait problem. Consider instead whether it is a loop or a performance problem.

Loops

A loop is the repeated execution of some code. If you have not planned the loop, or if you have designed it into your application but for some reason it fails to terminate, you get a set of symptoms that vary depending on what the code is doing. In some cases, a loop may at first be diagnosed as a wait or a performance problem, because the looping task competes for system resources with other tasks that are not involved in the loop.

The following are some characteristic symptoms of loops:

- **The 'system busy' symbol** is permanently displayed in the operator information area of a display unit, or stays displayed for long periods.

- **The transaction abends with abend code AICA**.
- **CPU usage is very high**, perhaps approaching 100%, yet some tasks stay suspended or ready, but not running, for a long time.

  You can check what the CPU usage is for any MVS job by using the DISPLAY ACTIVE command at the MVS console to display the active users.
- **There is reduced activity at terminals**, or possibly no activity at all.
- **One or more CICS regions appear to be stalled**, or to be continuing only slowly.
- **No CICS messages are written** to any destination, when they are expected.
- **No new tasks can be started**.
- **Existing tasks remain suspended**.
- **The CEMT transaction cannot be used**.
- **Repetitive output obtained**. Try looking in these areas:
  - Terminals, and the system console.
  - Temporary storage queues. You can use CEBR to browse them online.
  - Data files and CICS journals.
  - Trace tables, but remember that some loops are intentional—some CICS system tasks use them, for example, to see if there is any work to be done.
- **Demand for storage excessive**. If the loop contains a GETMAIN request, storage is acquired each time this point in the loop is passed, as long as sufficient storage to satisfy the request remains available. If storage is not also freed in the loop, CICS eventually goes short on storage (SOS) in one of the DSAs. You then get a message reporting that CICS is under stress in one of these areas.

  One further effect is that tasks issuing unconditional GETMAIN requests are suspended more often as the loop continues and storage is progressively used up. Tasks making storage requests do not need to be in the loop to be affected in this way.
- **Statistics show a large number of automatically initiated tasks.**
- **There are large numbers of file accesses** shown for an individual task.

Some loops can be made to give some sort of repetitive output. Waits and performance problems never give repetitive output. If the loop produces no output, a repeating pattern can sometimes be obtained by using trace. A procedure for doing this is described in Chapter 7, “Dealing with loops” on page 143.

If you are able to use the CEMT transaction, try issuing CEMT INQ TASK repeatedly. If the same transaction is shown to be running each time, this is a further indication that the task is looping. However, note that the CEMT transaction is always running when you use it to inquire on tasks.

If different transactions are seen to be running, this could still indicate a loop, but one that involves more than just a single transaction.

If you are unable to use the CEMT transaction, it may be because a task is looping and not allowing CICS to regain control. A procedure for investigating this type of situation is described in “What to do if CICS has stalled” on page 137.

Consider the evidence you have so far. Does it indicate a loop? If so, turn to Chapter 7, “Dealing with loops” on page 143, where there are procedures for defining the limits of the loop.

Poor performance

A performance problem is considered to be one in which system performance is perceptibly degraded, either because tasks fail to start running at all, or because they take a long time to complete once they have started.

In extreme cases, some low-priority tasks may be attached but then fail to be dispatched, or some tasks may be suspended and fail to resume. The problem might then initially be regarded as a wait.

If you get many messages telling you that CICS is under stress, this can indicate that either the system is operating near its maximum capacity, or a task in error has used up a large amount of storage—possibly because it is looping.

You see one of the following messages when CICS is under stress in one of the DSAs:

DFHSM0131 *applid* **CICS is under stress (short on storage below 16MB)**

DFHSM0133 *applid* **CICS is under stress (short on storage above 16MB)**

If there is no such indication, refer to Chapter 8, “Dealing with performance problems” on page 153 for advice on investigating the problem. However, before doing so, be as sure as you can that this is best classified as a performance problem, rather than a wait or a loop.

Poor application design

If you have only a poorly defined set of symptoms that might indicate a loop, or a wait, or possibly a performance problem with an individual transaction, consider the possibility that poor design might be to blame.

This book does not deal with the principles of application design, or how to check whether poor design is responsible for a problem. However, one example is given here, to show how poor design of an application gave rise to symptoms which were at first thought to indicate a loop.

Environment: CICS and DL/I using secondary indexes. The programmer had made changes to the application to provide better function.

Symptoms: The transaction ran and completed successfully, but response was erratic and seemed to deteriorate as the month passed. Towards the end of the month, the transaction was suspected of looping and was canceled. No other evidence of looping could be found, except that statistics showed a high number of I/Os.

Explanation: The programmer had modified the program to allow the user to compare on the last name of a record instead of the personnel number, which it had done in the past. The database was the type that grew through the month as activity was processed against it.

It was discovered that in making the change, the program was no longer comparing on a field that was part of the key for the

secondary index. This meant that instead of searching the index for the key and then going directly for the record, every record in the file had to be read and the field compared. The structure of the source program had not changed significantly; the number of database calls from the program was the same, but the number of I/Os grew from a few to many thousands at the end of the month.

Note that these symptoms might equally well have pointed to a performance problem, although performance problems are usually due to poorly tuned or overloaded systems, and affect more than just one transaction. Performance problems tend to have system wide effects.

Where to look next

For *transaction abends*, Chapter 4, “Dealing with transaction abends” on page 31

For *system abends*, Chapter 5, “Dealing with CICS system abends” on page 45

For *waits*, Chapter 6, “Dealing with waits” on page 57

For *loops*, Chapter 7, “Dealing with loops” on page 143

For *poor performance*, Chapter 8, “Dealing with performance problems” on page 153

For *incorrect output*, Chapter 9, “Dealing with incorrect output” on page 163

For *storage violations*, Chapter 10, “Dealing with storage violations” on page 189

For *XRF errors*, Chapter 11, “Dealing with XRF errors” on page 199.

If you have already decided that you should refer the problem to the IBM Support Center, you can find advice about dealing with the Center in Chapter 17, “IBM program support” on page 287.

Chapter 3. Sources of information

You should find some of the following sources of information useful in problem determination.

- Your own documentation
- Manuals for the products you are using
- Source listings and link-edit maps
- Abend codes and error messages
- The symptom strings from CICS system and transaction dumps, or from message DFHME0116
- Change log
- Dumps
- Statistics
- Monitoring information
- Transaction inputs and outputs
- Traces.

Your own documentation

"Your own documentation" is the collection of information produced by your organization about what your system and applications do and how they do it. How much of this kind of information you need depends on how familiar you are with the system or application, and could include:

- Program descriptions or functional specifications
- Record layouts and file descriptions
- Flowcharts or other descriptions of the flow of activity in a system
- Statement of inputs and outputs
- Change history of a program
- Change history of your installation
- Auxiliary trace profile for your transaction
- Statistical and monitoring profile showing average inputs, outputs, and response times.

Manuals

"Manuals" means the manuals in the CICS/ESA 4.1 library and the libraries for any other products you use with your application.

Make sure that the level of any manual you refer to matches the level of the system you are using. Problems often arise through using either obsolete information or information about a level of the product that is not yet installed.

Source listings and link-edit maps

Include the source listings of any applications written at your installation with your set of documentation. (They often form the largest single element of documentation. Large installations with thousands of programs might prefer to keep such listings on microfiche or CD-ROM.) Make sure you include the relevant linkage-editor output with your source listings to avoid wasting time trying to find your way through a load module with an out-of-date link map. Be sure to include the JCL at the beginning of your listings, to show the libraries that were used and the load library in which the load module was placed.

Abend codes and error messages

Messages are sent to several transient data destinations, for example:

CSMT for terminal error and abend messages
CSNE for messages issued by DFHZNAC
CSTL for terminal I/O error messages
CDBC for messages concerning DBCTL
CSFL for file control messages.

For a list of the destinations used by CICS, see the *CICS/ESA System Definition Guide*. Use a copy of the appropriate messages and codes manual to look up any messages whose meaning you do not know. Make sure that you also have some documentation of application messages and codes for programs that were written at your installation, as well as a copy of the *CICS/ESA Messages and Codes* manual.

Symptom strings

CICS produces symptom strings in CICS system and transaction dumps and in message DFHME0116.

The symptom string provides a number of keywords that can be directly typed in and used to search the RETAIN database. If your installation has access to the IBM INFORMATION/ACCESS licensed program, 5665-266, you can search the RETAIN database yourself. If you report a problem to the IBM Support Center, you are often asked to quote the symptom string.

Although the symptom string is designed to provide keywords for searching the RETAIN database, it can also give you significant information about what was happening at the time the error occurred, and it might suggest an obvious cause or a likely area in which to start your investigation.

Change log

The information in the change log can tell you of changes made in the data processing environment that may have caused problems with your application program. To make your change log most useful, include the data concerning hardware changes, system software (such as MVS and CICS) changes, application changes, and any modifications made to operating procedures.

Dumps

Dumps are an important source of detailed information about problems. Whether they are the result of an abend or a user request, they allow you to see a snapshot of what was happening in CICS at the moment the dump is taken. Chapter 15, “Using dumps in problem determination” on page 249 contains guidance about using dumps to locate problems in your CICS system. However, because they do only provide a “snapshot,” you may need to use them in conjunction with other sources of information relating to a longer period of time, such as logs, traces, and statistics.

Statistics

Statistics are often overlooked as a source of debugging information, but those that relate to an application program can help solve problems. It is useful to have a statistical profile (as mentioned in “Your own documentation” on page 23) to use for problem determination. If you compare the information in the profile with the statistical information produced by CICS, any differences you find may indicate the source of a problem.

Statistics are most often used in system tuning and diagnosis, but they also contain information that can indicate problems with the way your application handles resources. For example, you may notice from these statistics that tables are being loaded, or programs linked, for which there is no known requirement.

You can also use statistics to check terminals, files, queues, and so on for irregularities in their activity. For example, if a terminal has a number of errors recorded for a particular transaction that equal the number of times that transaction was run, this may indicate that an incorrect data stream is being sent to that terminal. See the *CICS/ESA Performance Guide* for more information about using statistics.

Monitoring

You can use CICS monitoring to provide information for debugging applications. In addition to the system-defined event monitoring points (EMPs) that already exist within CICS code itself, you can define user event monitoring points in your own application programs by using the EXEC CICS MONITOR POINT command.

At a user EMP, you can add your own data (up to 256 counters, up to 256 clocks, and a single character string of up to 8192 bytes) to fields reserved for you in performance class monitoring data records. You could use these extra EMPs to count how many times a certain event happens, or to time the interval between two events. Your definitions in the Monitoring Control Table (MCT) specify the type and number of fields that are available for your use within each task’s performance record. For further information on the MCT see the *CICS/ESA Resource Definition Guide*. See the *CICS/ESA Application Programming Reference* manual for programming information on syntax and options of the MONITOR POINT command.

When your monitoring data has been collected, you can read it into a database using, for example, the Service Level Reporter Version 2 (SLR II).

See the *CICS/ESA Performance Guide* for guidance about choosing performance tools. See the *CICS/ESA CICS-Supplied Transactions* manual for information about the transactions needed to invoke them.

Transaction inputs and outputs

Transaction inputs and outputs can be divided into the following areas:

- Terminal data
- Transient data and temporary storage
- Passed information
- Files and databases.

Terminal data

Terminal data is very important in solving problems, because it can help you answer the following questions:

> What data did you enter just before the transaction failed?
> Was there any output? If so, what did it look like?

The more you know about the information that was ***input*** at the terminal on which the transaction failed, the better your chance of duplicating the problem in a test environment. However, this information may not be precise, especially if there are many fields on the input screen. You are recommended to provide a quick and easy way for terminal operators to report problems, so that they can report the error while they can still see the data on the screen (or at least remember more clearly what it was).

The ***output*** from a transaction is sometimes easier to capture. If you have a locally attached printer, you can make a copy. (The problem may be that the printer output is incorrect.)

The items to look for on the ***input*** side are:

1. Were all necessary input fields entered?
2. Were the contents of the input fields correct?
3. Which transmit key was used, (that is ENTER, a PF key, or a PA key)?

On the ***output*** screen, check the following points:

1. Do all the required fields contain data?
2. Is the data correct?
3. Is the screen format as it was designed?
4. Are there any nondisplay fields used to pass data that may not be protected?

Transient data and temporary storage

If the program explicitly uses any transient data or temporary storage queues, inspect them to see if their content is what you expect. You can use the CICS-supplied transaction, CEBR, to inspect temporary storage queues in some detail. See the *CICS/ESA CICS-Supplied Transactions* for information about this transaction.

Even if the program does not use queues, look at the system queues for CEMT (or your site replacement) and CSTL (and CDBC if you use DBCTL) to see if there are any relevant messages.

The things you might want to look for in the queues are:

1. Are the required entries there?
2. Are the entries in the correct order?
3. Is the queue being ***written*** the same one that is being ***read***?

Passed information

Be particularly careful when you are using the common work area (CWA) because you only have one area for the entire system. A transaction may depend on a certain sequence of transactions and some other program may change that sequence. If you are using the CWA, you must also know if your CICS is split into multiple MRO regions because there is an independent CWA for each MRO region.

Terminal user areas can have problems because the area is associated with a terminal and not a particular transaction.

If you are using tables in the CWA, remember that there is no recovery; if a transaction updates the table and then abends, the transaction is backed out but the change is not.

Files and databases

Files and databases are often the main source of transaction input and output; you should always investigate both these areas whenever a program is having problems.

To do this, you need to use the appropriate utilities and diagnostic tools for the data access methods that you have at your installation.

Check the various indexes in files and databases. If you have more than one method of accessing information, one path may be working well but another path may be causing problems.

When looking through the data in files, pay particular attention to the record layout. The program may be using an out-of-date record description.

Traces

CICS provides a tracing facility that enables you to trace transactions through the CICS components as well as through your own programs. CICS auxiliary trace enables you to write trace records on a sequential device for later analysis.

For information about the tracing facilities provided by CICS, read Chapter 14, “Using traces in problem determination” on page 219.

Part 2. Dealing with the problem

Part 2 contains:

Chapter 4. Dealing with transaction abends

This chapter gives guidance about finding the cause of transaction abends.

When a CICS transaction **abends** (ends abnormally), a transaction abend message and an abend code of four alphanumeric characters are sent to CSMT, the CEMT transient data destination (or your site replacement). This is an example of what the message looks like:

DFHAC2006 *date time applid* **Transaction** *tranid* **program** *program name* **abend** *primary abcode* **at** *termid.*

The message contains several vital pieces of information. It identifies the transaction (*tranid*) that failed, and the program (*program name*) that was being executed when the failure was detected. Most importantly, it gives you the abend code (*abcode*), indicating the nature of the error.

The transaction abend can originate from several places, and the method you use for problem determination depends on the source of the abend. The procedures are described in the sections that follow. As you go through them, you might like to use the worksheet that is included at the end of this chapter to record your findings ("Worksheet for transaction abends" on page 43).

Collecting the evidence

You should find all the evidence you need to investigate the transaction abend in the information sent to the various transient data queues for error messages, and in the transaction dump. If no transaction dump has been produced, it is possible that transaction dumping has been suppressed for the transaction (via the transaction definition), or the dump code entry in the transaction dump code table suppresses dumping. For guidance about changing the dumping options so that you get a transaction dump, refer to Chapter 15, "Using dumps in problem determination" on page 249.

The transaction abend code and the abend message are recorded in the CSMT log. Make a note, too, of any other messages you find there that might relate to the abend, as they could provide additional valuable evidence.

Check also to see if any relevant messages have been sent to the transient data destinations used by CICS to record messages. For a list of destinations used by CICS, see the *CICS/ESA System Definition Guide*. Look in particular for any messages about files, terminals, or printers that you might be attempting to use.

Symptom string

CICS produces a symptom string as part of the transaction dump. The symptom string gives some details about the circumstances of the transaction dump. It might show, for example, that the dump was taken because the transaction abended with the abend code ASRA.

If you refer the problem that caused the dump to be taken to the IBM Support Center, they will use the symptom string to search the RETAIN database for problems similar to it. For an introduction to symptom strings and their contents, see “Looking at the symptom string in the dump” on page 47.

What the abend code can tell you

The first thing that the transaction abend code can indicate is whether or not this was a CICS abend. CICS transaction abend codes begin with the letter “A”. A user program or another product might also use abend codes beginning with “A”. However, if the transaction abend code begins with anything other than “A”, it is an abend code belonging to a user program or to some other product. For the sake of convenience, all such non-CICS abend codes are referred to in this chapter as user abend codes.

For an introduction to the types of transaction abend codes used by CICS and by other IBM products, refer to the *CICS/ESA Messages and Codes* manual.

If you have received a user abend code, it can still be difficult to find out which program is responsible for it unless you have adequate documentation. For this reason, it is good practice for all programmers who issue abends from within their programs to document the codes in a central location at your installation.

As far as vendor products are concerned, the documentation includes, in most cases, a list of abend codes that are issued from the programs making up the products. This list, together with the documentation for your internal applications, should make it possible for you to find what caused the abend. If it is not clear why the user abend was issued, you might need to describe the problem to the owner of the program.

CICS transaction abend codes

Your best source of information on CICS abends is the *CICS/ESA Messages and Codes* manual. That book contains a chapter that lists all transaction abend codes issued by CICS. There is an explanation of why the code was issued, followed by details of system and user actions. The same information is available online, using the CICS-supplied messages and codes transaction, CMAC.

If, after reviewing the material in the *CICS/ESA Messages and Codes* manual, you cannot find the cause of the problem, continue with the procedures described here. The abend codes AICA, ASRA, ASRB and ASRD are dealt with separately because special procedures apply to them. If your abend code was something other than these, use the procedures in “Locating the last command or statement” on page 274, to find the last command that was executed, and then turn to “Analyzing the problem further” on page 41.

AICA abends

If your transaction terminated with abend code AICA, the transaction is likely to have been in a loop. You can find detailed guidance about dealing with loops in Chapter 7, “Dealing with loops” on page 143.

ASRA abends

CICS issues an ASRA abend code when it detects that a program check has occurred within a transaction. Program checks can occur for a wide variety of reasons, but you can find the nature of the error from the program interrupt code in the program status word (PSW). The PSW is used by the machine hardware to record the address of the current instruction being executed, the addressing mode, and other control information. The PSW gives you the address at which the program check occurred, and so it represents a record of the circumstances of the failure.

ASRB abends

A transaction can abend with an abend code of ASRB when a program issues the MVS ABEND macro. For example, BDAM issues this ABEND macro when it detects errors, rather than sending a return code to the calling program. CICS is notified when an MVS abend occurs, and in turn issues an ASRB abend code for the transaction.

Use the procedures outlined in “Locating the last command or statement” on page 274 to find the origin of the abend in your program. That information, together with the description and procedures for ASRB abends given in the *CICS/ESA Messages and Codes* manual, should be sufficient for you to solve the problem.

ASRD abends

A transaction abends with code ASRD if:

- An application program attempts to invoke CICS macros.
- An application program attempts to access the CSA or TCA.
- An application program issues an EXEC CICS ADDRESS CSA command, and attempts to access storage addressed by the pointer that is returned.
- A COBOL application program attempts to access the CSA via a BLL cell.

Any of the above causes a program check that CICS diagnoses as an ASRD abend, rather than the usual ASRA abend. You can use the information in the PSW to investigate the cause of an ASRD abend.

AEYD abends

If command protection is activated by the CMDPROT(YES) option in the system initialization table (SIT), the AEYD transaction abend can occur. CICS/ESA terminates a transaction with this code when an output parameter of an EXEC CICS command addresses storage that the issuing transaction could not itself directly overwrite.

At the time of the abend, register 2 points to the parameter area containing the invalid address. The trace should include an exception trace entry created by DFHEISR. This entry should identify the parameter in error. If the abend is handled, EXEC CICS ASSIGN ASRASTG, ASRAKEY, ASRASPC, and ASRAREGS can give additional information.

To prevent a recurrence of the abend, it is recommended that you correct the program code. Alternatively, changing one or more of the following options may alleviate the problem:

- EXECKEY in the program definition, if storage protection is active
- TASKDATAKEY in the transaction definition
- ISOLATE in the transaction definition, if transaction isolation is enabled.

For further information, see “Avoiding storage violations” on page 189.

Finding where a program check occurred

When a transaction abends with code ASRA or ASRD, the first thing you need to do is find out where the program check occurred. CICS will have attempted to establish this for you. A record of the program in error and the offset of the program check within the program load module are contained in the following places:

- Message DFHAP0001 or DFHSR0001, which will have preceded the abend
- The transaction abend control block (TACB) which will have been created to describe the abend
- Exception trace point ID AP 0781 for an ASRA abend or AP 0783 for an ASRD abend.

See “Interpreting transaction dumps” on page 270.

The offset indicates the point in the program at which the program check occurred. Note that the offset is derived from the PSW next sequential instruction address and so may indicate the instruction **after** the one that failed. Unless the offset is X'FFFFFFFF', turn to “What type of program check occurred?” on page 35.

If the offset appears as X'FFFFFFFF', CICS was unable to establish the location of the program check. If this is the case, use the PSW to obtain the next sequential instruction address. The PSW may be found in the following places:

- The TACB for the abend
- At the head of the formatted transaction dump
- Within the kernel error data block traced by exception trace point IDs AP 0781 or AP 0783.

Now note down the start and end addresses of the different program areas in the transaction dump. Is the next sequential instruction address from the PSW in any of the programs? If so, then that is the program in which the interrupt occurred. Use the procedure described in “Locating the last command or statement” on page 274 to identify the last command executed.

If the address is *outside* all of the programs, one of two things is likely to have happened.

- The program in which the program check occurred was running on your behalf (for example, VSAM or DL/I), but not under CICS control. This is usually caused by incorrect parameters being passed to the program, or parameters being passed in the wrong sequence. These are usually caught and flagged with an appropriate return code, but certain combinations can cause problems.

- Your program might have taken a “wild” branch into some other piece of storage. If the address from the PSW ends in an odd number, this is probably the case, as valid instructions are always on an even address. The address could be within the CICS address space, or anywhere else in virtual storage.

  Often, a wild branch is taken to address zero, because the register that should contain the branch address is set to zero. The PSW usually contains address X'00000004' after such a branch has occurred.

Check the register contents to see whether any of them contains the next sequential instruction address from the PSW, or something close to it. This might help you find out how you got to the wrong address.

If the PSW does point to an instruction in one of your programs, the next thing to consider is the type of program check that occurred. Otherwise, turn directly to “Analyzing the problem further” on page 41.

What type of program check occurred?

Knowing what type of program check occurred can be helpful in finding the cause of the error. This is indicated by the program interrupt code (PIC), which you can find in the PSW at the start of the transaction dump. You can find information about the PSW in ESA/370 from the *IBM Enterprise Systems Architecture/370 Principles of Operation*.

PIC	PIC explanation
1	Operation exception—incorrect operation attempted. **Some possible causes** • Overlaid program • Overlaid register save area, causing incorrect branch • Resource unavailable, but program logic assumed valid address returned and took inappropriate action • Incorrect branch to data that contains no instruction known to the machine • In an assembler-language program, a base register was inadvertently changed.
2	Privileged operation—this program is not authorized to execute this instruction. **Some possible causes** • Incorrect branch to this code; may be due to: – Overlaid register save area – Program overlaid by data that contains the privileged operation code.
3	Execution exception—you are not allowed to EXECUTE an EXECUTE instruction. **Some possible causes** • Incorrect branch to this code

- Incorrect register contents; may be due to:
  - Overlaid register save area
  - Program overlaid by data that contains the incorrect instruction
  - Incorrect program logic.

4 Protection exception—read or write access violation has occurred.

Some possible causes

- Resource unavailable, and return code not checked. Program logic assumed valid address returned and took inappropriate action.
- Incorrect interface parameters to some other program or subsystem (for example, VSAM or DL/I).
- Overlaid register save area, causing incorrect reference to data.
- In an assembler-language program, incorrect initialization or modification of a register used to address data.
- Attempt to access internal control blocks illegally or used a CICS system or application programming macro call.
- Attempt to write to storage for which the application does not have an adequate key. For example, in a CICS system with storage protection, an application running in USER key attempts to write to the CDSA, the ECDSA or the ERDSA.
- Attempt to write to the ERDSA or RDSA when PROTECT is specified for the RENTPGM parameter.
- Attempt to read or write to another transaction's storage. For example, in a system running with transaction isolation, a program executing in USER key may suffer a protection exception when attempting to access the USER key task-lifetime storage of another transaction.
- Storage, passed to CICS as an output parameter through the EXEC interface, that is not addressable by the application issuing the call. The transaction is abended AEYD, and the PSW shows that a protection exception has occurred.

5 Addressing exception—the address that you referenced is not available or is not valid.

Some possible causes

- Incorrect register contents; may be due to:
  - Overlaid register save area.

6 Specification exception—incorrect format of an instruction or invalid registers.

Some possible causes

- Overlaid program
- Incorrect field lengths used in packed decimal multiply and divide instructions
- Branch to an odd-numbered address, caused by an overlaid register save area.

7 Data exception—data invalid in a packed or signed display decimal operation. One, or possibly both, of the operands contain data not suitable for the instruction.

Some possible causes

- Incorrect input data (often because blanks have been used where numeric data is expected)
- Overlaid data
- Overlaid register save area, causing an incorrect branch
- Incorrect program logic, execution of code with uninitialized variables
- Wrong length.

8 through F Arithmetic exceptions, such as divide checks, overflow, and underflow. They differ in the form of arithmetic that was being used—binary, packed decimal, or floating point.

Some possible causes

- Incorrect user data
- Overlaid data areas
- Overlaid register save area, causing incorrect reference to data.

10 and above Program checks associated with system-related interrupts.

Dealing with arithmetic exceptions

If the program check was due to an arithmetic error (interruption codes 7 through F), you need to find the operands used in the last instruction. Use the procedure described in section “Locating program data” on page 276 to locate the fields. You need to know a little about the type of arithmetic being done, so that you can tell if the operands are valid. The interrupt you received tells you what sort of arithmetic the system was doing (binary, packed decimal, or floating point), but you need to determine if that is what you had intended to do. You might need to consult a programming language manual if you have any queries about this.

When you have identified the operands, you need to decide where the problem is. Questions to consider include:

- Has the data been overlaid?
- Has the value been changed by faulty logic?
- Does the data type not match the operation type? For example, if you define the variable as being packed decimal and then you read in binary information, this causes a ‘data exception’ error.

Dealing with protection exceptions

With the storage protection facility, there are further situations in which a protection exception (interrupt code 4) may occur:

- An attempt is made to write to the CDSA, ECDSA, or ERDSA, when storage protection is active and the application is running in user key
- An attempt is made to write to the ERDSA or RDSA when PROTECT is specified for the RENTPGM system initialization parameter.

If transaction isolation (for which storage protection is a prerequisite) is enabled, additional situations can occur:

- A transaction, defined with ISOLATE(YES), is executing a USER key program and attempts to read or write to another transaction's USER key task-lifetime storage in the UDSA or EUDSA.
- A transaction, defined with ISOLATE(NO), is executing a USER key program and attempts to read or write to another transaction's USER key task-lifetime storage in the UDSA or EUDSA, but the second transaction is defined with ISOLATE(YES). (For a full description of the transaction isolation facility and its use, refer to the *CICS/ESA Resource Definition Guide*.)

If any of these events occurs, CICS abnormally terminates the transaction with abend code ASRA and issues message DFHSR0622 which identifies the DSA over which the program attempted to write. This information is in the TACB and is traced by exception trace point ID AP 0781. It is also useful to know the execution key of the program at the time of the protection exception and whether the program was executing in a subspace (CDSA, ECDSA, RDSA, ERDSA, UDSA, or EUDSA). This appears in the TACB, exception trace point ID AP 0781 and at the head of the formatted transaction dump.

If the command protection facility is enabled, a protection exception can occur if storage, passed to CICS as an output parameter through the EXEC interface, is not accessible for READ/WRITE by the program issuing the command. The program is passing to CICS storage that it cannot itself update, but it requires CICS to update the storage. The transaction terminates abnormally with abend code AEYD. CICS creates an exception trace entry AP 0779 and saves relevant data in the TACB that is formatted at the beginning of the transaction dump.

Note

Storage protection, transaction isolation, and command protection are facilities that add data integrity by highlighting application errors. In previous releases, such errors may not have been detected or may have appeared as CICS problems. The use of these facilities greatly reduces the number of abends that appear to be CICS problems.

It is still possible for CICS to abend when the problem is in the application. For example, command protection only checks output parameters and does not prevent the passing of fetch-protected storage as an input parameter to CICS. When CICS attempts to read such storage, an ASRA abend occurs.

Causes of protection exceptions

CICS storage protection is intended to prevent application programs erroneously overwriting CICS programs and control blocks. The occurrence of a protection exception in a new program running in a system with storage protection active probably indicates an error in the application program. However, when existing programs which need to be defined with EXECKEY(CICS) are first migrated to a system with storage protection active, protection exceptions may well occur.

Any application program causing a protection exception when defined with EXECKEY(USER) must be examined to determine why it is attempting to modify storage that it is not allowed to modify. Its definition should be changed to

EXECKEY(CICS) only if it is determined that the application program is legitimately accessing CICS key storage, and the exception is not the result of an application error.

Programs might also be incorrectly link-edited as reentrant, and, as a result, loaded by CICS into one of the read-only DSAs (RDSA, ERDSA). When such an incorrectly defined program attempts to modify itself, or another program tries to modify it, a protection exception occurs. The program should be checked to see whether it should be redefined as non-reentrant, or whether the program should be changed to be truly reentrant. The protection exception might indicate that the program uses poor programming techniques that could result in other problems if uncorrected.

Transaction isolation

Transaction isolation protects the data associated with a user transaction from being overwritten by EXECKEY(USER) programs invoked by other user transactions. If transaction isolation is active, the occurrence of a protection exception in a new transaction indicates a probable error in the transaction or program definition. An interdependency may exist between two or more transactions. In a system running without transaction isolation, a transaction can read or write to the task-lifetime storage of another transaction. The IBM CICS Transaction Affinities Utility MVS/ESA helps to identify potential dependencies. Ideally such interdependencies must be removed. If interdependencies cannot be removed, define all affected transactions with ISOLATE(NO). For further details about defining transactions, see the *CICS/ESA Resource Definition Guide*

When migrating from a CICS 3.3 system to a CICS 4.1 system, it may be helpful to alter all transaction definitions to include the ISOLATE(NO) option. Then gradually change the definitions to use ISOLATE(YES) as interdependencies are removed. This facilitates manageable migration to a system that takes full advantage of the transaction isolation facility.

Command protection

Command protection prevents CICS from updating storage if the storage address is passed as a command output parameter by a transaction that is not authorized to update that storage. The transaction terminates with abend code AEYD. The exception trace entry AP 0779 supplies details of the failing program and command. When migrating to a system with command protection enabled, EXEC commands that pass unauthorized storage are identified and can be corrected.

Possible causes of protection exceptions referencing CICS DSAs

The following is a summary of some of the causes of protection exceptions that can occur in user key programs:

- Issuing an MVS macro request. Most MVS macros and services are not supported in EXECKEY(USER) application programs. Use of unsupported macros and services may cause a failure if these macros or services attempt to reference MVS storage outside the CICS DSAs.
- Referencing storage obtained by an MVS GETMAIN request or another MVS macro. MVS storage obtained by these methods resides outside the CICS DSAs, and is therefore protected from user key programs.
- Using PL/I statements, COBOL verbs or compiler options that are not permitted in CICS application programs (see the *CICS/ESA Application Programming*

Guide for details of prohibited language statements and compiler options). For example, the use of dynamic link in OS/VS COBOL, CALL with the RES compiler option, or a verb such as INSPECT, may also cause MVS storage outside the CICS DSAs to be obtained or updated (such storage is protected from user-key programs).

In previous releases of CICS, these may have worked, or at least may not have caused the application to fail. However, the use of these statements and options can have other effects on the overall execution of the CICS system, and should be removed where possible.

- Modifying the CWA when CWAKEY=CICS is specified as a system initialization parameter. In a user key program, this is an invalid reference to storage allocated from the CDSA or ECDSA.
- Modifying the TCTUA when TCTUAKEY=CICS is specified as a system initialization parameter. In a user key program this is an invalid reference to storage allocated from the CDSA or ECDSA.
- Issuing EXEC CICS EXTRACT EXIT command and attempting to update an exit program's global work area. In a user key program this is an invalid reference to storage allocated from the CDSA or ECDSA.

  Note: If you are using CSP/AD, CSP/AE, or CSP/RS, you must ensure that the definitions for programs DCBINIT, DCBMODS, DCBRINIT and DCBNCOP specify EXECKEY(CICS). These are all examples of programs that modify global work areas set up by global user exit programs.
- If you are using DB2 and you use the DB2 message formatting routine DSNTIAR, which is link-edited with your application programs, you should apply the PTF for DB2 APAR PN12516, and relink-edit the applications using DSNTIAR so that they may run in user key. If the applications are not re-link-edited after this PTF is applied, they will have to run in CICS key. As a first step, until you have applied this PTF, you can define the applications which use DSNTIAR with EXECKEY(CICS).

Protection exceptions referencing the read-only DSAs

Protection exceptions occurring in programs resident in the ERDSA and RDSA are caused by the program not being truly reentrant. It might be that the program should not be defined as reentrant, or it might be that the program should be reentrant but is using poor coding techniques which should be corrected rather than making the program non-reentrant. For example:

- Using static variables or constants for fields which are set by CICS requests. For example, in assembler coding, if the LENGTH parameter for a retrieval operation such as EXEC CICS READQ TS is specified as a DC elsewhere in the program, a constant is set up in static storage. When CICS attempts to set the actual length into the data area, it causes a protection exception if the program is in the ERDSA or RDSA.

  In some cases, for example EXEC CICS READ DATASET INTO () LENGTH() ..., the LENGTH value specifies the maximum length that the application can accept, and is set by CICS to contain the actual length read on completion of the operation. Even if the program does not have RENT specified, using a variable in the program itself for this length could cause problems if the program is being executed concurrently for multiple users. The first transaction may execute correctly, resulting in the actual record length being set in the

LENGTH parameter, which is then used as the maximum length for the second transaction.

- Defining a table with the RENT attribute and then attempting to initialize or update the table during CICS execution. Such a table should not be defined as RENT.
- Defining BMS mapsets as RENT can cause a protection exception, if CICS attempts to modify the mapsets. In some cases, CICS needs to modify BMS mapsets during execution. Mapsets should not be link-edited with the RENT attribute. BMS mapsets should be loaded into CICS key storage (because they should not be modified by application programs) which means they must not be link-edited with the RENT attribute. (Partition sets are not modified by CICS and can be link-edited with the RENT attribute.)

Protection exceptions referencing the UDSA and EUDSA

In a system running with transaction isolation enabled, protection exceptions can occur in programs with EXECKEY(USER). Such an exception is caused by one transaction using a USER-key program to read or write to the USER key task-lifetime storage of another transaction. This may highlight a program error or an interdependency between two transactions. The IBM CICS Transaction Affinities Utility MVS/ESA can help to identify potential transaction interdependencies. Examples of transaction interdependencies are:

- A transaction may use EXEC CICS GETMAIN to obtain storage, and pass the address of the storage to other transactions. Access to this storage by one of these other transactions causes a protection exception if transaction isolation is enabled, unless both affected transactions are defined with ISOLATE(NO). Storage to be shared in this manner should be acquired by a GETMAIN with the SHARED option. This is preferable to defining the transactions with ISOLATE(NO).
- A transaction may attempt to post an ECB that exists in another transaction's task-lifetime storage. ECBs should be acquired by a GETMAIN from shared storage. Alternatively, the affected transactions should be defined with ISOLATE(NO).

Analyzing the problem further

You should now know the point in the program at which the abend occurred, and what the program was attempting to do.

- If your program uses or calls other programs or systems, examine the interface and the way you pass data to the program. Are you checking the returned information from the other system? Incorrect logic paths based on incorrect assumptions can give unpredictable results.
- Examine the flow of your program using tools like the Execution Diagnostic Facility (CEDF). Check the transient data and temporary storage queues with the CICS browse transaction (CEBR), and use the CICS command-level interpreter and syntax checker transactions (CECI and CECS). If necessary, insert additional statements into the program until you understand the flow.
- Look at any trace output you might have. If you have a "normal" trace output included in the documentation, compare the two for differences.

- Define the current environment, and try to isolate any changes in it since your program last worked. This can be difficult in large installations, because so many people interact with the systems and slight changes can affect things that seem unconnected.

Abends when CICS is using the DBCTL interface

If a transaction terminates abnormally while CICS is using DBCTL, you need to determine whether CICS or IMS was in control at the time of the abend. You can do this by examining the time stamps in the CICS and DBCTL traces. For guidance about this, see the *CICS/ESA CICS-IMS Database Control Guide*.

If tracing was off at the time of the failure, you can find an indicator in the task local work area for DFHDBAT. The indicator is set when CICS passes control to DBCTL, and reset when DBCTL returns control to CICS.

To find the indicator, locate the eye-catcher for the TIE in the dump and then locate the LOCLAREA eye-catcher that follows it. The indicator is at offset X'14' from the start of the LOCLAREA eye-catcher. If the indicator byte is set to X'08', CICS has passed control to DBCTL, and you should examine the IMS part of the transaction. If the byte is set to X'00', DBCTL has returned control to CICS, and you should investigate the CICS part of the transaction.

Worksheet for transaction abends

1. Record the abend code and messages
Find the abend code from the heading of the dump and record any pertinent messages.

_____

_____

2. Is this a CICS or a USER abend code?
For a CICS abend code, go to step 3.
If this is a USER abend code, tell the appropriate person.

3. Look up the abend code
If you need further advice, go to step 4.

4. Is this an AICA abend?
Yes; read Chapter 7, “Dealing with loops” on page 143.
No; go to step 5.

5. Is this an ASRA abend?
Yes; go to step 7.
No; go to step 6.

6. Is this an ASRD abend?
Yes; go to step 7.
No; go to step 14.

7. Record the program areas from the dump.

Find the program names from the Module Index at the end of the formatted dump.

Begin Address	End Address	Program Name

8. Record the address of the next instruction from the PSW, or the offset established by CICS.

Offset/address of next instruction	

9. Did the program check occur in one of the program areas listed above?
Yes; go to step 10.
No; go to step 14.

10. Record what type of program check occurred.

Program Interrupt Code	

11. Find the last statement executed.

Last Statement Executed

(See "Locating the last command or statement" on page 274.)

12. Was the PIC one of the arithmetic interrupts (7,8,9,A,B,C,D,E,F)?

No; go to step 15.
Yes; find the contents of the operands of the last instruction.

Contents

(See "Locating program data" on page 276.)

Now go to step 15.

13. Was the PIC a protection exception?

No; go to step 15.
Yes; see "Dealing with protection exceptions" on page 37 and go to step 15.

14. Find the last statement executed.

Last Statement Executed

(See "Locating the last command or statement" on page 274.)

15. Analyze the problem and the data gathered.

For most problems you should now have enough information to solve the problem. If you still cannot find the source, recheck the following:

1. Parameters to or from other programs or systems.
2. Any needed resource that may not be available.
3. The formatted trace, for any unexplained flow.
4. The running environment, for any changes in it.

FEPI abends

For information about FEPI-associated abends in CICS or MVS, refer to the *CICS/ESA Front End Programming Interface User's Guide*.

Chapter 5. Dealing with CICS system abends

The purpose of this chapter is to give guidance about gathering essential information about CICS system abends. You are likely to be reading it for any of these reasons:

- A message has alerted you that a CICS system abend has occurred.
- You have been directed here from the *CICS/ESA Messages and Codes* manual, or the symptom code given in a message has prompted you to look here.
- The symptom string at the head of a system dump gives insufficient information about the cause of a CICS system abend.

If you have not yet done so, use the CMAC transaction or look in the *CICS/ESA Messages and Codes* manual for an explanation of any message you may have received, because it could offer a straightforward solution to your problem.

If the abend was clearly caused by a storage violation, turn directly to Chapter 10, “Dealing with storage violations” on page 189. You know when CICS has detected a storage violation, because it issues this message:

DFHSM0102 *applid* **A storage violation (code X**'*code*'**) has been detected by module** *modname*.

On reading this chapter, you may find that the abend was due to an application error. In this case, you need to look at the application to find out why it caused the abend. However, if you find that a CICS module seems to be in error, you need to contact the IBM Support Center. Before doing so, you must gather this information:

- The name of the failing module, and the module level
- The offset within the module at which the failure occurred
- The instruction at that offset
- The abend type.

This chapter tells you how to find out all of these things.

The documentation you need

The primary documentation you need for investigating abends is the system dump, taken at the time the error occurred. This usually contains all the evidence needed to find the cause of the problem.

If system dumping is permitted for the dump code, and if system dumping has not otherwise been disabled, a system dump will have been taken when the error was detected. You can find out which dump relates to which message, because the time stamps and the dump IDs are the same.

If a system dump was not taken when the abend occurred, you need to find out why. Use the procedure described in “You did not get a dump when an abend occurred” on page 168, and follow the advice given there. When you are sure that dumping is enabled for the appropriate system dump code, you need to recreate the system abend.

You can use the interactive problem control system (IPCS) to process dumps and view them online. See “Formatting system dumps” on page 277 for guidance about processing dumps using IPCS VERBEXIT parameters. The kernel domain storage areas (formatting keyword KE) and the internal trace table (formatting keyword TR) are likely to be the most useful at the start of your investigation.

The formatted output for kernel domain (search for the eye-catcher `===KE`) contains summary information about the error. The internal trace table (eye-catcher `===TR`) contains the exception trace entry (if any) that was made at the time the error was detected.

Later, you might find that storage summaries for the application, transaction manager, program manager, dispatcher, and loader domains (formatting keywords AP, XM, PG, DS, and LD, respectively) are also useful. In each case, level-1 formatting is sufficient in the first instance.

You can format and print the dump offline. Details of how to do this are given in the *CICS/ESA Operations and Utilities Guide*.

You may need to copy the dump so that you can leave the system dump data set free for use, or so that you have a more permanent copy for problem reporting.

Whether you look at the dump online or offline, don’t purge it from the dump data set until you have either copied it or finished with it—you might need to format other areas later, or the same areas in more detail.

Interpreting the evidence

The first things to look at are any **messages** accompanying the abend, the **exception trace entry** in the internal trace table, and the **symptom string** at the start of the dump.

Looking at the messages

Any messages that accompany a CICS system abend can sometimes point directly to the cause of the failure. For every case, advice about how to react to a message is given in the *CICS/ESA Messages and Codes* manual.

Looking at the exception trace entry

When a CICS system abend occurs, an exception trace entry is made to the internal trace table and any other active trace destination. It does not matter whether you have tracing turned on or not—the trace entry is still made.

If the trace table contains more than one exception trace entry, it is likely that the last one is associated with the dump. However, this might not always be the case, and you should make sure that you have found the correct entry. Be aware, too, that dumps can sometimes be requested without a corresponding exception trace entry being made.

The exception trace entry gives information about what was happening when the failure occurred, and data that was being used at the time.

For details of trace entries, see Chapter 14, “Using traces in problem determination” on page 219.

Looking at the symptom string in the dump

The symptom string in a system dump is similar to the short symptom string at the beginning of a CICS transaction dump.

If your CICS system is running under MVS/ESA SP 4.1 or a later, upward-compatible release, the symptom string:

- Is written to SYS1.LOGREC
- Is issued as part of message DFHME0116
- Appears at the beginning of a CICS system dump.

If your CICS system is running under an earlier release of MVS/ESA, the symptom string:

- Is issued as part of message DFHME0116
- Appears at the beginning of a CICS system dump.

The symptom string provides a number of keywords that can be directly typed into RETAIN and used to search the RETAIN database. The possible keywords are shown in Table 1. The keywords are used at the IBM Support Center to discover duplicate problems, or problems that have already been reported by other users and for which a solution is available.

If you have the IBM INFORMATION/ACCESS licensed program, 5665-266, you can search the RETAIN database yourself.

If you report a problem to the IBM Support Center, you are often asked to quote the symptom string.

Table 1. Symptom string keywords

Keyword	**Meaning**
PIDS/	Product ID (CICS product number)
LVLS/	Level indicator (CICS release level)
RIDS/	Module name
PTFS/	Module PTF level
MS/	Message ID reporting error
AB/	Abend code
ADRS/	Address or offset indicator
PRCS/	Return code
OVS/	Overlaid storage
FLDS/	Name of a field associated with problem
REGS/	Software register associated with problem
VALU/	Value of a named field or register
# PCSS/	CICS jobname

Although the symptom string is designed to provide keywords for searching the RETAIN database, it can also give you significant information about what was happening at the time the error occurred, and it might suggest an obvious cause or a likely area in which to start your investigation. Amongst other things, it might

contain the abend code. If you haven't already done so, look in the *CICS/ESA Messages and Codes* manual to see what action it suggests for this abend code.

If the system is unable to gather much information about the error, the symptom string is less specific. In such cases, it might not help you much with problem determination, and you need to look at other parts of the dump. The kernel domain storage summary is a good place to start.

Looking at the kernel domain storage areas

The type of information that you can gather from the kernel domain storage areas is as follows:

- A summary of tasks and their status, and whether or not they were in error when the dump was taken.
- An error analysis report for each task currently in error.
- CICS retains information for the previous fifty errors.
- The linkage stack for each task, showing which programs have been called and have not yet returned.

The first thing you need to do is to find out which tasks are associated with the error.

Finding which tasks are associated with the error

You can find out which tasks are associated with the error from the kernel task summary. This tells you which tasks were in the system when the dump was taken, whether or not they were running, and whether they were in error.

The task summary is in the form of a table, each line in the table representing a different task.

The left-hand column of the task summary shows the kernel task number, which is the number used by the kernel domain to identify the task. This is not the same as the normal CICS task number taken from field TCAKCTTA of the TCA.

Figure 2 on page 49 shows an example of a kernel task summary with a task in error.

```
===KE: KERNEL DOMAIN KE_TASK SUMMARY
KE_NUM KE_TASK  STATUS       TCA_ADDR TRAN_# TRANSID DS_TASK  KE_KTCB  ERROR
0001   0397F930 KTCB STEP    00000000                00000000 039D0300
0002   0397FC20 KTCB QR      00000000                054C9000 039D2100
0003   0397FF10 KTCB RO      00000000                054CA000 039D1200
0004   03980200 NOT RUNNING  0007D5F0 00046  DLID    054D05C8 039D2100
0005   039804F0 NOT RUNNING  000805F0 00055  CEMT    054D0AA8 039D2100
0006   039807E0 UNUSED
0007   03980AD0 NOT RUNNING  03B86000 00058  CEDF    054D0428 039D2100
0008   03980DC0 NOT RUNNING  00077000 00038  ENA1    054D0288 039D2100
0009   039810B0 NOT RUNNING  00074000 00006  CSSY    054D01B8 039D2100
000A   039813A0 NOT RUNNING  0007D000 JBS    CSSY    054D09D8 039D2100
000B   03981690 NOT RUNNING  03B1B5F0 00004  CSNE    054D0908 039D2100
000C   03981980 NOT RUNNING  000775F0 J01    CSSY    054D0768 039D2100
000D   03981C70 NOT RUNNING  000745F0 00021  CSSY    054D0B78 039D2100
000E   03981F60 NOT RUNNING  0006A5F0 00005  CSSY    054D0838 039D2100
000F   03982250 NOT RUNNING  03B1B000 TCP    CSTP    054D0358 039D2100
0010   03982540 ***RUNNING** 00080000 00059  DELA    054D0DE8 039D2100 *YES*
0011   03982830 NOT RUNNING  0006A000 00036  ALL     054D0018 039D2100
0012   03982B20 UNUSED
0013   03982E10 NOT RUNNING  00000000                054D00E8 039D2100
0014   03983100 NOT RUNNING  00000000                054D0D18 039D2100
0015   039833F0 UNUSED
0016   039836E0 UNUSED
0017   039839D0 UNUSED
0018   03983CC0 UNUSED
0019   03983FB0 UNUSED
001A   039842A0 UNUSED
001B   03984590 UNUSED
```

Figure 2. Kernel task summary showing a task in error

When you have located the task summary table in the formatted dump, look in the ERROR column. If you find a value of *YES* for a particular task, that task was in error *at the time the dump was taken*.

Note: If the recovery routine that is invoked when the error occurs does not request a system dump, you will not see any tasks flagged in error. In such a case, the system dump is likely to have been requested by a program that is being executed lower down the linkage stack and that received an abnormal response following recovery. The program that received the error has gone from the stack, and so cannot be flagged. However, error data for the failing task was captured in the kernel domain error table (see “Finding more information about the error” on page 50). Error data is also captured in the error table even when no system dump is taken at all.

In Figure 2, you can see that kernel task number 0010 is shown to be in error.

Look next at the STATUS column. For each task you can see one of the following values:

- “***Running**”, meaning that the task was running when the system dump was taken. Most of the time, only one task is shown to be running. If more than one task is shown to be running, the different tasks are attached to separate TCBs.
- “Not Running”, meaning that the task is in the system but is currently not running. It may, for example, be suspended because it is waiting for some resource, or it may be ready to run but waiting for a TCB to become available.
- “KTCB”, referring to CICS control blocks corresponding to the CICS TCBs. These are treated as tasks in the kernel task summary.
- “Unused”, meaning either that the task was in the system but it has now terminated, or that there has not yet been a task in the system with the

corresponding task number. Earlier “Unused” tasks are likely to have run and terminated, and later ones are likely never to have represented actual tasks. It is most unlikely that you will ever need to distinguish between the two possibilities.

You are almost certain to find that the task shown to be in error has a status of “***Running**”, as in the example of Figure 2 on page 49. Such a task would have been running at the time the error was detected.

Tasks shown to be “Not Running” are less likely to be associated with the error, but it is possible that one of these could have been flagged with an error. If you find this to be so, the most likely explanation is that the task in error was attempting recovery when, for some reason, it was suspended.

Two of the columns in the kernel task summary are particularly important in solving problems that require the use of traces. They are the TRAN_# and KE_NUM columns. The TRAN_# column for a task can contain:

- A number that matches the task number in the corresponding trace
- “TCP” for the CICS terminal control task
- Other character entries for CICS system tasks (for example, a component identifier like “AP” for a CICS system task in the AP domain).

When you are working with trace output, you can use the number from the TRAN_# column to identify entries associated with a user task up to the point at which that task passes control to CICS. To identify the CICS processing associated with the user task, you need to use the entry in the KE_NUM column of the kernel task summary. This matches the KE_NUM shown in the full trace entries for the task, and enables you to distinguish the CICS processing associated with the task you are interested in from other CICS processing.

Finding more information about the error

More information about the failure is given in the summary information for the task in error. This is given after the kernel task summary. It gives you a storage report for the task, including registers and PSWs, and any data addressed by the registers. The PSW is the program status word that is used by the machine hardware to record the address of the current instruction being executed, the addressing mode, and other control information. An example of such a storage report is shown in Figure 3 on page 51, in this case for a program check.

Look first in the dump for this header, which introduces the error report for the task:

```
==KE: KE DOMAIN ERROR TABLE
```

Next, you will see the kernel error number for the task. Error numbers are assigned consecutively by the kernel, starting from 00000001. You might, for example, see this:

```
=KE: ERROR NUMBER:  00000001
```

The error number tells you the number of program checks and system abends that have occurred for this run of CICS. Not all of them have necessarily resulted in a system dump.

Some kernel error data follows. If you want to find the format of this data (and, in most cases, you will not need to), see the DFHKERRD section of the *CICS/ESA*

Diagnosis Handbook. The next thing of interest is the kernel's interpretation of what went wrong. This includes the error code, the error type, the name of the program that was running, and the offset within the program.

The error code gives you the system and user completion codes issued when the abend occurred.

The error type tells you whether the error was associated with, for example, a program check, a system abend, or an internal request for system recovery.

```
==KE: KE DOMAIN ERROR TABLE
=KE: ERROR NUMBER:  00000001
 KERRD 0397B950 KERNEL ERROR DATA
    0000  F0C3F461 C1D2C5C1 018400C4 000022EE  C4C6C8E3 E2D74040 04D3FC70 054D0B78  *0C4/AKEA.D.D....DFHTSP  .L...(..*    0397B950
(Data for offset 0020 to 0100 follows)                                  ..........E..........DM.;*    0397B970
   ERROR CODE:  0C4/AKEA    ERROR TYPE:  PROGRAM_CHECK    TIMESTAMP:  A4D9433F7D330600
   DATE (GMT)    :  25/08/93        TIME (GMT)   :  09:04:49.484592
   DATE (LOCAL)  :  25/08/93        TIME (LOCAL) :  09:04:49.484592
   KE_NUM:  0007    KE_TASK:  03980AD0    TCA_ADDR:  0006A000    DS_TASK:  054D0B78
 ERROR HAPPENED IN PROGRAM DFHTSP   AT OFFSET 22EE
 ERROR HAPPENED UNDER THE CICS RB.
 CICS REGISTERS AND PSW FOLLOW.
   PSW:  078D1000 84D41F5E   INSTRUCTION LENGTH:  4  INTERRUPT CODE:  04  EXCEPTION ADDRESS:  00000000
   EXECUTION KEY AT PROGRAM CHECK/ABEND: 8
   SPACE AT PROGRAM CHECK/ABEND: BASESPACE
   REGISTERS 0-15
    0000  F2000518 03AE25CC 00011280 00000002  FFFFFFFF 04D44DB3 84D41DB6 04D42DB5  *2....................M(.DM...M..*    0397B9A0
    0020  04D43DB4 03B2F140 039936A0 00000001  0006A000 8003CBB0 84D43A54 04D43A3E  *.M....1 .R.............DM...M..*    0397B9C0
   DATA AT PSW: 84D41F5E    MODULE: DFHTSP     OFFSET: 000022EE
    0000  5CC4C6C8 E3E2D740 4084D3FC E4F0F3F3  F0C91707 1122C7C2 F6D44040 40401400  **DFHTSP  DL.U0330I....GB6M    ..*    04D3FC70
(Data for offset 0020 to 2300 follows)                                  .......................*    04D3FC90
   DATA AT REGISTERS
   REG 0   F2000518
 31-BIT DATA CANNOT BE ACCESSED **
 24-BIT DATA FOLLOWS:
   -0080  00000000 00000000 00000007 00000000  00000000 00000000 00000000 00FFCF20  *.............................*    00000498
(Data for offset -0060 to 0100 follows)                                 .......................*    000004B8
(Similar data for CICS registers 1 to 15 follows)
```

Figure 3. Storage report for a task that has experienced a program check

Next, there is a report of where the system has recorded that the error occurred, and the circumstances of the failure. This is the general format of the information:

```
Error happened in program pppppppp at offset xxxxxxxx
Error happened ...
```

The program name (pppppppp) and offset (xxxxxxxx) are determined by searching through the CICS loader's control blocks for a program that owned the abending instruction at the time of the abend. If this search does not find such a program, the following text appears in the report:

```
PROGRAM QQQQQQQQ WAS IN CONTROL, BUT THE PSW WAS ELSEWHERE.
```

The program name (qqqqqqqq) reported, is the program that owns the current kernel stack entry for the abending task. If this text appears, it may be possible to locate the failing program using the method described in "Using the linkage stack to identify the failing module" on page 54.

The failing program name and offset are also displayed in the section of the report immediately after the contents of the registers have been reported. The format of this information is:

```
DATA AT PSW: AAAAAAAA   MODULE: PPPPPPPP   OFFSET: XXXXXXXX
```

If the failing program could not be located, the module name and offset are reported as unknown. The possible reasons for the program not being located are:

- The failure occurred in an MVS loaded module
- The failing program had been released by the CICS loader before the dump was taken
- A wild branch in the failing program caused the PSW to point to storage not occupied by a CICS loaded program.

Note that the accuracy of the program name and offset reported in a formatted dump that was produced as the result of a program executing a wild branch cannot be guaranteed.

After the kernel's interpretation of the error, you will see one of these diagnostic messages:

```
Error happened under the CICS RB
```

This means that the error was detected either when CICS code was executing, or when an access method called by CICS was running (for example, VSAM or QSAM). The CICS RB is the CICS request block, an MVS control block that records the state of the CICS program.

```
Error did not happen under the CICS RB
```

This message can be issued in any of these circumstances:

- An error occurs in CICS SVC code.
- An error occurs in a CICS VTAM exit.
- CICS detects a runaway task during the execution of an MVS service request.
- An error occurs during the execution of an SVC request that was made by CICS or an access method invoked by CICS.

After either of these messages, you next get some data that is likely to be related to the problem. The data you get depends on whether or not the error happened under the CICS RB.

The error data for the failing task

If the error happened under the CICS RB, the error data you get in the task storage report is based on values in the **PSW** and the **CICS registers** at the time the error was detected. Figure 3 on page 51 shows the storage report for a task that failed when a program check was detected. It illustrates the error data supplied when an error happens under the CICS RB.

If the error did not happen under the CICS RB, for example when CICS was calling an MVS service, you get data based on two sets of registers and PSWs. The registers and PSW of the CICS RB at the time of the error constitute one set. The registers and PSW of the RB in which the error occurred constitute the other set. This data will relate, very probably, to the execution of an SVC routine called by CICS. The error may have occurred, however, during an IRB interrupt or in an

SRB. You can confirm whether this has happened by checking flags KERNEL_ERROR_IRB and KERNEL_ERROR_SRB_MODE.

The storage addressed by the registers and PSW: Any storage addressed by the registers and PSW is included in the error data for the failing task.

Note that only the values of the registers and PSW, not the storage they address, are guaranteed to be as they were at the time of the error. The storage that is shown is a snapshot taken at the time the internal system dump request was issued. Data might have changed because, for example, a program check has been caused by an incorrect address in a register, or short lifetime storage is addressed by a register.

Also, in general, where error data is given for a series of errors, the older the error, the less likely it is that the storage is as it was at the time of the failure. The most recent error has the highest error number; it might not be the first error shown in the output.

The registers might point to data in the CICS region. If the values they hold can represent 24-bit addresses, you see the data around those addresses. Similarly, if their values can represent 31-bit addresses, you get the data around those addresses.

It could be that the contents of a register might represent both a 24-bit address and a 31-bit address. In that case, you get both sets of addressed data. (Note that a register might contain a 24-bit address with a higher order bit set, making it appear like a 31-bit address; or it could contain a genuine 31-bit address.)

If, for any reason, the register does not address any data, you see either of these messages:

```
24-bit data cannot be accessed
31-bit data cannot be accessed
```

This means that the addresses cannot be found in the system dump of the CICS region. Note that MVS keeps a record of how CICS uses storage, and any areas not used by CICS are considered to lie outside the CICS address space. Such areas are not dumped in an MVS SDUMP of the region.

It is also possible that the addresses were within the CICS region, but they were not included in the SDUMP. This is because MVS enables you to take SDUMPs selectively, for example “without LPA”. If this were to happen without your knowledge, you might think you had an addressing error when, in fact, the address was a valid one.

The format of the PSW is described in the *IBM Enterprise Systems Architecture/370 Principles of Operation*. The information in the PSW can help you to find the details needed by the IBM Support Center. You can find the address of the failing instruction, and hence its offset within the module, and also the abend type. You find the identity of the failing module itself by examining the kernel linkage stack, as described in “Using the linkage stack to identify the failing module” on page 54.

Using the linkage stack to identify the failing module

You can sometimes use the technique described in this section to gather the information that the IBM Support Center needs to resolve the CICS system abend. However, you should normally use the summary information presented in the formatted output for the kernel domain storage areas.

This method is only valid if the abend has occurred in a module or subroutine that has a kernel linkage stack entry. This is the case only where the module or subroutine has been invoked by one of these mechanisms:

- A kernel domain call
- A kernel subroutine call
- A call to an internal procedure identified to the kernel
- A LIFO call.

Routines that have been invoked by assembler language BALR instructions do not have kernel linkage stack entries. Having found which task was in error from the kernel's task summary (see "Finding which tasks are associated with the error" on page 48), you need next to find out which module was in error. The module name is one of the things you need to give the IBM Support Center when you report the problem to them.

Find the task number of the task in error from the KE_NUM column, and use this as an index into the linkage stack entries. These are shown in the dump after the task summary.

Figure 4 shows what a typical kernel linkage stack looks like.

```
KE_NUM @STACK   LEN  TYPE ADDRESS  LINK REG OFFS ERROR NAME
0031   0520A020 0120 Bot  84C00408 84C006D8 02D0       DFHKETA
0031   0520A140 01F0 Dom  84C0F078 84C0F18E 0116       DFHDSKE
0031   0520A330 0370 Dom  84CAA5A8 84CAACC2 071A       DFHXMTA
0031   0520A6A0 0330 Dom  84F25430 84F25CF6 08C6       DFHPGPG
                     Int     +00CC 84F254B6 0086       INITIAL_LINK
0031   0520A9D0 03C0 Dom  84F6C230 84E5DC40 0000       DFHAPLI1
                     Int     +0EEA 84F6C66E 043E       CICS_INTERFACE
0031   0520AD90 0108 Sub  0230B400 8230B8CA 04CA       DFHEIQSP
0031   0520AE98 0290 Sub  82136D90 82137178 03E8 *YES* DFHLDLD
       0520B128      Int     +08FC 82136F26 0196       LDLD_INQUIRE
       0520B128      Int     +128E 821376CE 093E       CURRENT_GET_NO_WAIT
0031   0520B128 0F70 Dom  84C6F8E0 84C72EA6 35C6       DFHMEME
                     Int     +2CB6 84C6FA4E 016E       SEND
                     Int     +1486 84C72684 2DA4       CONTINUE_SEND
                     Int     +350E 84C70DE4 1504       TAKE_A_DUMP_FOR_CALLER
0031   0520C098 03D0 Dom  84C52458 84C52F52 0AFA       DFHDUDU
                     Int     +08F4 84C5254A 00F2       SYSTEM_DUMP
                     Int     +1412 84C53212 0DBA       TAKE_SYSTEM_DUMP
```

Figure 4. Example of a kernel linkage stack showing a task in error

The TYPE column in the example can contain any of the following entries:

Bot This marks the first entry in the stack.

Dom This marks a stack entry caused by a domain call.

Sub This marks a stack entry caused by a subroutine.

Lifo This marks a stack entry caused by a LIFO module.

Int This marks a call to an internal procedure identified to the kernel.

A linkage stack for a task represents the sequence in which modules and subroutines have been called during execution of a task. It provides a valuable insight into the sequence of events up until the time of failure, and it also flags any program or subroutine that was executing when the error was detected.

The modules and subroutines are shown in the listing in the order in which they were invoked, so the first module you see is at the bottom of the stack, and the second module is next from bottom. You often see DFHKETA and DFHDSKE, respectively, in these two positions.

The last module or subroutine in the listing is at the top of the stack, and it represents the last call that was made before the dump was taken. Assuming that the system abend caused the dump to be taken, this is likely to be a routine associated with dump domain.

In the example shown, program DFHLDLD is shown to be in error. In this case, DFHLDLD is the module name that you would need to report to the IBM Support Center, together with the other information described in "Using the PSW to find the offset of the failing instruction."

Note: If module DFHAPLI is flagged in error, consider first whether an application is to blame for the failure. DFHAPLI is the Application Language Interface Program, and it is on the linkage stack whenever an application is being executed. If an application is the cause of the error, it is your responsibility to correct the problem.

Using the PSW to find the offset of the failing instruction

You can calculate the offset of the failing instruction from the PSW, although in practice you seldom need to because the offset is quoted in the storage report for the task. If you are not sure of the format of the PSW, or how to calculate the offset, refer to the *IBM Enterprise Systems Architecture/370 Principles of Operation* manual.

The Support Center also needs to know the instruction at the offset. Locate the address of the failing instruction in the dump, and find out what instruction is there. It is sufficient to give the hex code for the instruction, but make sure you quote as many bytes as you found from the PSW instruction length field.

Identify also the abend type from the program interruption code, so that you can report that, too. It might, for example, be 'protection exception' (interruption code 0004), or 'data exception' (interruption code 0007).

Finding the PTF level of the module in error

The IBM Support Center needs to know the PTF level of any module reported to them as being in error. You can find this in the loader domain program storage map summary, which you can get using the dump formatting keyword LD.

Figure 5 on page 56 shows some entries from a typical program storage map summary. It shows that program DFHZNEP, for example, is at PTF level ULnnnnn, where nnnnn is a five-digit number.

```
 ==LD: PROGRAM STORAGE MAP
  PGM NAME ENTRY PT  CSECT    LOAD PT. REL. PTF LVL. LAST COMPILED  COPY NO. USERS     LOCN  TYP ATTRIBUTE R/A MODE APE ADDR
                                                                                                OVERRIDE
  DFHDUIO  8003B000 DFHDUIO  0003B000 0410 ULnnnnn  06/26/94 19.32    1       1       CDSA  ANY REUSABLE  24   31  03AC6330
                    IPRDUIO  0003C358
  DFHCSA   8003CBB0 DFHKELCL 0003C440 0410 ULnnnnn  03/25/94 08.38    1       1       CDSA  RPL RESIDENT   -    -  03AC6430
1                   DFHKELRT 0003C740 0410 ULnnnnn  03/25/94 08.43
                    DFHCSA   0003C9C8 0410 ULnnnnn  I  01/11 10.08
                    DFHCSAOF 0003CF78 0410 ULnnnnn  I  01/11 10.08
                    DFHKERCD 0003D568 0410 ULnnnnn  10/22/93 18.48
                    DFHKERER 0003D738 0410 ULnnnnn  11/12/94 16.28
                    DFHKERRI 0003E0D8 0410 ULnnnnn  10/14/93 16.16
                    DFHKESFM 0003E308 0410 ULnnnnn  10/14/93 16.18
                    DFHKESGM 0003E820 0410 ULnnnnn  10/14/93 16.18
  DFHAIP   0003EE88 DFHEIP   0003EE60 0410 ULnnnnn  I  04/11 17.14    1       2       CDSA  ANY RESIDENT   -    -  03AC6730
                    DFHEIPA  000410A0 0410 ULnnnnn  I  16/10 15.45
                    DFHCPI   00041318 0410 ULnnnnn  I  05/10 18.56
  DFHZNEP  83ADF4C8 DFHYA330 03ADF480 0410 ULnnnnn                    1       0       ECDSA RPL RESIDENT   -    -  03B793B0
```

Figure 5. Part of the loader domain program storage map summary

Note: Entries made in the R/A MODE OVERRIDE columns are the value of the RMODE and AMODE supplied on the DEFINE_PROGRAM call for that program. If a REQUIRED_RMODE or REQUIRED_AMODE is not specified, a – (dash) symbol appears in the appropriate column. If AMODE_ANY or RMODE_ANY is specified, 'ANY' appears in the appropriate column. Other values are shown as specified.

Chapter 6. Dealing with waits

This chapter gives you information about what to do if you are aware that a task is in a wait state, or if CICS has stalled during:

- A run
- Initialization
- Termination
- Restart.

If CICS has stalled, turn directly to “What to do if CICS has stalled” on page 137.

If you have one or more tasks in a wait state, you should have already carried out preliminary checks to make sure that the problem is best classified as a wait, rather than as a loop or as poor performance. If you have not, you can find guidance about how to do this in Chapter 2, “Classifying the problem” on page 11.

You are unlikely to have direct evidence that a CICS system task is in a wait state, except from a detailed examination of trace. You are more likely to have noticed that one of your user tasks, or possibly a CICS user task—that is, an instance of a CICS-supplied transaction—is waiting. In such a case, it is possible that a waiting CICS system task could be the cause of the user task having to wait.

For the purpose of this chapter, a task is considered to be in a wait state if it has been suspended after first starting to run. The task is *not* in a wait state if it has been attached to the transaction manager but has not yet started to run, or if it has been resumed after waiting but cannot, for some reason, start running. These are best regarded as performance problems. Tasks that are ready to run but cannot be dispatched might, for example, have too low a priority, or the CICS system might be at the MXT limit, or the CICS system might be under stress (short on storage). If you think you might have such a problem, read Chapter 8, “Dealing with performance problems” on page 153.

Most tasks are suspended at least once during their execution, for example while they wait for file I/O to take place. This is part of the regular flow of control, and it gives other tasks a chance to run in the meantime. It is only when they stay suspended longer than they should that a problem arises.

There are two stages in resolving most wait problems involving user tasks. The first stage involves finding out what resource the suspended task is waiting for, and the second stage involves finding out why that resource is not available. This chapter focuses principally on the first of these objectives. However, in some cases there are suggestions of ways in which the constraints on resource availability can be relieved.

If you know that a CICS system task is in a wait state, it does not necessarily indicate an error in CICS. Some system tasks spend long periods in wait states, while they are waiting for work to do. For more information about waiting system tasks, see “CICS system task waits” on page 136.

Where to look next

- If you want information about the ways in which tasks in a CICS system can be made to wait, see “How tasks are made to wait” on page 65.
- If you want guidance about using online or offline techniques to investigate waits, see “Techniques for investigating waits.”
- If you already know the identity of the resource that a task is waiting for, but are not sure what functional area of CICS is involved, see Table 18 on page 76. It shows you where to look for further guidance.
- If you know the functional area with which the wait is associated, turn directly to the appropriate section later in this chapter:

  “Lock manager waits” on page 90
  “DL/I waits” on page 93
  “DBCTL waits” on page 96
  “Journal control waits” on page 98
  “Maximum task condition waits” on page 86
  “Task control waits” on page 101
  “Storage waits” on page 103
  “Temporary storage waits” on page 105
  “Terminal waits” on page 108
  “VTAM terminal control waits” on page 116
  “Interregion and intersystem communication waits” on page 118
  “Transient data waits” on page 118
  “Loader waits” on page 122
  “File control waits” on page 123
  “Interval control waits” on page 128
  “XRF alternate system waits” on page 134
  “CICS system task waits” on page 136
  “FEPI waits” on page 137.

Note: Throughout this chapter, the terms “suspension” and “resumption” and “suspended” and “resumed” are used generically. Except where otherwise indicated, they refer to any of the SUSPEND/RESUME and WAIT/POST processes by which tasks can be made to stop running and then be made ready to run again.

Techniques for investigating waits

You can investigate waits in a CICS system by online inquiry, by tracing, or by analysis of the formatted CICS system dump. The last two techniques are, to some extent, complementary.

Online inquiry is the least powerful technique, and it can only tell you what resource a suspended user task is waiting for. This is enough information to locate the failing area, but you often need to do more investigation before you can solve the problem. The advantage of online inquiry is that you can find out about the waiting task as soon as you detect the problem, and so you capture the data early.

Tracing can give you much more detail than online inquiry, but it involves significant processing overhead. It must also be running with the appropriate options selected when the task first enters a wait state, so this usually means you need to reproduce the problem. However, the information it gives you about system activity in the period leading up to the wait is likely to provide much of the information you need to solve the problem.

A CICS system dump can give you a picture of the state of the CICS system at an instant during the wait. You can request the dump as soon as you notice that a task has entered a wait state, so it gives you early data capture for the problem. However, the dump is unlikely to tell you anything about system activity in the period leading up to the wait, even if you had internal tracing running with the correct selectivity when the task entered the wait. This is because the trace table has probably wrapped before you have had a chance to respond. However, the formatted dump might contain much of the information you need to solve the problem.

If you are able to reproduce the problem, consider using auxiliary tracing and dumping in combination.

Investigating waits – online method

Online, you can use either CEMT INQ TASK or EXEC CICS INQUIRE TASK to find out what resource a user task is waiting on. EXEC CICS INQUIRE TASK can be executed under CECI, or issued from a transaction. Whatever online method you use, you need to supply the task ID of the suspended user task.

If the task *is* suspended, the information that is returned to you includes the resource type and/or the resource name identifying the unavailable resource.
CEMT INQ TASK displays the resource type of the unavailable resource in the
HTYPE field. The HVALUE field displays the resource name of the unavailable
resource. EXEC CICS INQUIRE TASK returns values in the SUSPENDTYPE and
SUSPENDVALUE fields which correspond to the resource type and resource name
of the unavailable resource.

HTYPE and SUSPENDTYPE, and HVALUE and SUSPENDVALUE correspond to
the values in the resource type and resource name fields of the dispatcher task
summary.

Table 18 on page 76 gives a list of all the resource types and resource names that user tasks might be suspended on, and references showing where to look next for guidance about solving the wait.

You probably need a system dump of the appropriate CICS region to investigate the wait. If you don't yet have one, you can get one using CEMT PERFORM SNAP or CEMT PERFORM DUMP—but make sure the task is still in a wait state when you take the dump. You subsequently need to format the dump using keywords for the given resource type. Advice on which keywords to use is given, where appropriate, in the individual sections.

Investigating waits – using trace

You can find detailed information about the suspension and resumption of tasks during a run of CICS by studying the trace table. Tracing must, of course, be running when the task in question is suspended or resumed, and the tracing options must be selected correctly.

When you look at the trace table, you can find trace entries relating to a particular task from the task numbers that the entries contain. Each is unique to a task so you can be sure that, for any run of CICS, trace entries having the same task number belong to the same task.

For general guidance about setting tracing options and interpreting trace entries, see Chapter 14, “Using traces in problem determination” on page 219.

Setting up trace for wait problems

Gate DSSR of dispatcher domain provides the major functions associated with the suspension and resumption of tasks. (See “How tasks are made to wait” on page 65.) The level-1 trace points **DS 0004** and **DS 0005** are produced on entry to, and exit from, the gate.

You need, therefore, to select tracing to capture the DS level-1 trace entries to investigate any wait problem. You need to capture trace entries for other components as well, when you know what functional areas are involved. The functional area invoking the task wait might, for example, be terminal control (TC), or file control (FC). Level-1 tracing is often enough for these components. However, there are cases, such as those relating to VSAM I/O errors where level-2 trace is needed to examine the RPL as it is passed to VSAM.

Next, you need to ensure that tracing is done for the task that has the wait problem. At first select special tracing for just that task, and disable tracing for all other tasks by setting the master system trace flag off. Subsequently, you can select special tracing for other tasks as well if it becomes clear that they are implicated in the wait.

Interpreting trace for wait problems

For new-style trace entries, which include those for point IDs DS 0004 and DS 0005, the function being traced is shown explicitly in the interpretation string. The functions in Table 4 on page 66 are all provided by gate DSSR, and you might see any of them traced by these two trace points. The functions that can cause a task to enter a wait state are identified in the table. Look out for these in particular in the trace entries for any waiting task you are investigating.

Each function has its own set of input and output parameters, and these, too, are shown in the interpretation strings of the formatted trace entries. Input parameters are shown in the trace entries made from point ID DS 0004, and output parameters in the trace entries made from point ID DS 0005.

The values of the parameters can provide valuable information about task waits, so pay particular attention to them when you study the trace table.

Investigating waits – the formatted CICS system dump

If you are suitably authorized, you can request a CICS system dump using CEMT PERFORM DUMP, CEMT PERFORM SNAP, or EXEC CICS PERFORM DUMP. Make sure that the task in question is waiting when you take the dump, so that you capture information relevant to the wait.

You need to use the dump formatting keyword DS to format the dispatcher task summary. You probably need to look at other areas of the dump as well, so keep the dump on the dump data set.

Interpreting the dump for wait problems

The dispatcher task summary gives you information like that shown in Figure 6.

```
===DS: DISPATCHER DOMAIN - SUMMARY


 KEY FOR SUMMARY

     TY = TYPE OF TASK            SY=SYSTEM  NS=NON-SYSTEM
     S  = STATE OF TASK           DIS=DISPATCHABLE  SUS=SUSPENDED
                                  RUN=RUNNING  REE=RESUMED EARLY
     P  = PURGEABLE WAIT/SUSPEND  Y=YES  N=NO
     PS = PURGE STATUS            OK=NO PURGE  PU=PURGED  PP=PURGE PENDING
     TT = TIMEOUT TYPE            IN=INTERVAL  DD=DEADLOCK DELAYED DI=DEADLOCK IMMEDIATE
     ST = SUSPEND TYPE            MVS=WAIT_MVS  SUSP=SUSPEND  OLDC=WAIT_OLDC  OLDW=WAIT_OLDW
     DTA= DISPATCHER TASK AREA
     AD = ATTACHING DOMAIN
     MO = TASK MODE               CO=CONCURRENT  QR=QUASI-REENTRANT RO=RESOURCE OWNING  RP=ONC RPC  SZ=FEPI OWNING


DS_TOKEN KE_TASK  TY S   P PS TT RESOURCE RESOURCE ST   TIME OF      TIMEOUT      DTA      AD ATTACHER MO SUSPAREA XM_TXN_TOKEN
                                 TYPE     NAME          SUSPEND      DUE          (DSTSK)     TOKEN

00000003 055C9C80 SY SUS N OK -                    SUSP 11:07:46.318 18      -    056FB080 XM 05706300 QR 056FB080 05706300000001
00020003 055D6C80 SY SUS N OK -  TIEXPIRY DS_NUDGE SUSP 11:08:20.291
000C0003 056F0C80 SY SUS N OK -  ICEXPIRY DFHAPTIX SUSP 11:08:20.291
00120005 055C9580 SY SUS N OK -  ICMIDNTE DFHAPTIM SUSP 11:05:01.563
001A0003 055C9200 SY SUS N OK -  TCP_NORM DFHZDSP  OLDW 11:08:20.465
```

Figure 6. Dispatcher task summary

A brief explanation of the summary information is given in the dump. A more detailed explanation is given in the section that follows.

Dispatcher task summary fields

Detailed descriptions of the fields in the dispatcher task summary are given in Table 2. Some of the fields relate to all tasks known to the dispatcher, and some (identified in the table) relate only to suspended tasks. Values are not provided in fields of the latter type for tasks that are not suspended.

Table 2 (Page 1 of 3). Descriptions of fields shown in the dispatcher task summary

Field	Description
AD	The 2-character domain index identifying the domain that attached the task to the dispatcher.
ATTACHER TOKEN	A token provided by the domain that attached the task. This token uniquely identifies the task to the attaching domain.
DS_TOKEN	A token given by the dispatcher to a domain that attaches a task. It identifies the attached task uniquely to the dispatcher.
DTA	An address used internally by the dispatcher.
KE_TASK	A value that uniquely identifies to the kernel a task that has been created.
MO	The dispatching mode for the task. The possible values are: CO – concurrent. QR – quasi-reentrant. RO – resource owning. RP – ONC RPC SZ – FEPI owning.

Table 2 (Page 2 of 3). Descriptions of fields shown in the dispatcher task summary	
Field	**Description**
P	Whether the suspend call issued by the suspending task specified PURGEABLE(YES) or PURGEABLE(NO). PURGEABLE(NO) inhibits deadlock time-out, CEMT SET TASK PURGE, and EXEC CICS SET TASK PURGE. The possible values are: Y (=YES) – the task is purgeable. N (=NO) – the task is not purgeable.
PS	The purge status of the task. The possible values are: OK – the task has not been purged, and there is no purge pending. PU – the task has been purged, either by the dispatcher or by the operator. PP – there is a purge pending on the task.
RESOURCE NAME *(suspended tasks only)*	The name of the resource that a suspended task is waiting for. A value is given only if RESOURCE_NAME has been included as an input parameter on the suspend call.
RESOURCE TYPE *(suspended tasks only)*	The type of the resource that the task is waiting for. A value is given only if RESOURCE_TYPE has been included as an input parameter on the suspend call.
S	The state of the task within the dispatcher. Possible values are: DIS – the task is dispatchable. It is ready to run, and it will be dispatched when a TCB becomes available. RUN – the task is running. SUS – the task has been suspended by any of the functions SUSPEND, WAIT_MVS, WAIT_OLDC, or WAIT_OLDW of gate DSSR. For an explanation of these functions, see “How tasks are made to wait” on page 65. REE – the task has been resumed early, possibly because a RESUME request has arrived before the corresponding SUSPEND request. (The SUSPEND/RESUME interface is asynchronous. For more details, see “Function SUSPEND of gate DSSR” on page 68.)
ST *(suspended tasks only)*	The type of function that was invoked to suspend a currently suspended task. Possible values include: MVS – function WAIT_MVS OLDC – function WAIT_OLDC OLDW – function WAIT_OLDW SUSP – function SUSPEND For a description of the functions, see “How tasks are made to wait” on page 65.

Table 2 (Page 3 of 3). Descriptions of fields shown in the dispatcher task summary	
Field	**Description**
SUSPAREA *(suspended tasks only)*	Either an address used internally by the dispatcher, or an ECB address, or an ECB list address. These are the cases: • Address used internally, if the task was suspended by a SUSPEND call. • ECB address or ECB list address, if the task was suspended by a WAIT_MVS or WAIT_OLDW call. • ECB address, if the task was suspended by a WAIT_OLDC call. Look at the value given in the ST column to see which one of these descriptions applies.
TIME OF SUSPEND *(suspended tasks only)*	The time when a currently suspended task was suspended. The format is hh:mm:ss.mmm (hours, minutes, seconds, milliseconds), GMT.
TIMEOUT DUE *(suspended tasks only)*	The time that a suspended task is due to timeout, if a timeout interval has been specified. A suspended task only times out if it is not resumed before this time arrives. The format is hh:mm:ss.mmm (hours, minutes, seconds, milliseconds).
TT *(suspended tasks only)*	The time-out type for the task. The possible values, where one is given, are: IN – a time-out interval has been specified for the task. DD – deadlock action is to be delayed when the time-out interval expires. DI – deadlock action is immediate when the time-out interval expires. See "INTERVAL and DEADLOCK_ACTION parameters" on page 73 for more details of time-out interval and deadlock action.
TY	Whether this is a system task or a non-system task. Possible values are: SY – this is a system task. NS – this is a non-system task. A non-system task can be either a user written transaction, or a CICS-supplied transaction.

Parameters and functions setting fields in the dispatcher task summary

Many of the values shown in the dispatcher task summary are provided directly by parameters included on calls to and from the dispatcher. If you are using trace, you can see the values of the parameters in the trace entries, and this can be useful for debugging. For details of how you can use trace to investigate waits, see "Investigating waits – using trace" on page 59.

Table 3 on page 64 shows the parameters that set task summary fields, the functions that use those parameters, and the domain gates that provide the functions. Task summary fields that are *not* set by parameters are also identified (by *none* in "Related parameter" column).

Table 3 (Page 1 of 2). Parameters and functions that set fields shown in the dispatcher task summary

Field	Related parameter	Function	Input or output	Gate
AD	DOMAIN_INDEX	INQUIRE_TASK GET_NEXT	IN OUT	DSBR
DTA	ATTACH_TOKEN	CREATE_TASK	IN	KEDS
DS_TOKEN	TASK_TOKEN	ATTACH CANCEL_TASK PURGE_INHIBIT_QUERY SET_PRIORITY TASK_REPLY GET_NEXT INQUIRE_TASK	OUT IN IN IN IN OUT OUT	DSAT DSBR
KE_TASK	TASK_TOKEN	CREATE_TASK CREATE_TCB PUSH_TASK TASK_REPLY TCB_REPLY	OUT OUT IN IN IN	KEDS
MO	MODE	ATTACH CHANGE_MODE GET_NEXT INQUIRE_TASK	IN IN OUT OUT	DSAT DSBR
P	PURGEABLE	SUSPEND WAIT_MVS WAIT_OLDC WAIT_OLDW	IN IN IN IN	DSSR
PS	*none*			
RESOURCE NAME	RESOURCE_NAME	ADD_SUSPEND SUSPEND WAIT_MVS WAIT_OLDC WAIT_OLDW GET_NEXT INQUIRE_TASK	IN IN IN IN IN OUT OUT	DSSR DSBR
RESOURCE TYPE	RESOURCE_TYPE	ADD_SUSPEND SUSPEND WAIT_MVS WAIT_OLDC WAIT_OLDW GET_NEXT INQUIRE_TASK	IN IN IN IN IN OUT OUT	DSSR DSBR
S *(see note 1)*	STATE	GET_NEXT INQUIRE_TASK	OUT OUT	DSBR
SUSPAREA *(see note 2)*	ECB_ADDRESS *or* ECB_LIST_ADDRESS *(see note 3)*	WAIT_MVS WAIT_OLDC WAIT_OLDW	IN IN IN	DSSR
TIME OF SUSPEND	*none*			
TASKNO	*none*			

Table 3 (Page 2 of 2). Parameters and functions that set fields shown in the dispatcher task summary				
Field	**Related parameter**	**Function**	**Input or output**	**Gate**
TIMEOUT DUE *(see note 4)*	*none*			
TT	INTERVAL *and* DEADLOCK_ACTION	SUSPEND WAIT_MVS WAIT_OLDW WAIT_OLDC	IN IN IN IN	DSSR
TY	*none*			
ATTACHER TOKEN	USER_TOKEN	ATTACH PURGE_INHIBIT_QUERY TASK_REPLY GET_NEXT INQUIRE_TASK	OUT OUT	DSAT DSBR
ST	*none*			

Notes:

1. Field S (for STATE) of the dispatcher task summary has a wider range of values than parameter STATE of DSBR functions GET_NEXT and INQUIRE_TASK. Parameter STATE can only have the values READY, RUNNING, or SUSPENDED. For the possible values of field S, see Table 2 on page 61.
2. Parameters ECB_ADDRESS and ECB_LIST_ADDRESS only relate to SUSPAREA when the task has been suspended by the WAIT_MVS, WAIT_OLDW, or WAIT_OLDC functions of gate DSSR.
3. Parameter ECB_LIST_ADDRESS is only valid for functions WAIT_MVS and WAIT_OLDW, and not for function WAIT_OLDC.
4. If INTERVAL has been specified, the value of TIMEOUT DUE should be equal to INTERVAL + TIME OF SUSPEND.

How tasks are made to wait

The suspension and resumption of tasks in a CICS system are performed by the dispatcher domain, usually on behalf of some other CICS component. If the exit programming interface (XPI) is being used, it can be at the request of user code.

The major functions associated with the suspension and subsequent resumption of tasks are provided by gate DSSR of dispatcher domain. When a task is to be suspended or resumed, the requesting component calls gate DSSR with the appropriate set of parameters. The required function is included in the parameter list sent on the call, and the corresponding routines are executed by the dispatcher. You can use trace to see the functions that are requested, and the values of parameters that are supplied. See “Investigating waits – using trace” on page 59.

Table 4 on page 66 lists the functions provided by the gate, and gives a brief summary of their effect on the status of tasks in the CICS system.

Table 4. Functions provided by dispatcher gate DSSR	
Function	**Effect on status of tasks**
ADD_SUSPEND	None – this does not cause a task to be suspended.
INQUIRE_SUSPEND_TOKEN	None.
DELETE_SUSPEND	None – this does not cause a task to be resumed.
SUSPEND	This can cause a running task to be suspended.
RESUME	This can cause a suspended task to be resumed.
WAIT_MVS	This can cause a running task to wait.
WAIT_OLDW	This can cause a running task to wait.
WAIT_OLDC	This can cause a running task to wait.

The functions that are significant for problem determination are now described in detail. It is necessary to consider their effects, the protocols that must be used, and the parameters they use. If you are using trace, your approach to the wait problem depends on the features of the function that caused the task to wait, so you must find out early in your investigation how the task was suspended.

Some of the functions are available to users through the exit programming interface (XPI). If you have any applications using these XPI functions, make sure that they follow the rules and protocols exactly. For programming information about the XPI, see the *CICS/ESA Customization Guide*.

Function ADD_SUSPEND of gate DSSR

The ADD_SUSPEND function of gate DSSR returns a SUSPEND_TOKEN to the calling component. The function is available to users through the exit programming interface. ADD_SUSPEND does not cause any running task to be suspended.

The ADD_SUSPEND function has only to be called once for any sequence of SUSPEND and RESUME calls for a particular task, because the same token can be used each time.

The input and output parameters for the ADD_SUSPEND function are described in Table 5 and Table 6, respectively.

Table 5. DSSR ADD_SUSPEND input parameters	
Parameter	**Description**
[RESOURCE_NAME] [RESOURCE_TYPE]	Provide default values for SUSPEND calls using this suspend token. The defaults are overridden if values are specified on the SUSPEND call. The parameters are optional, so in some cases you might not see them shown in the formatted DS 0004 and DS 0005 trace entries. A complete list of resource names and types is given in Table 18 on page 76.

Table 6. DSSR ADD_SUSPEND output parameters	
Parameter	**Description**
SUSPEND_TOKEN	The token used to identify a SUSPEND/RESUME pair.
RESPONSE	Possible values are: `OK|EXCEPTION|DISASTER|INVALID|KERNERROR` For the meanings of RESPONSE values, see Table 16 on page 73.
[REASON]	When RESPONSE is 'DISASTER', REASON has this value: `INSUFFICIENT_STORAGE`

Function INQUIRE_SUSPEND_TOKEN of gate DSSR

INQUIRE_SUSPEND_TOKEN is used by some CICS system tasks to find out the suspend token provided when the task was created.

The output parameters for the DSSR INQUIRE_SUSPEND_TOKEN function are described in Table 7. (There are no input parameters.)

Table 7. DSSR INQUIRE_SUSPEND_TOKEN output parameters	
Parameter	**Description**
SUSPEND_TOKEN	The token that was provided when the task was created.
RESPONSE	Possible values: `OK|DISASTER` For the meanings of RESPONSE values, see Table 16 on page 73.

Function DELETE_SUSPEND of gate DSSR

The DELETE_SUSPEND function of gate DSSR discards a SUSPEND_TOKEN. It does not cause any suspended task to be resumed. The function is available to users through the exit programming interface.

When the SUSPEND_TOKEN has been discarded, it can no longer be used to SUSPEND or RESUME a task. However, note that the dispatcher does not allow a SUSPEND_TOKEN to be discarded while a task is suspended against it, even if the RESUME request is serviced before the SUSPEND request. (This is an asynchronous interface—see "Function SUSPEND of gate DSSR" on page 68.)

The input and output parameters for the DSSR DELETE_SUSPEND function are described in Table 8 on page 68 and Table 9 on page 68, respectively.

Table 8. DSSR DELETE_SUSPEND input parameter

Parameter	Description
SUSPEND_TOKEN	The suspend token to be discarded.

Table 9. DSSR DELETE_SUSPEND output parameters

Parameter	Description
RESPONSE	Possible values are: `OK|EXCEPTION|DISASTER|INVALID|KERNERROR` For the meanings of RESPONSE values, see Table 16 on page 73.
[REASON]	When RESPONSE is 'INVALID', REASON can have either of these values: INVALID_SUSPEND_TOKEN – the dispatcher does not recognize the suspend token that has been supplied. SUSPEND_TOKEN_IN_USE – a task is already suspended against the suspend token that has been supplied.

Function SUSPEND of gate DSSR

The SUSPEND function of gate DSSR causes a running task to be suspended. The function is available to users through the exit programming interface, both explicitly (SUSPEND call) and implicitly (GETMAIN SUSPEND(YES) call).

The task to be suspended is identified by a SUSPEND_TOKEN, which might have been obtained by a previous ADD_SUSPEND call. A task suspended by a DSSR SUSPEND call is resumed by a corresponding DSSR RESUME call, if the task is not purged before the RESUME is issued.

It is important to recognize that this is an asynchronous interface: the RESUME call might have been received before the SUSPEND call. This can be significant if you are attempting to match SUSPEND and RESUME call pairs in the trace table, because you might find the RESUME trace entry was made before the corresponding SUSPEND trace entry. However, there can never be more than one outstanding SUSPEND or RESUME against any suspend token.

The input and output parameters for the DSSR SUSPEND function are described in Table 10 and Table 11 on page 69, respectively.

Table 10 (Page 1 of 2). DSSR SUSPEND input parameters

Parameter	Description
SUSPEND_TOKEN	The suspend token that the task is to be suspended against.

Table 10 (Page 2 of 2). DSSR SUSPEND input parameters	
Parameter	**Description**
PURGEABLE	The purgeable status of the task. Possible values are: YES|NO A task can be purged by the dispatcher when deadlock is detected, or purged by the operator or an application, if PURGEABLE is YES.
[INTERVAL] [DEADLOCK_ACTION]	INTERVAL and DEADLOCK_ACTION are mutually exclusive alternatives. For a discussion of their significance, see “INTERVAL and DEADLOCK_ACTION parameters” on page 73.
[RESOURCE_NAME] [RESOURCE_TYPE]	The name and the type of the resource that the task is to wait for. The parameters are optional, so in some cases you might not see them shown in the formatted DS 0004 and DS 0005 trace entries. A complete list of resource names and types is given in Table 18 on page 76.
[WLM_WAIT_TYPE]	The type of workload manager wait. Possible values are: LOCK|IO|CONV|SESS_LOCALMVS|SESS_SYSPLEX| SESS_NETWORK|TIMER|OTHER_PRODUCT|MISC For the meanings of WLM_WAIT_TYPE, see Table 17 on page 74.

Table 11 (Page 1 of 2). DSSR SUSPEND output parameters	
Parameter	**Description**
[COMPLETION_CODE]	A completion code to be supplied by the issuer of the RESUME.
RESPONSE	Possible values are: OK|EXCEPTION|DISASTER|INVALID|KERNERROR|PURGED For the meanings of RESPONSE values, see Table 16 on page 73.
[REASON]	When RESPONSE is ‘PURGED’, REASON can have either of these values: TASK_CANCELLED – the task was purged by the operator or an application while it was suspended. TIMED_OUT – the task was automatically resumed because the specified INTERVAL (or the deadlock time-out value specified at task attach) expired. When RESPONSE is ‘INVALID’, REASON can have any one of these values: INVALID_SUSPEND_TOKEN – the supplied suspend token was not recognized by the dispatcher. ALREADY_SUSPENDED – a SUSPEND call was issued against a suspend token that was already in use. CLEAN_UP_PENDING – the task has automatically been resumed because it has timed out, but resume processing is not yet complete.

Table 11 (Page 2 of 2). DSSR SUSPEND output parameters	
Parameter	**Description**
[DELAY]	If the ECB is posted within the time specified by the DELAY value, CICS is not to consider the task for dispatch until either the DELAY interval has expired or there is no other work to do.

Function RESUME of gate DSSR

The RESUME function of gate DSSR causes a suspended task to be resumed. The function is available to users through the exit programming interface.

The task to be resumed is identified by a SUSPEND_TOKEN, which is the same as the one used to SUSPEND the task in the first instance.

It is important to recognize that this is an asynchronous interface: so the RESUME call might have been received before the SUSPEND call. This can be significant if you are attempting to match SUSPEND and RESUME call pairs in the trace table, because you might find the RESUME trace entry was made before the corresponding SUSPEND trace entry. However, there can never be more than one outstanding SUSPEND or RESUME against any suspend token.

The input and output parameters for the DSSR RESUME function are described in Table 12 and Table 13, respectively.

Table 12. DSSR RESUME input parameters	
Parameter	**Description**
SUSPEND_TOKEN	The suspend token that the task was suspended against.
[COMPLETION_CODE]	A user-supplied completion code.

Table 13. DSSR RESUME output parameters	
Parameter	**Description**
RESPONSE	Possible values are: `OK|EXCEPTION|DISASTER|INVALID|KERNERROR|PURGED` For the meanings of RESPONSE values, see Table 16 on page 73.
[REASON]	When RESPONSE is 'EXCEPTION', REASON can have either of these values: TASK_CANCELLED – the task was purged by the operator or an application while it was suspended. TIMED_OUT – the task was automatically resumed because the specified INTERVAL (or the deadlock time-out value specified at task attach) expired. When RESPONSE is INVALID, REASON can have either of these values: INVALID_SUSPEND_TOKEN – the suspend token was not recognized. ALREADY_RESUMED – the task that was suspended against this suspend token has already been resumed.

Functions WAIT_MVS, WAIT_OLDW, and WAIT_OLDC of gate DSSR

Functions WAIT_MVS, WAIT_OLDW, and WAIT_OLDC of gate DSSR have much in common, and they are considered together. Unlike function SUSPEND of gate DSSR, no corresponding function is provided explicitly by gate DSSR to resume a task suspended by any of these functions. Instead, the task automatically becomes ready to run when an event control block (ECB) is posted. The input and output parameters for the functions are described in Table 14 and Table 15 on page 72, respectively.

WAIT_MVS

The WAIT_MVS function of gate DSSR causes a task to wait for an ECB, which might be in a list of ECBs, to be posted by the MVS POST macro. The function is available to users through the exit programming interface. A task made to wait by the WAIT_MVS function becomes eligible to run again when the single ECB, or any ECB in the list, has been posted.

WAIT_OLDW

The WAIT_OLDW function of gate DSSR causes a task to wait on a single ECB, or list of ECBs. The task becomes eligible to run again when the single ECB, or any ECB in the list, has been posted either by an MVS POST macro or by “hand posting”.

WAIT_OLDC

The WAIT_OLDC function of gate DSSR causes a task to wait on a single ECB that must be hand posted. A task made to wait by the WAIT_OLDC function becomes eligible to run again when the ECB has been posted.

Input and output parameters for WAIT_MVS, WAIT_OLDW, and WAIT_OLDC

Table 14 (Page 1 of 2). DSSR WAIT_MVS, WAIT_OLDW, and WAIT_OLDC input parameters

Parameter	Description
ECB_ADDRESS ECB_LIST_ADDRESS	These two parameters are mutually exclusive alternatives. ECB_ADDRESS is the address of a single ECB, and ECB_LIST_ADDRESS is the address of a list of addresses. ECB_LIST_ADDRESS is valid only for WAIT_MVS and WAIT_OLDW.
PURGEABLE	The purgeable status of the task. Possible values are: `YES|NO` A task can be purged by the dispatcher when deadlock is detected, or purged by the operator or an application, if PURGEABLE is YES.
[INTERVAL] [DEADLOCK_ACTION]	INTERVAL and DEADLOCK_ACTION are mutually exclusive alternatives. For a discussion of their significance, see “INTERVAL and DEADLOCK_ACTION parameters” on page 73.

Table 14 (Page 2 of 2). DSSR WAIT_MVS, WAIT_OLDW, and WAIT_OLDC input parameters	
Parameter	**Description**
[RESOURCE_NAME] [RESOURCE_TYPE]	The name and the type of the resource that the task is to wait for. The parameters are optional, so in some cases you might not see them shown in the formatted DS 0004 and DS 0005 trace entries. A complete list of resource names and types is given in Table 18 on page 76.
[BATCH] (WAIT_MVS only)	Whether the requests can be batched. Possible values are: YES|NO
[SPECIAL_TYPE] (WAIT_OLDW only)	Identifies the task as a special type. Its value is always CSTP.
[TIME_UNIT]	The time units used in specifying the INTERVAL value. Can be either seconds or milliseconds.
[DELAY]	If the ECB is posted within the time specified by the DELAY value, CICS is not to consider the task for dispatch until either the DELAY interval has expired or there is no other work to do.
[WLM_WAIT_TYPE]	The type of workload manager wait. Possible values are: LOCK|IO|CONV|SESS_LOCALMVS|SESS_SYSPLEX| SESS_NETWORK|TIMER|OTHER_PRODUCT|MISC For the meanings of WLM_WAIT_TYPE, see Table 17 on page 74.

Table 15. DSSR WAIT_MVS, WAIT_OLDW, and WAIT_OLDC output parameters	
Parameter	**Description**
RESPONSE	Possible values are: OK|EXCEPTION|DISASTER|INVALID|KERNERROR|PURGED For the meanings of RESPONSE values, see Table 16 on page 73.
[REASON]	When RESPONSE is 'PURGED', REASON can have either of these values: • TASK_CANCELLED – the task was purged by the operator or an application while it was waiting for the ECB to be posted. • TIMED_OUT – either the INTERVAL or the deadlock time-out interval expired. In case of the deadlock time-out, PURGEABLE(YES) must be in effect. When RESPONSE is 'INVALID', REASON can have either of these values: • ALREADY_WAITING – the single ECB specified in ECB_ADDRESS was already being waited on when this WAIT request was received. • INVALID_ECB_ADDR – the ECB address that was supplied is not valid.

INTERVAL and DEADLOCK_ACTION parameters

INTERVAL and DEADLOCK_ACTION are mutually exclusive input parameters on the SUSPEND and WAIT calls to gate DSSR of the dispatcher domain.

INTERVAL is a length of time after which a task is to be resumed automatically by the dispatcher, with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TIMED_OUT'.

DEADLOCK_ACTION describes whether or not the suspended task is to be purged if deadlock is detected, and if so, how it should be purged. The action to be taken depends on the anticipated cause of the deadlock, and it can have any one of these values:

- DELAYED. CICS ensures that it does not deadlock purge more than one task specifying DELAYED in a given interval. This is in the hope that purging one task will free the resources required by the others.
- IMMEDIATE. Deadlock purges of tasks specifying IMMEDIATE go ahead regardless of how many tasks are purged in a given interval. This is used, for example, when purging one task will have no effect on other tasks specifying IMMEDIATE.
- INHIBIT. This specifies that there is to be no deadlock time-out. INHIBIT overrides any deadlock time-out interval specified at task attach.

The meanings of RESPONSE values returned on DSSR calls

Table 16 explains the meaning of each of the RESPONSE values that can be returned from calls to DSSR functions.

Table 16. Meanings of DSSR function RESPONSE values

RESPONSE	**Meaning**
OK	When a domain command succeeds, a response of OK is given and the REASON code is not set. The requested function has been completed successfully.
EXCEPTION	Processing of the function could not be completed for the reason specified in the REASON field. If a command fails, the reason for the failure is returned together with an exception response.
DISASTER	The domain could not complete the request because of an unrecoverable system problem.
INVALID	The requested function is not supported by the domain, possibly because the format of the supplied parameters is incorrect.
KERNERROR	The kernel was unable to call the required function gate for the specified REASON.
PURGED	A purge has been requested for the task making the domain call.

The meanings of the WLM_WAIT_TYPE parameter

Table 17 explains the meaning of each of the values that can be specified on the WLM_WAIT_TYPE parameter on DSSR SUSPEND, WAIT_MVS, WAIT_OLDW, and WAIT_OLDC functions. See the *MVS/ESA Workload Management Services* manual, GC28-1494-01 for further information about this parameter.

Table 17. Meanings of WLM_WAIT_TYPE parameter values

WLM_WAIT_TYPE	Meaning
LOCK	Waiting on a lock. For example, waiting for: • A lock on CICS resource • A record lock on a recoverable VSAM file • A record lock in a BDAM file • An application resource that has been locked by an EXEC CICS ENQ command.
IO	Waiting for an I/O request or I/O related request to complete. For example: • File control, transient data, temproary storage, or journal I/O. • Waiting on I/O buffers or VSAM strings.
CONV	Waiting on a conversation between work manager subsystems.
SESS_LOCALMVS	Waiting on the establishment of a session with another CICS region in the same MVS image in the sysplex.
SESS_SYSPLEX	Waiting on establishment of a session with another CICS region in a different MVS image in the sysplex.
SESS_NETWORK	Waiting on the establishment of an ISC session with another CICS region (which may, or may not, be in the same MVS image).
TIMER	Waiting for a timer event or an interval control event to complete. For example, an application has issued an EXEC CICS DELAY or EXEC CICS WAIT EVENT command which has yet to complete.
OTHER_PRODUCT	Waiting on another product to complete its function; for example, when the work request has been passed to a DB2 or DBCTL subsystem.
MISC	Waiting on a resource that does not fall into any of the other categories.

Notes:

1. The MVS workload manager monitoring environment is set to STATE=IDLE when either:
   - A conversational task is waiting for terminal input from its principal facility, or
   - A CICS system task is waiting for work.
2. If the task is waiting on resource type ALLOCATE, the current MVS workload manager monitoring environment is set to STATE=WAITING and either:
   - RESOURCE=SESS_LOCALMVS if the session being waited on is a session with another CICS region in the same local MVS image.

- RESOURCE=SESS_SYSPLEX if the session being waited on is a session with a CICS region in another MVS image in the same sysplex.
- RESOURCE=SESS_NETWORK if the session being waited on is an ISC session which may, or may not, be in the same MVS image.

3. If the task is waiting on resource type IRLINK, the current MVS workload manager monitoring environment is set to STATE=WAITING, RESOURCE=CONV. Look at the RMF workload activity report to see whether the task continued beyond the current MVS monitoring environment. The SWITCHED column in this report can contain the following values:
   - LOCALMVS - the communicating CICS region is on the same local MVS image.
   - SYSPLEX - the communicating CICS region is on another MVS image in the same sysplex.
4. If the task is waiting on resource type ZCIOWAIT, the current MVS workload manager monitoring environment is set to either:
   - STATE=IDLE for a conversational task, or DTP transaction, that is awaiting input from its principal facility.
   - STATE=WAITING,RESOURCE=CONV for a task awaiting input from its alternate facility. Look at the RMF workload activity report to see whether the task continued beyond the current MVS monitoring environment. The SWITCHED column in this report can contain the following values:
     - LOCALMVS - the communicating CICS region is on the same local MVS image.
     - SYSPLEX - the communicating CICS region is on another MVS image in the same sysplex.
     - NETWORK - the communicating CICS region is in the VTAM network, which may, or may not be, in the same MVS image.

The resources that tasks in a CICS system can wait on

Table 18 on page 76 shows all the resources that tasks in a CICS system can wait on. Some resources are identified by both a resource name and a resource type, some by a resource name alone, and some by a resource type alone. The resource names and resource types that are shown are the ones that you can see in formatted trace entries and, for some resources, by online inquiry.

User tasks can be made to wait only on some of the resources. For each such resource, you will see a page reference showing you where to look for guidance about dealing with the wait. The values in the column 'Purge status' indicate whether the suspending module permits normal task purging and deadlock timeout at each suspend point. Normal task purging is permitted if there is a 'Y' (standing for 'yes') in the first column. If normal task purging is not permitted, there is an 'N' (standing for 'no') in this column. Deadlock timeout is permitted if there is a 'Y' (standing for 'yes') in the second column. If deadlock timeout is not permitted, there is an 'N' in this column.

The remaining resources are used only by CICS system tasks. If you have evidence that a system task is waiting on such a resource, and it is adversely affecting the operation of your system, you probably need to contact your IBM

Support Center. Before doing so, however, read “CICS system task waits” on page 136.

Table 18 (Page 2 of 10). Resources that a suspended task might be waiting on

Resource type Purge status	Resource name	Suspending module	DSSR call WLM wait type	Task	Where to look next
(none)	(none)	DFHDUIO	WAIT_MVS IO	System only	"CICS system task waits" on page 136
(none)	DLCNTRL	DFHDBCT	WAIT_MVS *See 1 on page 74*	System only	"CICS system task waits" on page 136
(none)	DLCONECT	DFHDBCON	WAIT_MVS OTHER_ PRODUCT	System only	"CICS system task waits" on page 136
(none)	DMWTQUEU	DFHDMWQ	SUSPEND MISC	System only	"CICS system task waits" on page 136
(none) N N	LMQUEUE	DFHLMLM	SUSPEND LOCK	User	"Lock manager waits" on page 90
ADAPTER N N	FEPI_RQE	DFHSZATR	WAIT_MVS MISC	User	*See note 6 on page 85*
ALLOCATE Y Y	TCTTETI value	DFHALP	SUSPEND *See 2 on page 74*	User	"Interregion and intersystem communication waits" on page 118
Any_MBCB N N	transient data queue name	DFHTDSUB	SUSPEND IO	User	"Transient data waits" on page 118
Any_MRCB N N	transient data queue name	DFHTDSUB	SUSPEND IO	User	"Transient data waits" on page 118
AP_INIT	CSADLECB	DFHSII1	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
AP_INIT	ECBTCP	DFHAPSIP	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
AP_INIT	SIPDMTEC	DFHAPSIP	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
AP_INIT	TCTVCECB	DFHSII1	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
AP_INIT	ZGRPECB	DFHSII1	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
AP_QUIES	CSASSI2	DFHSTP	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
AP_QUIES	SHUTECB	DFHSTP	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
AP_TERM	STP_DONE	DFHAPDM	WAIT_MVS LOCK	System only	"CICS system task waits" on page 136
CCSTWAIT	VSMSTRNG	DFHCCCC	WAIT_MVS IO	System only	"CICS system task waits" on page 136
CCVSAMWT	ASYNRESP	DFHCCCC	WAIT_MVS IO	System only	"CICS system task waits" on page 136

Table 18 (Page 3 of 10). Resources that a suspended task might be waiting on

Resource type Purge status		Resource name	Suspending module	DSSR call WLM wait type	Task	Where to look next
CCVSAMWT		EXCLOGER	DFHCCCC	WAIT_MVS IO	System only	"CICS system task waits" on page 136
CDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
CSMI		IRC	DFHMIRS	SUSPEND	System only	"CICS system task waits" on page 136
CSNC		MROQUEUE	DFHCRNP	WAIT_MVS *See 1 on page 74*	System only	"CICS system task waits" on page 136
DBCTL		DLSUSPND	DFHDBSPX	WAIT_MVS OTHER_ PRODUCT	User	"DBCTL waits" on page 96
N	N					
DBDXEOT		(none)	DFHDXSTM	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
DBDXINT		(none)	DFHXSTM	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
DFHAIIN		AITM	DFHAIIN1	SUSPEND MISC	System only	"CICS system task waits" on page 136
DFHCPIN		CPI	DFHCPIN1	SUSPEND MISC	System only	"CICS system task waits" on page 136
DFHPRIN		PRM	DFHPRIN1	SUSPEND MISC	System only	"CICS system task waits" on page 136
DFHSIPLT		EARLYPLT	DFHSII1	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
DFHSIPLT		LATE_PLT	DFHSIJ1	WAIT_MVS MISC	System only	"CICS system task waits" on page 136
DLI		IWAIT	DFHDLR	WAIT_MVS OTHER_ PRODUCT	User	"DL/I waits" on page 93
Y	Y					
ECDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
EDF		DBUGUSER	DFHEDFX	SUSPEND MISC	User	"EDF waits" on page 98
Y	N					
EKCWAIT		SINGLE	DFHEKC	WAIT_OLDW MISC	User	"Task control waits" on page 101
N	N					
ERDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
ESDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
EUDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					

Table 18 (Page 4 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status		Resource name	Suspending module	DSSR call / WLM wait type	Task	Where to look next
FCBFWAIT		file ID	DFHFCVR	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FCDWWAIT		file ID	DFHFCVR	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FCFSWAIT		file ID	DFHFCFS	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FCINWAIT		STATIC	DFHFCIN1	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
FCIOWAIT		file ID	DFHFCBD DFHFCVR	WAIT_MVS IO WAIT_MVS IO	User	"File control waits" on page 123
N	N					
FCPSWAIT		file ID	DFHFCVR	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FCRBWAIT		file ID	DFHFCVR	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FCSRSUSP		file ID	DFHFCVR	SUSPEND IO	User	"File control waits" on page 123
Y	Y					
FCTISUSP		file ID	DFHFCVR	SUSPEND IO	User	"File control waits" on page 123
Y	Y					
FCXCWAIT		file ID	DFHFCVR	WAIT_OLDC IO	User	"File control waits" on page 123
Y	Y					
FEPRM		SZRDP	DFHSZRDP	WAIT_MVS MISC	CSZI	*See note 6 on page 85*
N	N					
FOREVER		DFHXMTA	DFHXMTA	WAIT_MVS MISC	User	"A user task is waiting on resource type FOREVER" on page 90
N	N					
ICEXPIRY		DFHAPTIX	DFHAPTIX	SUSPEND TIMER	System only	"CICS system task waits" on page 136
ICGTWAIT		terminal ID	DFHICP	SUSPEND MISC	User	"Interval control waits" on page 128
Y	Y					
ICMIDNTE		DFHAPTIM	DFHAPTIM	SUSPEND TIMER	System only	"CICS system task waits" on page 136
ICWAIT		terminal ID *(See note 1 on page 85)*	DFHICP	SUSPEND MISC	User	"Interval control waits" on page 128
Y	N					
IRLINK		SYSIDNT concatenated with session name	DFHZIS2	WAIT_MVS *See 3 on page 75*	User	"Terminal waits" on page 108
Y	N					

Table 18 (Page 5 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status	Resource name	Suspending module	DSSR call / WLM wait type	Task	Where to look next
JASUBTAS N N	JASTMECB	DFHJCSDJ	WAIT_OLDC MISC	User	"Journal control waits" on page 98
JCAVLECB N N	LECBECB	DFHJCP	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCBUFFER N N	AVAIL_nn *(See note 2 on page 85)*	DFHJCP	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCBUFFER	JCTBAECB	DFHJCKOJ	WAIT_OLDC IO	System only	"CICS system task waits" on page 136
JCBUFFER N N	JCTBAECB	DFHJCSDJ	WAIT_OLDC MISC	User	"Journal control waits" on page 98
JCCLDONE N N	SUBTASK	DFHJCC	WAIT_OLDW IO	User	"Journal control waits" on page 98
JCDETACH N N	SUBTASK	DFHJCSDJ	WAIT_OLDC MISC	User	"Journal control waits" on page 98
JCFLBUFF N N	FLUSH_nn *(See note 2 on page 85)*	DFHJCP	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCINITN	JOURNALS	DFHJCP	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
JCIOBLOK N N	Jnnbbbbb *(See notes 2 on page 85 and 3 on page 85)*	DFHJCP	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCIOCOMP	JCTICA	DFHJCI DFHJCP	WAIT_MVS IO WAIT_MVS IO	System only	"CICS system task waits" on page 136
JCIOCOMP N N	JCTIOECB	DFHJCC	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCIOCOMP	RDBLOKnn *(See note 2 on page 85)*	DFHJCO	WAIT_MVS IO	System only	"CICS system task waits" on page 136
JCIOCOMP	RDRECDnn *(See note 2 on page 85)*	DFHJCO	WAIT_MVS IO	System only	"CICS system task waits" on page 136
JCJAGET N N	JABVECB	DFHJAP	WAIT_MVS IO	User	"Journal control waits" on page 98
JCJAIOWT	JCLDECB	DFHJASP	WAIT_MVS IO	System only	"CICS system task waits" on page 136
JCJAPUT N N	JABVECB	DFHJAP	WAIT_MVS IO	User	"Journal control waits" on page 98

Table 18 (Page 6 of 10). Resources that a suspended task might be waiting on

Resource type Purge status	Resource name	Suspending module	DSSR call WLM wait type	Task	Where to look next
JCJASUS	JABSUTOK	DFHJASP	SUSPEND *See 1 on page 74*	System only	"CICS system task waits" on page 136
JCJOURDS	data set name	DFHJCP	SUSPEND *See 1 on page 74*	System only	"CICS system task waits" on page 136
JCLASTBK N N	Jnnbbbbb *(See notes 2 on page 85 and 3 on page 85)*	DFHJCEOV	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCOPDONE N N	SUBTASK	DFHJCO	WAIT_OLDW IO	User	"Journal control waits" on page 98
JCREADY N N	JCTXAECB	DFHJCO	WAIT_MVS MISC	User	"Journal control waits" on page 98
JCREADY N N	JCTXBECB	DFHJCO	WAIT_MVS MISC	User	"Journal control waits" on page 98
JCREADY N N	JCTXXECB	DFHJCO	WAIT_MVS MISC	User	"Journal control waits" on page 98
JCRQDONE N N	SUBTASK	DFHJCC DFHJCKOJ DFHJCO	WAIT_OLDW IO WAIT_OLDW IO WAIT_OLDW IO	User System only User	"Journal control waits" on page 98
JCSWITCH	DS_SW_nn *(See note 2 on page 85)*	DFHJCP	WAIT_OLDC IO	System only	"CICS system task waits" on page 136
JCSWITCH N N	DS_SDJnn *(See note 2 on page 85)*	DFHJCSDJ	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCTAPE2 N N	PREOPNnn *(See note 2 on page 85)*	DFHJCEOV	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCTAPE2 N N	STABLEnn *(See note 2 on page 85)*	DFHJCSDJ	WAIT_OLDC IO	User	"Journal control waits" on page 98
JCTERMN	SUBTASK	DFHJCBSP	WAIT_OLDW *See 1 on page 74*	System only	"CICS system task waits" on page 136
JCTIMER	DFHJ1Snn *(See note 2 on page 85)*	DFHJCP	WAIT_OLDC TIMER	System only	"CICS system task waits" on page 136
KCCOMPAT N N	CICS	DFHXCPA	WAIT_OLDC LOCK	User	"Task control waits" on page 101

Table 18 (Page 7 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status		Resource name	Suspending module	DSSR call / WLM wait type	Task	Where to look next
KCCOMPAT		LIST	DFHXCPA	WAIT_OLDW MISC	User	"Task control waits" on page 101
N	N					
KCCOMPAT		SINGLE	DFHXCPA	WAIT_OLDW MISC	User	"Task control waits" on page 101
N	N					
KCCOMPAT		SUSPEND	DFHXCPA	SUSPEND MISC	User	"Task control waits" on page 101
Y	N					
KCCOMPAT		TERMINAL	DFHXCPA	SUSPEND MISC	User	"Task control waits" on page 101, and "Terminal waits" on page 108
Y	N					
KC_ENQ		SUSPEND	DFHXCPC	SUSPEND LOCK	User	"Task control waits" on page 101, "Transient data waits" on page 118, and "File control waits" on page 123
Y	Y					
MBCB_xxx *(See note 4 on page 85)*		transient data queue name	DFHTDSUB	SUSPEND IO	User	"Transient data waits" on page 118
N	N					
MRCB_xxx *(See note 4 on page 85)*		transient data queue name	DFHTDSUB	WAIT_MVS IO	User	"Transient data waits" on page 118
N	N					
MXT		XM_HELD	DFHXMAT	*(See note 5 on page 85)*	User	"Maximum task condition waits" on page 86
N	N					
PROGRAM		program ID	DFHLDLD	SUSPEND LOCK	User	"Loader waits" on page 122
Y	Y					
RDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
SDSA		(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103
Y	Y					
SMSYSTEM		(none)	DFHSMSY	SUSPEND TIMER	System only	"CICS system task waits" on page 136
STARTUP		TSMCPECB	DFHRCRP	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
SUBTASK		SISUBECB	DFHRCRP	WAIT_OLDC MISC	System only	"CICS system task waits" on page 136
SUCNSOLE		WTO	DFHSUWT	WAIT_MVS MISC	System only	"CICS system task waits" on page 136

Table 18 (Page 8 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status	Resource name	Suspending module	DSSR call / WLM wait type	Task	Where to look next
TCLASS N N	tclass name	DFHXMAT	*(See note 7 on page 85)*	User	"Transaction manager waits" on page 85
TCP_NORM	DFHZDSP	DFHZDSP	WAIT_OLDW *See 1 on page 74*	System only	"CICS system task waits" on page 136
TCP_SHUT	DFHZDSP	DFHZDSP	WAIT_OLDW MISC	System only	"CICS system task waits" on page 136
TDEPLOCK N N	transient data queue name	DFHTDEXP	SUSPEND LOCK	User	"Transient data waits" on page 118
TD_INIT N N	DCT	DFHTDA	SUSPEND MISC	User	"Transient data waits" on page 118
TDIPLOCK N N	transient data queue name	DFHTDSUB	SUSPEND LOCK	User	"Transient data waits" on page 118
TIEXPIRY	DS_NUDGE	DFHTISR	SUSPEND TIMER	System only	"CICS system task waits" on page 136
TSAUX Y Y	(none)	DFHTSP	SUSPEND MISC	User	"Temporary storage waits" on page 105
TSBUFFER Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSEXTEND Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSIO N N	(none)	DFHTSP	WAIT_MVS IO	User	"Temporary storage waits" on page 105
TSOPEN4B Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSQUEUE Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSSTRING Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSUT Y Y	(none)	DFHTSP	WAIT_MVS WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
TSWBUFFR Y Y	(none)	DFHTSP	WAIT_OLDC LOCK	User	"Temporary storage waits" on page 105
UDSA Y Y	(none)	DFHSMSQ	SUSPEND MISC	User	"Storage waits" on page 103

Table 18 (Page 9 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status		Resource name	Suspending module	DSSR call WLM wait type	Task	Where to look next
USERWAIT		ECB list	DFHEIQSK	WAIT_MVS MISC	User	—
Y	Y					
N	N					
XRGETMSG		message queue name	DFHWMQG	WAIT_MVS *See 1 on page 74*	System only	"CICS system task waits" on page 136
XRPUTMSG		message queue name	DFHWMQP	WAIT_MVS MISC	User	"XRF alternate system waits" on page 134
Y	Y					
ZC_ZCGRP		ZSLSECB	DFHZCGRP	WAIT_MVS MISC	System only	"VTAM terminal control waits" on page 116
ZC_ZGCH		CHANGECB	DFHZGCH	WAIT_MVS MISC	User	"VTAM terminal control waits" on page 116
ZC_ZGRP		PSINQECB	DFHZGRP	WAIT_MVS MISC	System only	"VTAM terminal control waits" on page 116
ZC_ZGRP		PSOP1ECB	DFHZGRP	WAIT_MVS MISC	System only	"VTAM terminal control waits" on page 116
ZC_ZGRP		PSOP2ECB	DFHZGRP	WAIT_MVS MISC	System only	"VTAM terminal control waits" on page 116
ZC_ZGUB		PSUNBECB	DFHZGUB	WAIT_OLDC MISC	System only	"VTAM terminal control waits" on page 116
ZCIOWAIT		DFHZERH1	DFHZERH	SUSPEND CONV	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZARQ1	DFHZARQ	SUSPEND *See 4 on page 75*	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZARL1	DFHZARL	SUSPEND *See 4 on page 75*	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZARL4	DFHZARL	SUSPEND *See 4 on page 75*	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZARR1	DFHZARR1	SUSPEND *See 4 on page 75*	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZARER	DFHZARER	SUSPEND MISC	User	"VTAM terminal control waits" on page 116
Y	N					

Table 18 (Page 10 of 10). Resources that a suspended task might be waiting on

Resource type / Purge status		Resource name	Suspending module	DSSR call WLM wait type	Task	Where to look next
ZCIOWAIT		DFHZERH2	DFHZERH	SUSPEND CONV	User	"VTAM terminal control waits" on page 116
Y	N					
ZCIOWAIT		DFHZERH3	DFHZERH	SUSPEND CONV	User	"VTAM terminal control waits" on page 116
Y	N					
ZCZGET		DFHZARL2	DFHZARL	SUSPEND MISC	User	"VTAM terminal control waits" on page 116
Y	N					
ZCZNAC		DFHZARL3	DFHZARL	SUSPEND MISC	User	"VTAM terminal control waits" on page 116
Y	N					
ZCZNAC		DFHZERH4	DFHZERH	SUSPEND CONV	User	"VTAM terminal control waits" on page 116
Y	N					

Notes:

1. If there is a terminal associated with the task.
2. "nn" is the two-character journal ID.
3. "bbbbb" is the block number.
4. "xxx" is literal.
5. The task has not yet started, because the system is at MXT.
6. These waits are used by the CICS/ESA Front End Programming Interface and are not discussed in this manual.
7. The task has not yet started because it is being held for transaction class purposes.

Transaction manager waits

Formatting a system dump using the keyword XM=1 provides a number of transaction manager summaries that are useful for identifying why tasks have failed to run.

A task may fail to run if the system is at MXT, or if the task is defined in a transaction class that is at its MAXACTIVE limit.

Maximum task condition waits

Tasks can fail to run if either of the following limits is reached:

- MXT (maximum tasks in CICS system)
- MAXACTIVE (maximum tasks in transaction class)

If a task is waiting for entry into the MXT set of transactions, the resource type is MXT, and the resource name is XM_HELD. If a task is waiting for entry into the

MAXACTIVE set of transactions for a TCLASS, the resource type is TCLASS, and the resource name is the name of the TCLASS that the task is waiting for.

If a task is shown to be waiting on resource type MXT, it is being held by the transaction manager because the CICS system is at the MXT limit. The task has not yet been attached to the dispatcher.

The limit that has been reached, MXT, is given explicitly as the resource name for the wait. If this type of wait occurs too often, consider changing the MXT limit for your CICS system.

Transaction summary

The transaction summary (Figure 7 on page 88) lists all transactions (user and system) that currently exist. The transactions are listed in order of task number and the summary contains two lines per transaction.

The meanings of the column headings are as follows:

Tran id The primary transaction id associated with the transaction

Tran num The unique transaction number assigned to the transaction

Txn Addr The address of the transaction control block

Txd Addr The address of the transaction definition instance associated with the transaction

Start Code The reason the transaction was attached, as follows:

- **C** A CICS internal attach
- **T** A terminal input attach
- **TT** A permanent transaction terminal attach
- **QD** A transient data trigger level attach
- **S** A START command without any data
- **SD** A START command with data
- **SZ** A front end programming interface (FEPI) attach
- **DF** Start code not yet known–to be set later.

Sys Tran Indicator (Yes or No) of whether the transaction is attached as a system transaction. System transactions do not contribute towards MXT.

Status An indicator of how far through attach the transaction has progressed and whether the transaction is abending or not. The first line may take the following values:

- **PRE** The transaction is in the early stages of attach.
- **TCLASS** The transaction is waiting to acquire membership of a tclass.
- **MXT** The transaction is waiting on MXT.
- **ACT** The transaction is active, that is, it has been DS attached.

Depending on the value in the first line, the second line of the status field may further qualify the transaction state. For each first line value, the meaning of the second line is as follows:

PRE No data is displayed in the second line

TCLASS The second line contains the name of the tclass that the transaction is waiting to join.

MXT or ACT If applicable, the second line indicates if the transaction is flagged for deferred abend or a deferred message, or if the transaction is already abending, as follows:

DF(xxxx) indicates that the transaction is scheduled for deferred abend, where xxxx is the abend code.

DM(yy) indicates that the transaction is scheduled for a deferred message, and yy indicates the message type

AB(xxxx) indicates that the transaction is already abending with abend code xxxx.

DS token The token identifying the DS task (if any) assigned to the transaction.

Facility type Type of the principal facility owned by the transaction.

Facility token Transaction token for the principal facility owner.

AP token The AP domain transaction token.

The first word of this token contains the address of the TCA (if any) associated with the transaction.

PG token The program manager transaction token.

XS token The security domain transaction token.

US token The user domain transaction token.

RM token The recovery manager transaction token.

SM token The storage manager domain transaction token.

MN token The monitoring domain transaction token.

Figure 7 on page 88 shows a transaction summary.

```
==XM: TRANSACTION SUMMARY

Tran Tran  TxnAddr Start Sys  Status   DS       Facility Facility AP       PG       XS       US       RM       SM
id   num   TxdAddr code  Tran           token    type     token    token    token    token    token    token    token
-------------------------------------------------------------------------------------------------------------------
CSTP 00003 10106200 C    Yes  ACT      00120003 None     n/a      10164600 00000000 00000000 00000000 1016C000 10089020
           101793C0                                              01000000 1017E000 00000000 00000000 10164600 00000000

CSNE 00031 10106100 C    Yes  ACT      00000003 None     n/a      10164C00 00000000 00000000 00000000 1016C058 11542054
           10A34B40                                              01000000 1017E048 00000000 00000000 10164C00 00000000

IC06 10056 10E2B200 T    No   ACT      089601C7 Terminal 10E167A0 1124F600 00000000 00000000 10114023 1016C9A0 11543610
           10AC9300                                              00000000 00000000 1017E7E0 00000000 10E0F6A0 1124F600 00000000

IC12 10058 10E34C00 SD   No   ACT      050601AD None     n/a      001DE600 00000000 00000000 10114023 1016C9F8 11545114
           10AC93C0                                              00000000 1017E828 00000000 10E31400 001DE600 00000000

TA03 93738 10E0E000 T    No   ACT      088211E3 Terminal 10ED9000 0024B000 00000000 00000000 10114023 1016C738 115437B0
           10AD3D40                                              00000000 00000000 1017E090 00000000 10117D60 0024B000 00000000

TA03 93920 10AFF200 T    No   TCL      00000000 Terminal 11214BD0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3D40           DFHTCL03                       00000000 00000000 00000000 00000000 10117680 00000000 00000000

TA03 93960 10E2D200 T    No   TCL      00000000 Terminal 10E573F0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3D40           DFHTCL03                       00000000 00000000 00000000 00000000 10E0F6C0 00000000 00000000

TA03 93967 10AFEA00 T    No   TCL      00000000 Terminal 10ECCBD0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3D40           DFHTCL03                       00000000 00000000 00000000 00000000 10117540 00000000 00000000

TA03 94001 10E34800 T    No   ACT      00000000 Terminal 10E2C3F0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3D40           DF(AKCC)                       00000000 00000000 00000000 00000000 10E31120 00000000 00000000

TA02 95140 10E2D300 T    No   ACT      0386150D Terminal 10E2C5E8 00057000 00000000 00000000 10114023 1016C790 11544754
           10AD3C80                                              00000000 00000000 1017E510 00000000 10E0F320 00057000 00000000

TA02 95175 10E12C00 T    No   TCL      00000000 Terminal 10E937E0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3C80           DFHTCL02                       00000000 00000000 00000000 00000000 10E0F100 00000000 00000000

TA02 95187 10E0B000 T    No   TCL      00000000 Terminal 10EA95E8 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3C80           DFHTCL02                       00000000 00000000 00000000 00000000 10117800 00000000 00000000

TA02 95205 10E2D600 T    No   MXT      00000000 Terminal 10E837E0 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3C80           DF(AKCC)                       00000000 00000000 00000000 00000000 10E0F780 00000000 00000000

TA04 96637 10E33000 T    No   ACT      060408E7 Terminal 10E05BD0 00057600 00000000 00000000 10114023 1016C7E8 115457C8
           10AD3E00                                              00000000 00000000 1017E558 00000000 10E31040 00057600 00000000

TA04 96649 10E34000 T    No   TCL      00000000 Terminal 10AE89D8 00000000 00000000 00000000 10114023 00000000 00000000
           10AD3E00           DFHTCL04                       00000000 00000000 00000000 00000000 10E312C0 00000000 00000000

F121 99305 10E2D800 T    No   ACT      020C1439 Terminal 10EA93F0 00060000 00000000 00000000 10114023 1016C898 115423FC
           10AD3BC0           AB(AFCY)                       00000000 00000000 1017E708 00000000 10E0F920 00060000 00000000

TS12 99344 10AFED00 T    No   MXT      00000000 Terminal 10E499D8 00000000 00000000 00000000 10114023 00000000 00000000
           10AD6B40                                              00000000 00000000 00000000 00000000 101178C0 00000000 00000000
```

Figure 7. Transaction summary

Notes for Figure 7

1. Transactions 00003 and 00031 are system transactions.
2. Transactions 93920, 93960, and 93967 are waiting because tclass DFHTCL03 is at its MAXACTIVE limit.
3. Transaction 94001 is scheduled to abend AKCC because it was attached when tclass DFHTCL03 was at its PURGETHRESH limit.
4. Transactions 95175 and 95187 are waiting because tclass DFHTCL02 is at its MAXACTIVE limit.
5. Transaction 95205 was scheduled to abend AKCC because it was attached when tclass DFHTCL02 was at its PURGETHRESH limit. It was subsequently made to queue because CICS is at its MXT limit.

6. Transaction 99305 is abnormally terminating with abend code AFCY.
7. Transaction 99344 is queuing because the system is at its MXT limit.
8. Transactions waiting in the transaction manager have no DS token, which is indicated by zeros in the summary.

MXT summary

The MXT summary indicates whether CICS is currently at MXT and shows the current number of queued and active transactions. To check the status of an individual transaction, consult the main transaction summary (Figure 7 on page 88).

```
==XM: MXT SUMMARY

   Maximum user tasks (MXT):                7

   System currently at MXT:                 Yes

   Current active user tasks:               7

   Current queued user tasks:               2

 * Peak active user tasks:                  7

 * Peak queued user tasks:                  2

 * Times at MXT limit:                      1

* NOTE: these values were reset at 18:00:00 (the last statistics interval collection)
```

Transaction class summary

The transaction class summary lists each transaction class that is currently installed. For each class, the current number of active and queued transactions is shown. A transaction class is at its MAXACTIVE limit if its 'current active' total is greater than or equal to its 'max active' setting. If a transaction class is at its MAXACTIVE limit, a number of transactions may be queueing in that transaction class. The transaction id and number of each queued transaction is listed with its transaction class (for example, transaction classes DFHCTL01, DFHCTL02, and DFHCTL03 in Figure 8 on page 90).

```
==XM: TCLASS SUMMARY

 Tclass       Max     Purge    Current   Current     Total    Queuing  Queuing      Queuing
  Name      Active  Threshld    Active    Queued  Attaches    TranNum  Transid     Start Time
--------  --------  --------  --------  --------  --------    -------   ----     ------------
DFHTCL01         1         0         0         0         0
DFHTCL02         1         3         1         2         7    95175     TA02     18:00:19.677
                                                             95187     TA02     18:00:24.624
DFHTCL03         1         4         1         3        29    93920     TA03     17:55:40.584
                                                             93960     TA03     17:55:42.230
                                                             93967     TA03     17:55:52.253
DFHTCL04         1         0         1         1        23    96649     TA04     18:06:04.348
DFHTCL05         1         0         0         0         0
DFHTCL06         1         0         0         0         0
DFHTCL07         1         0         0         0         0
DFHTCL08         1         0         0         0         0
DFHTCL09         1         0         0         0         0
DFHTCL10         1         0         0         0         0

*** Note that the 'Total Attaches' figures were reset at 18:00:00 (the last statistics interval collection)
```

Figure 8. Transaction class summary

A user task is waiting on resource type FOREVER

If you have found that a user task is waiting on a resource type of FOREVER, and resource name DFHXMTA, transaction manager has detected a severe error during task initialization or task termination. Transaction manager has suspended the task.

The suspended task is never resumed, and holds its MXT slot until CICS is terminated. **You must cancel CICS to remove this task as you will be unable to quiesce the system**. You cannot purge or forcepurge the task.

This wait is always preceded by one of the following messages: DFHXM0303, DFHXM0304, DFHXM0305, DFHXM0306, DFHXM0307, DFGXM0308, DFHXM0309, DFHXM0310. Transaction manager also takes a dump and message DFHME0116 is produced and contains the symptom string.

Lock manager waits

Read this section if a resource name of LMQUEUE has been shown for a task. It means that the suspended task cannot acquire the lock on a resource it has requested, probably because another task has not released it.

A user task cannot explicitly acquire a lock on a resource, but many of the CICS modules that run on behalf of user tasks do lock resources. If this is a genuine wait, and the system is not just running slowly, this could indicate a CICS system error.

Collecting the evidence

You need to take a system dump, and format it using keywords LM and DS. This formats the storage areas belonging to lock manager domain and dispatcher domain. This section describes the data that you should find there if the resource locks are being managed correctly.

Turn to the lock manager summary information (Figure 9 shows an example of this).

```
LOCK      LOCK       OWNER       MODE    COUNT   # LOCK     # LOCK    -> QUEUE
NAME      TOKEN                                  REQUESTS   SUSPENDS
----      -----      -----       ----    -----   --------   --------  --------
SMLOCK    03B051D8                                      0          0
DSITLOCK  03B05208                                      4          0
LD_GBLOK  03B05238   03B0AAD0    EXCL                   1          1  03B09378
LD_LBLOK  03B05268                                      0          0
DMLOCKNM  03B05298   03B0B690    EXCL                  35          0
CCSERLCK  03B052C8                                      0          0
==LM: LOCK WAIT QUEUE
LOCK      ADDRESS    -> NEXT     OWNER     MODE  SUSPEND    STATUS
NAME                                           TOKEN
----      -------    -------     -----     ----  -------    ------
LD_GBLOK  03B09378   00000000    03B0B3A0  EXCL  010B0001
```

Figure 9. Lock manager summary information

Table 19 describes each of the fields in the lock manager summary information.

Table 19 (Page 1 of 2). Fields in the lock manager summary information

Field	Description
LOCK NAME	The name given to the lock by the domain that originally issued the ADD_LOCK command.
LOCK TOKEN	The token assigned by the lock manager to uniquely identify the lock.
OWNER	A token that uniquely identifies the owner of the lock. It is blank unless a task currently holds the lock, in which case the KE_TAS number of the task is given.
MODE	The lock mode. It can be: **Blank** No task currently holds the lock. **EXCL** The lock is exclusive – only one task can hold the lock at any one time. The lock owner is identified in the OWNER field. **SHR** The lock is shared – several tasks can hold the lock. In this case, the OWNER field will be blank.
COUNT	Blank unless the lock mode is SHR, when it shows the number of tasks currently holding the shared lock.
# LOCK REQUESTS	The cumulative total of the number of times a lock has been requested – that is, the number of times the LOCK request has been issued for the lock.
# LOCK SUSPENDS	The cumulative total of the number of tasks that have been suspended when requesting this lock because the lock is held by another task.

Table 19 (Page 2 of 2). Fields in the lock manager summary information	
Field	**Description**
-> QUEUE	Blank unless tasks are currently suspended, awaiting the lock. If this is the case, this field contains the address of the first such task. Further information about the task is given in the 'LOCK WAIT QUEUE' section of the information.
ADDRESS	The address of the lock manager LOCK_ELEMENT that represents the suspended task.
-> NEXT	The address of the next task in the queue awaiting the lock. If this field is zeros, this is the last task in the queue.
OWNER	The KE_TAS number of the task that is currently suspended, awaiting the lock.
MODE	The lock mode. It can be: **EXCL** The lock is exclusive – only one task can hold the lock at any one time. The lock requester is identified in the OWNER field. **SHR** The lock is shared – several tasks can hold the lock.
SUSPEND TOKEN	The dispatcher suspend token for the suspended task.
STATUS	The status of the suspended task. It can be: **Blank** The task is waiting to acquire the lock. **DELETED** The suspended task has been deleted from the queue. This occurs only if the lock is deleted. **PURGED** The task was purged while waiting to acquire the lock.

The first step is to establish which lock the suspended task is waiting on. Obtain the KE_TAS number from the dispatcher domain summary for the suspended task and match this with an OWNER in the 'LOCK WAIT QUEUE' section of the lock manager summary information.

In the example, only one task is suspended and waiting to obtain the LD_GBLOK lock. The owner (KE_TAS identifier) of this task is 03B0B3A0.

You then have to find out which task currently holds the lock that the suspended task is waiting on. You can do this by looking at the lock manager summary for that lock – in this case, LD_GBLOK.

If the mode of the lock is SHR (shared), you will not be able to proceed any further and you will have to contact your IBM Support Center.

If the mode is EXCL (exclusive), the identifier of the task that currently holds the lock is given in the OWNER field. In the example, the task that currently has the lock – LD_GBLOK – is 030B0AAD0. Because the OWNER field is the KE_TAS identifier of the task, you can find out from the dispatcher domain summary the status, dispatcher task number, and TCA address of the task that currently holds the lock.

When you have all this information ready, contact the IBM Support Center and report the problem to them.

DL/I waits

Unless your task is waiting on resource type DLI, with resource name IWAIT, it can be difficult to gather online evidence that the task is waiting for a DL/I-related resource. However, you might suspect this from your knowledge of the transaction.

The waiting task might be in a local CICS region or in a remote CICS region, but the possible causes of DL/I waits are the same in either case.

These are the potential reasons why a task might be waiting:

- Tasks might be contending for PSB scheduling.
- There might be no DL thread available.
- The task might have issued a termination call while some other task was doing PSB scheduling.
- There might be insufficient storage in the PSB pool.
- There might be insufficient storage in the DMB pool.
- Tasks might be contending for PCBs, when concurrent use is not allowed.

To find which of these is the reason for the wait, you need to analyze the dump taken while the task was in a wait state. First format the dump to give you the XM and dispatcher summaries and the DL/I information. Determine the ECB for the wait from the dispatcher summary. Then determine the address of the IMS module that issued the wait. This address is located in register 14 of the IMS save area passed to the IWAIT routine in DFHDLR.

If you can reproduce the wait with tracing active, you can use the trace, with a dump, to find the same information.

Investigating DL/I waits using dump and trace

The technique described in this section enables you to find which DL/I resource that your task is waiting on.

You need to select some appropriate tracing options (see below), and then reproduce the problem. You probably need to re-create the conditions that existed when the problem first appeared, for example in your production system when the transaction throughput has reached a certain level. This is because the problem is more likely to occur when competition between tasks for resources becomes significant.

Use the transaction CETR to select level-1 trace points for component FC (file control) for special tracing. For guidance about using the CETR transaction, refer to Chapter 14, “Using traces in problem determination” on page 219. Make sure that no other special trace points have been defined. Next, use CETR to select special tracing for a task that you know is affected by the problem. Direct trace output to the auxiliary trace destination only (to minimize overhead), and set the master system trace flag off to prevent any standard task tracing.

Start tracing, and keep invoking the task that experiences the problem until it enters a wait state. Now stop tracing, and force a CICS system dump using CEMT PERFORM SNAP. If the task is waiting for the same DL/I resource as when you

first observed the problem, you should have all the evidence you need to find the cause. You can do all the rest of the investigation offline.

DL/I waits – interpreting the evidence

You need to process the auxiliary trace data set containing the trace entries you want, using the trace utility program, DFHTU410. You can then either browse it online, or print it. You also need to process the CICS system dump, using keyword **DLI**.

Consider first the auxiliary trace output, where you should look for significant trace entries relating to your task. They are made from trace points having the trace point identifier AP 00F8. For interpretation of this trace point, see the *CICS/ESA Diagnosis Reference* manual.

Trace entries having this trace point ID are made:

- When a DL/I function is requested from CICS. The request byte field is REQD(0003).
- When the DL/I function is exited. The request byte field is REQD(0005).
- After an IMS module has issued an IWAIT, causing a user task in the CICS address space to be suspended. The request byte field is REQD(0004).

Trace entries made when a task is suspended, that is, those with REQD(0004), are the most interesting ones. The information given in the trace entry includes:

- The address of the ECB being waited on. This is given as the FIELD B value.
- The address of the IMS module issuing the IWAIT. This is the FIELD A value.

Most of the ECBs addressed by FIELD B are owned by CICS, and are located in the DL/I interface parameter list (control block DLP). Sometimes you see an ECB owned by IMS, and located in the partition specification table control block (PST), but you find it in the CICS address space. You get all of the ECBs when you use DLI as the dump formatting keyword.

Some ECBs are only ever used for one specific type of wait, and the identity of the ECB defines the reason for the wait. Other ECBs are used for more than one type of wait, and in such a case you cannot find the reason for the wait from the identity of the ECB alone.

The address of the caller is only of interest for high-level debugging when the ECB can be used for more than one type of wait. Then, it enables you to find the specific reason for the wait from the various possibilities.

Table 20 lists the ECBs that can be used for DL/I waits, the control blocks where they can be found, and the reasons why tasks might be waiting on them.

Table 20. ECBs used for DL/I waits

ECB name	Control block	Reason for wait
DLSCHECB	DLP	PSB scheduling code locked
DLISBECD	DLP	No DL thread available
DLDSTECB	DLP	Termination request
DLPPSECB	DLP	No PSB pool space available
DLPSBECB	DLP	No DMB pool space available
PSTDECB	PST	1. DLI code lock 2. PSB load I/O 3. DMB load I/O

The following is a quick way of finding the identity of an ECB from its address, either online using soft copy of the formatted dump, or offline using hard copy:

1. Locate the address given in FIELD B.
2. Browse backward through the dump until you see an eye-catcher telling you the name of the control block.
3. Determine the offset of the ECB from the start of the control block, and then use either the diagnosis information in the *CICS/ESA Diagnosis Handbook* (for CICS control blocks) or the *IMS Diagnosis Guide and Reference manual* (for IMS control blocks) to find its identity.

Each of the possible types of wait, and the way you can distinguish between them for ECBs DLPSBECB and PSTDECB, are discussed in the sections that follow.

ECB “DLSCHECB” – PSB scheduling code locked

If you find that a task is waiting on ECB DLSCHECB, it is because another task has a lock on the PSB scheduling code.

ECB “DLISBECD” – no DL thread available

One of the activities of PSB scheduling is to obtain a DL thread. If no thread is available, the one task currently doing the scheduling is made to wait on ECB DLISBECD.

If this is a frequent problem in your installation, consider increasing the number of DL threads. However, you must consider the trade-offs—for example, the requirement for extra storage in the DSAs.

ECB “DLDSTECB” – Termination request during PSB scheduling

DL/I does not allow a task to execute a termination call while some other task is scheduling a PSB. CICS therefore causes tasks making termination requests to wait on this ECB until the scheduling is complete.

Also be aware that the task making the termination call releases the PSB it has scheduled when a return is made from the call. Thus, if any more DL/I calls are to be made, the task must schedule another PSB. This could involve a further wait until the scheduling thread becomes available again.

ECB “DLPPSECB” – no PSB pool space available

If the task is waiting on ECB DLPPSECB, it means that there is currently no PSB pool space available.

The waiting task is doing PSB scheduling, and has successfully obtained a DL thread. An attempt has been made to load the PSB into the PSB pool, but it has failed because all the PSB pool space is being used by other tasks doing PSB scheduling.

In a busy system where many tasks are scheduling PSBs, you can expect this sort of wait from time to time. You might consider increasing the PSB pool size, but consider the trade-offs—it means using up valuable storage.

ECB “DLPSBECB” – No DMB pool space

If your task is waiting on ECB DLPSBECB, it is because no DMB pool space is available.

When a PSB has been successfully loaded, any DMBs it references that are not already in storage must also be loaded. If no space is available in the DMB pool, no more DMBs can be loaded and the task needing them is suspended. The task resumes only when other tasks finish their DL/I processing, and sufficient DMB pool space becomes available.

If you find that DMB pool space is often exhausted, you might consider increasing the amount of storage available by specifying a greater amount on the DMBPL system initialization parameter.

ECB “PSTDECB” – DLI code lock, PSB load I/O, or DMB load I/O

If you find that a task is waiting on ECB PSTDECB, it indicates either an error within CICS or IMS code, or some hardware fault preventing a PSB or DMB from being loaded.

If you have no evidence of a hardware fault, contact the IBM Support Center and report the problem to them.

DBCTL waits

Read this section if you have any of the following problems:

- You have attempted to connect to DBCTL using the CICS-supplied transaction CDBC, but the connection process has failed to complete.
- You have a user task in a wait state, and you have found that it is waiting on resource type DBCTL, with resource name DLSUSPND.
- You have attempted to disconnect from DBCTL using the CICS-supplied transaction CDBC, but the disconnection process has failed to complete.

Each of these types of problem is dealt with in turn.

Connection to DBCTL has failed to complete

Connection to DBCTL using the CICS-supplied transaction CDBC takes place in two phases. You can find the current phase using either transaction CDBC, by refreshing the screen display, or transaction CDBI.

In phase 1, CDBC simply passes the request for connection to IMS, and returns. It is very unlikely for a wait to occur during this phase, unless there is an error in CICS code. In such a case, you would see this message displayed whenever you inquired on the connection status using CDBI:

```
DFHDB8291I DBCTL connect phase 1 in progress.
```

In phase 2, IMS processes the request asynchronously, and returns to CICS when connection is complete. Until the connection is complete, you see this status message displayed whenever you inquire with CDBI:

```
DFHDB8292I DBCTL connect phase 2 in progress.
```

If this phase fails to complete, the failure is associated with IMS. See the *IMS Diagnosis Guide and Reference manual* for guidance about debugging the problem.

A user task is waiting on resource type DBCTL

If you have found that a user task is waiting on a resource type of DBCTL, and resource name DLSUSPND, the task has made a DL/I request. It has been suspended by CICS while the request is serviced by DBCTL. If the task has not resumed, the request has not been completed, for some reason.

Disconnection from DBCTL has failed to complete

When you use CDBC to disconnect from DBCTL, it invokes another CICS transaction, CDBT. CDBT makes the disconnection request to DBCTL, and is suspended by CICS while DBCTL services the request asynchronously.

If disconnection fails to complete, you can inquire on CDBT using, for example, CEMT INQ TASK to see how far disconnection has progressed. You will probably find that CDBT is waiting on resource type DBCTL and resource name DLSUSPND, in which case the request is being processed by DBCTL.

- If CDBT *is* waiting on DBCTL, what you do next depends on whether you have requested “orderly” or “immediate” disconnection.
  - If you have requested “orderly” disconnection, it is likely that DBCTL is waiting for conversational tasks to finish. You can override an “orderly” disconnection by requesting “immediate” disconnection, in which case the process should end at once.
  - If you have requested “immediate” disconnection, and this does not happen, there is an unexpected wait within IMS. See the *IMS Diagnosis Guide and Reference manual* for guidance about investigating the problem.
- If CDBT *is not* waiting on DBCTL, this indicates a problem with CICS code. Contact the IBM Support Center for further assistance.

EDF waits

A user task is made to wait on resource type EDF and resource name DBUGUSER when, under the EDF session, CICS has control for EDF processing.

Journal control waits

Read this section if the resource type your task is waiting on is JASUBTAS or starts with the characters JC, indicating journal control.

Resource type JASUBTAS

The purpose of this wait is to make the shutdown (normal or immediate) wait until the JASP subtask has completely submitted all the archiving jobs of those journals needing to be archived (as determined in the DFHJACD data set).

Resource type JCAVLECB

If the resource type is JCAVLECB and the resource name is LECBECB, the task is waiting for a logical ECB to become available. This type of wait should only occur at times of high journaling activity.

The number of logical ECBs is a function of the number of journals defined in the JCT. Increasing the number of journals increases the number of logical ECBs.

Resource type JCBUFFER

If the resource type is JCBUFFER, with resource name AVAIL_nn ("nn" is a journal number in the range 01 through 99), the buffer is probably nearly full, or the journal is unavailable, for example, because of a volume switch or an output error.

If the resource type is JCBUFFER, with resource name JCTAECB, the task that has requested shutdown is waiting for the journaling task to flush the buffer, close the journal, and terminate itself.

Resource type JCFLBUFF

Tasks are made to wait on resource type JCFLBUFF after they have issued a request to CLOSE a journal, while the buffer is flushed (written to the journal). You can identify the journal for which the close request has been made from the final two characters of the resource name, FLUSH_nn ("nn" is the journal number, in the range 01 through 99).

You are likely to have an extended wait on this resource type only if an I/O problem arises. Check to see if any diagnostic messages have been issued to indicate that this has happened.

Resource type JCIOBLOK

User tasks are suspended on resource type JCIOBLOK, with resource name Jnnbbbbb ("nn" is the journal number, in the range 01 through 99, and "bbbbb" is the block number), when they make a JOURNAL request with the WAIT option. A task suspended in this way with the WAIT option waits until I/O is complete.

An extended wait at this point indicates that there is either an I/O problem or a system error.

Resource type JCIOCOMP

Resource type JCIOCOMP can have any of these resource names associated with it:

JCTICA
JCTIOECB
RDRECDnn
RDBLOKnn

Only JCTIOECB is associated with waits that can involve user tasks.

Resource name JCTIOECB: User tasks that issue CLOSE requests are made to wait on resource type JCIOCOMP and resource name JCTIOECB while I/O completes. Specifically, the CLOSE request is made to wait for an already active I/O request to complete before proceeding. An extended wait might indicate an I/O problem.

Resource type JCJAGET

A user task is made to wait on resource type JCJAGET and resource name JABVECB when it inquires on a journal. The journal status is obtained from the journal archive control data set (JACD), and the wait is for I/O to complete as the record is read.

If there is an extended wait on this resource type, this indicates that there is either an I/O problem or a system error.

Resource type JCJAPUT

A user task is made to wait on resource type JCJAPUT and resource name JABVECB when any of its journaling activities cause the state of a journal data set to change. This can be, for example, when a switch is made from one journal extent to another, as the one extent is closed and the next is opened. The change in status is written to the JACD, and the wait is for I/O to complete.

If there is an extended wait on this resource type, this indicates that there is either an I/O problem or a system error.

Resource type JCLASTBK

A user task that has requested a journal switch can be made to wait on resource type JCLASTBK and resource name Jnnbbbbb (“nn” is the journal number, from 01 through 99, and “bbbbb” is the block number) when I/O is already active. The wait is for the block identified by “bbbbb” to be written to the journal before it can be closed. It is essentially an I/O wait, and there could be an I/O problem if there is an extended wait at this point.

Resource type JCCLDONE

User tasks can be made to wait on resource type JCCLDONE and resource name SUBTASK, but an extended wait at this point indicates that there is a system error. The wait is for the OPEN/CLOSE subtask to complete the close request, and the user task cannot influence this.

Resource type JCDETACH

A task that has requested shutdown can be made to wait on the detaching of the journal subtask from the operating system.

Resource type JCOPDONE

User tasks can be made to wait on resource type JCOPDONE and resource name SUBTASK, but an extended wait at this point indicates that there is a system error. The wait is for the OPEN/CLOSE subtask to complete the open request, and the user task cannot influence this.

Resource type JCREADY

Resource type JCREADY can have any of these resource names associated with it:

JCTXAECB
JCTXBECB
JCTXXECB

User tasks are made to wait on resource type JCREADY and one of these resource names when no operator reply has been received to message DFH4583, and message DFH4584 has subsequently been issued. For details of the circumstances under which these messages are issued, and the actions you need to take, use the CMAC transaction for online message information, or see the *CICS/ESA Messages and Codes* manual.

The fifth character in the resource name (A, B, or X) identifies the next extent to be used.

Resource type JCRQDONE

User tasks can be made to wait on resource type JCRQDONE and resource name SUBTASK in two routines:

DFHJCC (user tasks)
DFHJCO (user tasks)

The task is waiting for the journaling subtask to complete an open or close request issued by another task. Extended waits indicate either a system error, or possibly an I/O problem.

Resource type JCSWITCH

If the resource type is JCSWITCH, with resource name DS_SDJnn (“nn” is the journal number, from 01 through 99), a wait is issued on the task that requested shutdown when a journal switch is already active. Extended waits indicate either a system error, or possibly an I/O problem.

Resource type JCTAPE2

Resource type JCTAPE2 can be associated with two different resource names:

PREOPNnn
STABLEnn
(“nn” is the journal number, which is in the range 01 through 99).

The waits are both associated with “open ahead” tape processing, but in different circumstances.

Resource name PREOPNnn: This wait is issued when a volume switch has been requested and “open ahead” processing has not completed. It probably indicates that the volume that is to be opened ahead is not ready.

Resource name STABLEnn: This wait is issued during shutdown processing, when previous “open ahead” processing failed to complete. Like resource name PREOPNnn, it probably indicates that the volume to be opened is not ready.

Journaling cannot be shut down until this processing is complete, and it is potentially a cause of CICS stalling during termination processing.

Task control waits

If your task is waiting on a resource type of KCCOMPAT or KC_ENQ, it has been suspended by the transaction manager. If your task is waiting on a resource type of EKCWAIT, it has been suspended by task control.

KC_ENQ indicates that CICS code acting for a task has issued an EXEC CICS ENQ command or a DFHKC TYPE=ENQ macro. If there is an extended wait for no apparent reason, this might indicate an error within CICS. If that turns out to be the case, contact the IBM Support Center.

EKCWAIT indicates that a task has issued an EXEC CICS WAIT EVENT command. If the wait is prolonged, you should identify the event being waited on, and:

- Check that the EXEC CICS WAIT EVENT command specified the correct event.
- Check for problems with the task that should be completing the work for the specified event. It might be waiting or looping, it might have a performance problem, or it might have failed completely.

If the resource type is EKCWAIT and the EXEC CICS WAIT EVENT command included the NAME option, the specified name is the resource name. For programming information about the NAME option of the WAIT EVENT command, see the *CICS/ESA Application Programming Reference* manual.

Resource type KCCOMPAT

If you have a resource type of KCCOMPAT, the resource name tells you more about the circumstances of the wait. The meanings of the resource names are described in Table 21.

Table 21 (Page 1 of 2). KCCOMPAT waits: meaning of resource names

Resource name	Meaning
CICS	The task has been suspended on a DFHKC TYPE=WAIT,DCI=CICS macro call. CICS has issued the macro. The task is waiting for some internal event, and the ECB should be posted by CICS under another task.

Table 21 (Page 2 of 2). KCCOMPAT waits: meaning of resource names	
Resource name	**Meaning**
LIST	The task has been suspended on a DFHKC TYPE=WAIT,DCI=LIST macro call issued by CICS code. It is waiting for any ECB in a list of ECBs to be posted, after which it is resumed.
SINGLE	The task has been suspended on a DFHKC TYPE=WAIT,DCI=SINGLE macro call issued by CICS code. It is waiting for a single ECB to be posted, after which it is resumed.
TERMINAL	The task has been suspended on a DFHKC TYPE=WAIT,DCI=TERMINAL macro call. CICS has suspended the task. The task is waiting for terminal I/O to complete, and stays suspended until resumed by CICS.

If the resource name for the wait is SINGLE, CICS, or LIST, look at the entry in the SUSPAREA column of the dispatcher summary in the dump. The type of value it contains depends on the resource name:

- For SINGLE or CICS, it is the address of an ECB
- For LIST, it is the address of a list of ECBs.

(The contents of the SUSPAREA entry are not significant for TERMINAL, because this type of wait is subject to the dispatcher RESUME function. For more information about debugging terminal waits, see “Terminal waits” on page 108.)

Check the contents of the SUSPAREA entry. Does it contain a valid address? That is, is it within the CICS address space, and actually pointing at an ECB, or a list of ECBs?

If you find an invalid address: It is possible that a storage overlay is the cause of the wait problem. If you suspect this to be the case, turn to Chapter 10, “Dealing with storage violations” on page 189 for further advice. However, note that this is likely to be a “random” overlay, and such problems are often very difficult to solve.

From the kernel information in the dump, find out which code issued the DFHKC macro call. If you think that CICS has passed an incorrect address, contact the IBM Support Center, and report the problem to them.

If you find a valid address: Consider what area the ECB is in. Does the position of the ECB, and its environment, suggest that it relates to a resource whose availability you can control? If so, you might be able to solve the problem by redefining the quantity of that resource.

If the ECB does not lie within an area that you can control, refer the problem to the IBM Support Center.

Resource type KC_ENQ

If your task is waiting on resource type KC_ENQ, it is unconditionally enqueued on a single server resource that is currently unavailable. Typically, tasks are made to wait on KC_ENQ when they make certain types of file control request, if the file is already in use. These are the cases:

- The waiting task has attempted to change the state of a file that is in use. Another task has already attempted to change the state of the same file, and is suspended on resource type FCFSWAIT. For more details, see “Resource type FCFSWAIT – waits for file state changes” on page 124.
- The waiting task has attempted to update a record in a recoverable file while another task has the lock on it. The task owning the record lock retains it until it reaches the end of the current logical unit of work (syncpoint or end of task). For more details of record locking for VSAM files, see “Resource type KC_ENQ – wait for a record lock in a recoverable VSAM file” on page 127.

If the wait on resource type KC_ENQ is prolonged:

- More than one task might be enqueued on the resource, and the task you are investigating could be some way down the list. Check the programming logic of any of your programs accessing the resource, to see if it can be released more quickly. Consider whether you can include EXEC CICS DEQ commands.
- Another (long-running) task might have used the resource and finished with it, without issuing an EXEC CICS DEQ command or a DFHKC TYPE=DEQ macro call. The resource is made available automatically when the task terminates, but in the meantime, no other tasks are able to use it.
- There might be a CICS system error. If you have considered the other possibilities and you think this is the most likely explanation, refer the problem to the IBM Support Center.

Storage waits

Read this section if you have found that a task is waiting for a long time on any of the resource types CDSA, UDSA, ECDSA, EUDSA, ERDSA, SDSA, ESDSA, or RDSA. Waits on these resources occur when tasks make unconditional storage requests (SUSPEND=YES) that cannot be satisfied. The type is CDSA, UDSA, SDSA, or RDSA for storage requests below the 16MB line, and ECDSA, EUDSA, ESDSA, or ERDSA for storage requests above the line.

Note that, if conditional requests are made (SUSPEND=NO), tasks are not suspended on these resources. Instead, an exception response is returned if the request cannot be satisfied.

CICS automatically takes steps to relieve storage when it is under stress, for example by releasing storage occupied by programs whose current use count is 0.

In addition, your task may be automatically purged if it has waited for storage longer than the deadlock time-out parameter specified in the installed transaction definition. Certain conditions prevent purging of a task, for example, a deadlock time-out value of 0, or a specification of SPURGE(NO).

The two most likely explanations for extended waits on storage requests are:

1. The task has issued an unconditional GETMAIN request for an unreasonably large amount of storage.
2. The task has issued an unconditional GETMAIN request for a reasonable amount of storage, but the system has too little available. It could be approaching SOS, or the storage could have become too fragmented for the request to be satisfied.

The first step is to get a CICS system dump, and format it using the formatting keyword SM. The way you interpret the dump to investigate each of the above possibilities is dealt with in the sections that follow.

Was the request for too much storage?

To find out if the suspended task has issued a request for too much storage, look in the SM suspend queue summary. This gives, amongst other things, the number of bytes requested by every task that has been suspended by the storage manager. You can see whether any of them has made a GETMAIN request for an unreasonably large amount of storage. For example, the following is the dump output in the SM suspend queue summary when task 41 requests 10 000 000 bytes:

```
==SM: Suspend queue summary

   KE Task  Tran #  Susptok  Subpool  DSA       Request

   053E5400 0000041 04080011 U0000041 EUDSA    10000016
```

If the suspended task has made a reasonable GETMAIN request, you next need to see if the system is approaching SOS.

Is the storage close to being exhausted?

It could be that your task has made a reasonable request for storage, but the system is too close to SOS for the request to be satisfied.

To see if this could be the cause of the wait, look at the DSA summary in the formatted dump. This tells you the current free space in each DSA, both in bytes and as a percentage of the total storage. It also tells you the size of the largest free area, that is, the biggest piece of contiguous storage. ("Contiguous storage" in this context means storage not fragmented by other records. It is accepted that records too large to fit in a single CI can be split across two or more CIs that are not necessarily contiguous.)

If the largest free area is smaller than the requested storage, this is likely to be the reason why the task cannot obtain its requested storage.

If the amount of free space is unexpectedly small, look at the task subpool summary. If a task has made an unusually large number of GETMAIN requests, this could indicate that it is looping. A looping task might be issuing GETMAIN requests repetitively, each for a reasonable amount of storage, but collectively for a very large amount. If you find evidence for a looping task, turn to Chapter 7, "Dealing with loops" on page 143.

If your task has made a reasonable request and the system seems to have sufficient free storage, you next need to see if fragmentation of free storage is causing the GETMAIN request to fail.

Is fragmentation of free storage causing the GETMAIN request to fail?

If the DSA summary shows that the current free space is significantly greater than the largest free area, it is likely that the DSA has become fragmented.

Temporary storage waits

Read this section if you have found that a user task is waiting on a resource type starting with TS, showing that it is for temporary storage.

Resource type TSAUX

A task is forced to wait on TSAUX if it has made an unconditional request for temporary storage, and the request cannot be met because insufficient auxiliary storage is available. The task has issued an EXEC CICS WRITEQ TS command, with or without the REWRITE option, but without specifying NOSUSPEND and without any code to handle the NOSPACE condition. If SPURGE(YES) is defined for the task on the CEDA DEFINE TRANSACTION command, and a deadlock time-out interval other than 0 has been specified, the task is purged when that time expires. Otherwise, it is not purged, and is liable to be suspended indefinitely.

A task that makes a conditional temporary storage WRITEQ TS request (NOSUSPEND specified) is not suspended if the request cannot be met. Instead, if the required auxiliary storage is not available, an exception response is returned to it. (There might still be a suspension for another reason—for example, the temporary storage program itself might become suspended after issuing a GETMAIN, if CICS went short on storage.)

These are the two most likely reasons why a task that has issued an unconditional WRITEQ TS request might be suspended on resource type TSAUX:

1. The task has issued a request requiring too large a piece of temporary storage.
2. The task has issued a request requiring a reasonable amount of temporary storage, but there is too little available.

   This could indicate that the amount of auxiliary storage is becoming exhausted. Otherwise, it could be that there is quite a large amount of auxiliary storage left, but the storage is too fragmented for the request to be satisfied.

The first step is to get a CICS system dump, and format it using the formatting keyword TSP to show the temporary storage control blocks. Include formatting keywords SM and KE, too, as you might need to refer to the summaries for these two components as well. The way you analyze the dump to investigate the cause of the problem is described in the sections that follow.

Is temporary storage close to being exhausted?

It could be that your task has made a reasonable request for temporary storage, but the amount of unallocated space is close to exhaustion.

To see if this could be the cause of the wait, look at the temporary storage summary in the formatted dump. If the current free space is very small, this is likely to be the reason why the task cannot obtain its requested temporary storage. In such a case, consider defining secondary extents for the data set.

Look also at the temporary storage request summary. If a task has made an unusually large number of WRITEQ TS requests, it could be looping. A looping task might be issuing WRITEQ TS requests repetitively, each for a reasonable amount of storage, but collectively for a very large amount. If you find evidence for a looping task, turn to Chapter 7, “Dealing with loops” on page 143.

If your task has made a reasonable request and the system seems to have sufficient unallocated temporary storage, you next need to see if fragmentation of unallocated storage is causing the WRITEQ TS request to fail.

Is fragmentation of unallocated storage causing the WRITEQ TS request to fail?

You can tell whether fragmentation of unallocated temporary storage is causing the WRITEQ TS request to fail by looking at the temporary storage byte map, TSBMAP, in the dump.

Each byte in the TSBMAP shows how the corresponding control interval (CI) in auxiliary storage is being used. If the CI is completely unused, the byte representing it has a value equal to the number of segments in the CI. The number of segments in each CI is shown in the temporary storage program summary in a system dump. Alternatively, you can find out the number of segments per CI from field TSASPCI in the temporary storage auxiliary control area, TSACA. A value of X'3F', for example, in field TSASPCI would show that each CI has X'3F' segments.

If the CI is completely allocated, you see a value of 0 for it in the TSBMAP. Such a CI cannot be used to satisfy further temporary storage WRITEQ TS requests.

If the TSBMAP shows you any value between 0 and the total number of segments in the CI, the CI is partly used. Even if the temporary storage data set is not yet full, that is, one or more of the CIs has unused segments, there must be sufficient *contiguous* storage available for the WRITEQ TS request before it can be satisfied. You can find out if this is a likely reason for the failure by looking in the TSBMAP at the distribution of unused segments between CIs. The temporary storage program summary in a system dump shows the number of bytes per segment. Alternatively, look in field TSABPSEG of the TSACA, to find this information.

Resource type TSBUFFER

Resource type TSBUFFER indicates that the task that is waiting has issued an auxiliary temporary storage request, but the buffers are all in use. If you find that tasks are often made to wait on this resource, consider increasing the number of auxiliary temporary storage buffers (system initialization parameter TS).

Resource type TSEXTEND

Resource type TSEXTEND indicates that the waiting task has issued a request to extend the auxiliary temporary storage data set, but some other task has already made the same request. The wait does not extend beyond the time taken for the extend operation to complete. If you have a task that is waiting for a long time on this resource, it is likely that there is a hardware fault or a problem with VSAM.

Resource type TSIO

Resource type TSIO indicates that the task is being made to wait while physical I/O takes place during an auxiliary temporary storage read or write. If there is an extended wait on this resource, it is likely that there is a hardware fault or a problem with VSAM.

Resource type TSQUEUE

Resource type TSQUEUE indicates that the waiting task has issued a request against a temporary storage queue that is already in use by another task. The latter task is said to have the lock on the queue.

The length of time that a task has the lock on a temporary storage queue depends on whether or not the queue is recoverable. If the queue is recoverable, the task has the lock until the logical unit of work is complete. If it is not recoverable, the task has the lock for the duration of the temporary storage request only.

If tasks in your system are frequently made to wait on temporary storage queues, consider the following:

- Are tasks that are performing operations on the same temporary storage queue intended to do so, or is the ID of the queue unintentionally not unique?
- Is it possible to create more temporary storage queues to reduce the contention between tasks?
- If the queue in question is recoverable, is it possible to make tasks relinquish control of it more quickly? Consider reducing the size of LUWs, or making conversational tasks pseudoconversational.

Resource type TSSTRING

Resource type TSSTRING indicates that the task is waiting for an auxiliary temporary storage VSAM string. If you find that tasks frequently wait on this resource, consider increasing the number of temporary storage strings (system initialization parameter TS).

Resource type TSUT

If a user task is waiting on resource type TSUT, activity keypointing is taking place. This involves a large amount of I/O, and, if there are many temporary storage queues, it could take a relatively long time to complete.

Resource type TSWBUFFR

Resource type TSWBUFFR indicates that the waiting task has issued an auxiliary temporary storage request, but the write buffers are all in use. You have no control over how temporary storage allocates read buffers and write buffers from the buffer pool, but if you find that tasks are often made to wait on this resource, increasing the number of auxiliary temporary storage buffers (system initialization parameter TS) should help solve the problem.

Terminal waits

Read this section if you have any of the following problems:

- A task should have started at a terminal, but has failed to do so.
- A task is waiting on a resource type of KCCOMPAT, with a resource name of TERMINAL.
- A task is waiting on a resource type of IRLINK, with a resource name of SYSIDNT concatenated with the session name.

Note that, if you have one or more unresponsive terminals, that is terminals that are showing no new output and accepting no input, this does not necessarily indicate a *terminal* wait. If you have this problem, use CEMT INQ TERMINAL to find the transaction running at the terminal, and then CEMT INQ TASK to find out what resource that task is waiting on. When you know that, look at Table 18 on page 76 to find where you can get further guidance.

If all the terminals in the network are affected, and CICS has stalled, read “What to do if CICS has stalled” on page 137 for advice about how to investigate the problem.

If yours is a genuine terminal wait, remember when you carry out your investigation that terminals in the CICS environment can have a wide range of characteristics. A terminal is, in fact, anything that can be at the end of a communications line. It could, for example, be a physical device such as a 3270 terminal or a printer, or a batch region, or it could be another CICS region connected by an interregion communication link, or it could be a system that is connected by an LUTYPE6.1 or APPC (LUTYPE6.2) protocol.

If LUTYPE6.1 is in use, the other system might be another CICS region or an IMS region. With APPC (LUTYPE6.2), the range of possibilities is much wider. It could include any of the systems and devices that support this communications protocol. For example, apart from another CICS region, there might be a PC or a DISOSS system at the other end of the link.

If you eventually find that the fault lies with a terminal, or a resource such as DISOSS, the way in which you approach the problem depends on what type it is. In some cases, you probably need to look in appropriate books from other libraries for guidance about problem determination.

Terminal waits – first considerations

Before taking a systematic approach to the problem, here are a few preliminary considerations that could point to a simple solution.

- Is there an obvious physical explanation for the wait? For example, is a terminal operator failing to respond to a request for input? In the case of a printer, has it been powered off, or has it run out of paper?
- Have you checked in the CSTL and CSNE logs to see if there is a message you might have missed? If either DFHTCP or DFHZCP detected an error related to terminal control, a message reporting the problem will have been sent to the CSNE log, and, perhaps, to the console too.

  If you do find a message there reporting some terminal error that can be related to the task, it should give you an idea of why the task is waiting. You can find an explanation of the message and a description of the system action in response to the error by using the CMAC transaction, or by looking in the *CICS/ESA Messages and Codes* manual.

  It is also possible that the CSNE log entry will show that an error was detected, but that no action was taken by TACP or NACP. This could occur if the line or terminal was out of service, or if the error actions had been turned off in the user exits of DFHTEP and DFHNEP. In such a case, the CICS code enabling the waiting task to be resumed would never be executed, and the task would wait indefinitely.
- Have any HANDLE CONDITION routines for terminal errors been coded incorrectly? If an attempt were made to access the terminal having the error in such a routine, the application would be very likely to wait indefinitely.
- If the terminal is installed using autoinstall, has the system failed to load DFHZATA, the autoinstall program, or DFHZCQ which is called by DFHZATA, because of a short on storage condition? If so, you need to deal with the cause of the short on storage condition. Check the delete delay that you have specified—if it is too short, your system may be deleting and reinstalling terminals unnecessarily. If storage fragmentation is preventing DFHZATA or DFHZCQ from being loaded consider defining them as resident. However, be aware that it is a large program, and check your storage requirements before making this change.

If none of these considerations applies, you need to start a systematic investigation into the reason for the wait.

Terminal waits – a systematic approach

You first need to determine the type of terminal involved in the wait, and the type of access method in use for that terminal. Both of these factors influence the way you perform problem determination.

Your strategy must then be to find where in the communication process the fault lies. These are the basic questions that must be answered:

1. Is the problem associated with the access method?
2. If the access method has returned, or has not been involved, is terminal control at fault?
3. If terminal control is working correctly, why is the terminal not responding?

To answer most of these questions, you will need to do some offline dump analysis. Use CEMT PERFORM SNAP to get the dump, and then format it using the formatting keyword TCP. **Don't cancel your task before taking the dump. If you do, the values in the terminal control data areas won't relate to the error.**

What type of terminal is not responding?

You can check the terminal type either online, using a system programming command, or offline, by looking at the appropriate TCTTE in the formatted system dump.

Online method: Use the transaction CECI to execute the system programming command EXEC CICS INQUIRE TERMINAL DEVICE. This returns one of the terminal types identified in the *CICS/ESA Resource Definition Guide*.

Offline method: Look at the formatted dump output you have obtained for keyword TCP. First, identify the TCTTE relating to the terminal, from the four-character terminal ID shown in field TCTTETI. Now look in field TCTTETT, and read the 1-byte character that identifies the type of terminal. You can find what terminal type is represented by the value in the field from the description given in the *CICS/ESA Diagnosis Handbook*.

What type of access method is in use?

You can use both an online method and an offline method for finding the type of access method being used by the terminal that is not responding.

Online method: Use the CECI transaction to execute the system programming command EXEC CICS INQUIRE TERMINAL ACCESSMETHOD. This returns the access method in use by the terminal.

Offline method: You can find the access method for the terminal from the TCTTE. Look in field TCTEAMIB, which is the byte name definition for the access method. The *CICS/ESA Diagnosis Handbook* relates values to access methods.

The most common access method is VTAM, identified by a value of TCTEVTAM. The problem determination procedures outlined below focus exclusively on VTAM.

You might also find either of these values, each being of special significance for problem determination:

- TCTEISMM, meaning that the access method is ISMM. This is used for interregion communication, and it shows that the resource that is not responding is a remote CICS region. In such a case, the most likely reason for the wait is that some task in the remote region is also in a wait state. The way you deal with this type of problem is described in "Your task is waiting on another CICS region" on page 115.
- TCTELU6, meaning that you have intersystem communication (ISC). In this case, the resource that is not responding is a remote system, and the way you deal with the wait depends on what the remote system is. If it is a CICS system, you need to diagnose the problem in the remote system using the techniques given in this book. If the remote system is a non-CICS system, you might need to refer to a diagnosis book from another library for advice on problem determination.

If you have any other access method, for example TCAM, you need to adapt the guidance given here accordingly.

VTAM in use – debugging procedures

The sections that follow give guidance about debugging terminal waits when the access method is VTAM.

The first step is to look in the CSNE log to see if there is an error message that explains the wait. If it contains an error code, you can find out what it means from the *VTAM Messages and Codes for MVS and VSE* manual.

Look next for any NACP error codes in fields TCTEVRC5, TCTEVRC6, TCTEVRC7, and TCTEVRC8 of the terminal table entry, TCTTE. Look also for any SNA sense code in field TCTEVNSS.

Is the problem associated with VTAM?

You can find the VTAM process status with respect to the waiting task from fields TCTEICIP and TCTEIDIP in the TCTTE. The following are the values you might find there, and their interpretations:

```
TCTECIP          command request in progress
TCTEDIP          data request in progress
```

Either of these status values indicates that a VTAM request is in progress, and that the VTAM RPL is active. A response is expected either from VTAM, or from the terminal. You can find the address of the RPL from field TCTERPLA, unless the request was for a RECEIVE on an APPC session, in which case you can find the RPL address from field TCTERPLB.

If a VTAM request is not in progress, the next field of interest is in the VTAM system area of the TCTTE. Find four bytes of VTAM exit IDs, starting at field TCTEEIDA. If any are nonzero, the VTAM request has completed. Nonzero values suggest that VTAM is not involved in the wait. You can find the meanings of the values from the VTAM module ID codes list in the *CICS/ESA User's Handbook*.

If you suspect that the problem is associated with VTAM, consider using either CICS VTAM exit tracing or VTAM buffer tracing. Both of these techniques can give

you detailed information about the execution of VTAM requests. For guidance about using the techniques, read the appropriate sections in Chapter 14, “Using traces in problem determination” on page 219.

VTAM in use – is terminal control at fault?

The first place to look is field TCTVAA1 in the terminal control table prefix, TCTFX. This contains either the address of the first TCTTE on the active chain, or the value X'FFFFFFFF'. If you see the latter value, it means that terminal control does not recognize that it has work to do. If this conflicts with the INQ TASK report you have received that the task is waiting on some terminal related activity, report the problem to your IBM Support Center.

If field TCTVAA1 points to a TCTTE on the active chain, check that the TCTTE of the terminal your task is waiting for is included in the chain. You can find this out by following the chain using the “next” pointer, field TCTEHACP of the TCTTE. If it does not contain the address of the next TCTTE on the chain, it contains either of these values:

```
X'FFFFFFFF'     this is the last TCTTE on the chain
X'00000000'     this TCTTE is not on the active chain
```

If you find a value of X'00000000', report the problem to the IBM Support Center.

VTAM in use – is the terminal at fault?

If you have found that the access method and terminal control are not causing the wait, the terminal itself must be waiting for some reason. You need now to look at some fields in the TCTTE for the terminal to find its status.

CICS system dumps contain an index to the VTAM terminal entries. It appears in the terminal control (TCP) summary section of the dump.

Information about the status and attributes of the VTAM terminals appears in an interpreted form at the end of the control block for each terminal entry. The information shown depends on the attributes of the terminal.

The example in Figure 10 on page 113 shows the index followed by a terminal entry with its interpreted status and attribute information.

```
===TCP: TERMINAL CONTROL SUMMARY (VTAM TERMINALS)
        TERMINAL                 TASK         IN        ERROR    ACTIVE RPL   WORK    ZNAC    INTERVENTION   AUTOINSTALL
TERMID    TYPE     LOGGED ON    ATTACHED     SERVICE     STATS.    REQUEST     TO DO  QUEUED     REQUIRED       ACTIVITY
 R51       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 R52       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 R53       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 R54       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 R55       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 S51       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 S52       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 S53       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 S54       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 S55       C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 -AAA      C0         NO           NO          YES      00000001      NO         NO      NO          NO            N/A
 -AAB      C0         NO           NO          YES      00000000      NO         NO      NO          NO            N/A
 TCTTE.R51 03B5C420 TCT TERMINAL ENTRY
    0000  D9F5F140 C0000504 03B5C424 00000000  00000000 00000000 00000000 00080000  *R51 ......D.....................*    03B5C420
    0020  00000000 0C000000 C5D5E400 00008080  00000000 00000000 00000000 00000000  *........ENU.....................*    03B5C440
    0040  00000000 00000000 00000000 00000000  00000000 01D80000 00000000 03B22030  *.....................Q..........*    03B5C460
    0060  00000000 00000000 00000000 03B58690  00000000 00000000 03B46390 00000000  *..............f................*    03B5C480
    0080  00000000 00000000 00000000 00000000  03B430F8 00000000 00000000 00000000  *...................8............*    03B5C4A0
    00A0  00000000 00000000 00000000 00840000  00000000 00000000 00000000 00000000  *.............d..................*    03B5C4C0
    00C0  00000000 80000000 00000000 00000000  00000000 0000003B 01000000 00000000  *................................*    03B5C4E0
    00E0  10000000 00000000 00000000 03B5C618  00000000 00000000 00000000 00000000  *..............F................*    03B5C500
    0100  3A008400 00000000 00000000 00000000  00000000 FFFF0000 00000000 00000000  *..d............................*    03B5C520
    0120  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*    03B5C540
    0140  10001000 10000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*    03B5C560
    0160  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*    03B5C580
    0180  08090000 00000000 00000000 00000000  00000000 00000000 00000000 03B59421  *............................m.*    03B5C5A0
    01A0  00440000 00001008 D4FD6038 80800014  00000000 00000000 00000000 00000000  *........M.-.....................*    03B5C5C0
    01C0  00000000 00000000 00000000 00000000  00000000 00000000                    *........................       *    03B5C5E0
    TERMID = R51                           EXIT FOOTPRINTS (HEX) = 00000000
    IN SERVICE                             TCTTECA (NO TASK ATTACHED)
    TCTECCV (STARTED BY TTI)               TCTECSM (CA-MODE)
    INPUT STATISTICS (DECIMAL) = 00000000  OUTPUT STATISTICS (DECIMAL) = 00000000
    ERROR STATISTICS (DECIMAL) = 00000000  TCTE1RY (CICS IS PRIMARY)
    TCTELSE (LUC CONTENTION LOSER)         TCTESBIF (SBI/BIS SUPPORTED)
```

Figure 10. Terminal index and terminal entry with interpreted information

The values that are given below for fields in the TCTTE are not the only possibilities, but they show important things about the terminal status. If you find any other values for these fields, look in the *CICS/ESA Diagnosis Handbook* to find out what they mean.

The following are the questions that need to be asked, and some values that could provide the answers.

1. Is the terminal in service? Look at field TCTTETS of the TCTTE, to find the terminal status. The values that indicate why a terminal was failing to respond include:

   ```
   TCTTESPO = 1 and TCTTESOS = 1     terminal out of service
   TCTTESOS = 1 only                 terminal in error recovery
   ```

   Look also at field TCTESEST, to find the session status with respect to automatic transaction initiation (ATI) for the terminal. Some of the values you might see are:

   ```
   TCTESLGI = 0     CREATESESS(NO) in TYPETERM definition
   TCTESLGI = 1     CREATESESS(YES) in TYPETERM definition
   TCTESLGT = 1     recovering CREATESESS
   ```

   A value of TCTESLGI = 0, with TCTESLGT = 0, too, shows that CREATESESS(NO) has been specified in the TYPETERM definition. This means that EXEC START requests and ATI requests cannot cause a session to be created. The request is either queued or rejected when no session is

currently established. This can put a terminal into an indefinite wait state, especially if the terminal is a printer.

A value of TCTESLGI = 1 shows that CREATESESS(YES) has been specified in the TYPETERM definition. This means that CICS is allowed to create sessions for the terminal, so the CREATESESS status is not the cause of the wait.

A value of TCTESLGT = 1 means that the session is in error recovery. This could explain why there is no response from the terminal.

2. Has a task been created for this terminal? Look first at field TCTTECA of the TCTTE.

   - If its value is nonzero, there is a task attached to the terminal. You can tell whether the task has been started from a terminal, or by ATI, from field TCTEICCV:

     ```
     TCTECCV = 0      the task has been started by a terminal
     TCTECCV = 1      the task has been started by ATI
     ```

   - If its value is zero, look in fields TCTTEIC and TCTECTI. The values you might find there are:

     ```
     TCTEOIC = 1      ATI is waiting to start
     TCTECTI = 1      there is ATI work for ZCP to do
     ```

3. Is there a task related to the terminal? You can find the task session state with VTAM from field TCTEMOST, and, if bracket protocol is required (from field TCTEIBPE), its conversation state with the terminal from field TCTEIINB. The significant values that might provide further clues to the cause of the wait are:

   ```
   TCTECSM = 1      the task is in conversation with the terminal
   TCTECSM = 0      terminal control will accept new tasks from
                    the terminal
   ```

   Look now at field TCTEIBPE, to see if bracket protocol is required:

   ```
   TCTEBPE = 1      bracket protocol is required
   ```

   If you find that bracket protocol is required, look in field TCTEIINB:

   ```
   TCTEINB = 0      a conversation has not been started
   TCTEINB = 1      a task is in conversation with the terminal
   ```

4. Is the terminal logged on to CICS? Look first at the node session status in the TCTTE. The three stages of session creation are represented by three separate bits, in fields TCTEILOS, TCTEIOPD, and TCTEINSD:

   ```
   TCTELOS = 1      the node is logged on
   TCTEOPD = 1      VTAM OPNDST macro issued
   TCTENSD = 1      Start Data Traffic sent
   ```

   If all three bits are set, so the value of the byte is TCTENIS, the node is in session.

   You next need to see if the terminal is logging off, or if it has already been logged off. The fields of interest are TCTEINND, TCTEINBD, and TCTEIPSA. The values to look for are:

   ```
   TCTENND = 1      the terminal is to be logged off
   TCTENBD = 1      the terminal is logging off because of an error
   TCTEPSA = 1      the session with the terminal ended abnormally
                    – look for any explanatory message on CSMT
   ```

If any of these bits are set, the terminal might not be able to respond to the waiting task.

5. Should the terminal respond to the task? Field TCTEIPRA tells you this:

```
TCTEPRA = 1      the terminal should respond
```

If the values you have found in all these fields suggest that the terminal status is normal, the terminal is probably waiting for some activity to complete before it responds. The type of investigation you need to do next depends on the type of terminal involved in the wait. You should already have determined this, for example by using the system programming command EXEC CICS INQUIRE TERMINAL DEVICE.

Tools you can use for debugging terminal waits when VTAM is in use

Amongst your debugging tools, two are likely to be of particular use for investigating terminal waits in a VTAM environment. They are:

- VTAM buffer trace. This is a feature of VTAM itself, and you need to refer to the appropriate manual in the VTAM library for details of how to use it.
- CICS VTAM exit trace. This is a feature of CICS, and you can control it from the CETR panel.

For a description of the use of these two types of tracing in CICS problem determination, see Chapter 14, “Using traces in problem determination” on page 219.

Your task is waiting on a physical terminal

If your task is waiting on a physical terminal, the terminal should first be checked physically to see why it is not responding. If the terminal is at a remote location, you need to ask someone else to check it for you. Some possibilities are:

- A terminal with a keyboard might be waiting for an operator to enter some data.
- A printer might have been powered off, or it could have run out of paper.

Consider also the possibility of hardware error in the terminal.

Your task is waiting on another CICS region

If a session has been acquired and it has not failed, your task is likely to be waiting for some response from a task in the other region. This can apply to any of the interregion or intersystem communication activities—function shipping, asynchronous processing, transaction routing, distributed transaction processing, or distributed program link. No matter which of these applies, it is most likely that the other region is not responding because the related task there has been suspended.

You need to identify the related task in the remote region, and find out the resource it is waiting on. When you have done that, refer to Table 18 on page 76 to find out which part of this chapter to turn to next.

Investigating the related task in the remote region

The first thing to do is to identify the region that is not responding to your local task.

If the task is using interregion communication (IRC), look first at the name of the resource being waited on, returned together with resource type IRLINK by CEMT INQ TASK. The first four characters give you the SYSIDNT of the remote CICS region.

If the task is using intersystem communication (ISC), you need to look in field TCTTEIST of the TCTTE, which points to the ISC system table entry. The first field in the system table entry is the identity of the remote region.

When you have identified the region, you need to take a system dump of it. You can do that either by using the CEMT PERFORM DUMP command in that region, or by using the MVS DUMP command. Take a system dump of the local region, too.

Format the dumps using the formatting keyword DS to get a summary of the tasks in each region, and TCP to get the TCTTEs.

First find the TCTTE for the task in the local region. The way you find the TCTTE for the task in the remote region depends on whether you are using LUTYPE6.1 sessions (or IRC) or APPC sessions:

- For LUTYPE6.1 sessions and IRC sessions, look in local control block TCTENIB, the TCTTE extension for the NIB descriptor, at field TCTESQP. This gives you the session qualifier pair for the session. It provides the terminal ID associated with the local task, concatenated with the terminal ID associated with the remote task.

  Now go to the dump of the remote region, and use the terminal ID to locate its TCTTE. Check in field TCTESQP of TCTENIB to make sure that the session qualifier pair matches that in the local system. It should be made up of the same terminal IDs, but with their order reversed.

- For APPC sessions, look in local control block TCTTELUC, the APPC extension, in field TCTESII. Ignoring the high-order byte, this gives you the session instance identifier of the session.

  Now go to the dump of the remote region, and use the session instance identifier you have found for the remote task to locate its TCTTELUC. The TCTTE precedes the TCTTELUC in the dump.

When you have confirmed that you have located the correct TCTTE, look in field TCTTECA. This gives you the TCA address of the task that is not responding.

Using the TCA address as the entry point, you can now investigate the reason why the task has not responded. It is very likely that it has been suspended because some resource is unavailable. Look in the dispatcher and transaction manager summaries. If you can identify your task, you can see what resource it is waiting on.

When you have identified the resource, turn to the appropriate section in this chapter for guidance about investigating waits on that resource.

VTAM terminal control waits

VTAM terminal control waits are associated with three resource types. These resource types are:

- ZC_ZCGRP
- ZC_ZGCH
- ZC_ZGRP
- ZC_ZGUB
- ZCIOWAIT
- ZCZGET
- ZCZNAC

The implication of waits on any of these VTAM terminal resource types are as follows:

Resource type ZC_ZCGRP

DFHZSLS has to set the TCT prefix VTAM fields from the ACB. This wait is issued to ensure that these fields are set before being used.

Resource type ZC_ZGCH

DFHZGCH is waiting for the VTAM CHANGE ENDAFFIN macro to complete.

Resource type ZC_ZGRP, resource name PSINQECB

If the task is waiting on resource name PSINQECB, this means that DFHZGRP has issued the VTAM macro INQUIRE PERSESS during VTAM persistent session restart, or during the reopening of the VTAM ACB, and is waiting for a response from VTAM. The wait expires after 8 minutes if VTAM does not respond.

If VTAM is using Multi Node Persistent Sessions, the INQUIRE may take longer if the information is not yet available.

Resource type ZC_ZGRP, resource name PSOP2ECB

If the task is waiting on resource name PSOP2ECB, this means that DFHZGRP has issued the VTAM macro OPNDST RESTORE during emergency restart, or during the reopening of the VTAM ACB, and is waiting for a response from VTAM. The wait expires after 8 minutes if VTAM does not respond.

Resource type ZC_ZGUB

DFHZGUB issues ten VTAM CLSDST or TERMSESS macros during persistent sessions restart. It waits for an RPL to become free for VTAM to post the VTAM exit. The wait expires after 5 minutes if VTAM does not respond.

Resource type ZCIOWAIT

Suspends on resource type ZCIOWAIT occur when the task is waiting for some terminal I/O. Once the expected I/O event occurs, the task is resumed.

Resource type ZCZGET

If your task is waiting on a resource name of DFHZARL2, it is suspended by module DFHZARL which deals with application request logic for LU6.2 devices. The suspend is caused by a GETMAIN call to DFHZGET failing. DFHZGET is continually invoked until the GETMAIN is successful.

Resource type ZCZNAC

Suspends on resource type ZCZNAC are on resource names DFHZARL3 or DFHZERH4. The wait is for DFHZNAC to issue an error message. The error message to be issued depends on the error that led to the suspend. Various actions may be taken by DFHZNAC before control is returned to the suspended task.

Interregion and intersystem communication waits

If you have a user task that is waiting for resource type ALLOCATE, it has attempted to get a session with another CICS region, but all the sessions are in use. Consider defining a greater number of sessions, which should solve the problem. For guidance about this, see the *CICS/ESA Resource Definition Guide* and the *CICS/ESA Intercommunication Guide*.

If you otherwise have a problem that you have identified as an interregion or an intersystem communication wait, investigate it as for terminal waits. This is dealt with in “Terminal waits” on page 108.

The method of debugging is the same in each case. You need to consider the access method, terminal control, and the “terminal” itself.

For interregion and intersystem communication, the remote region or system *is* the terminal. Its status can be found using the same online or offline techniques that you would use to find the status of a physical terminal. The status may lead you to suspect that the task running in the remote region is the cause of the problem, and you then need to investigate why that task is waiting. So you could find that what started as a terminal wait might, after all, be a wait on some other type of resource.

Transient data waits

Tasks issuing requests to read and write to transient data destinations can be made to wait for several different reasons. The reasons depend on the type of request being made, and whether the task is attempting to access an extrapartition or an intrapartition queue.

The resource types that might be associated with the wait are:

- TD_INIT
- TDEPLOCK
- TDIPLOCK
- KC_ENQ
- Any_MBCB
- Any_MRCB
- MBCB_xxx
- MRCB_xxx

The resource name is the transient data queue name, except in the case of TD_INIT, whose resource name is DCT.

Resource type TD_INIT – waits during initialization processing

A second stage PLT program being executed during system initialization can issue a request for a resource that is not yet available, because the component that services the request has not yet been initialized. If the program issues a transient data request that cannot yet be serviced, it is suspended on a resource type of TD_INIT with a resource name of DCT.

You are unlikely to see any evidence for this type of wait, unless you have trace running during initialization with DS level-1 tracing selected. An error at this stage would be likely to cause CICS to stall (see “CICS has stalled during initialization” on page 137), or to terminate abnormally.

Resource type TDEPLOCK – waits for transient data extrapartition requests

If you have a task suspended on resource type TDEPLOCK, with a resource name corresponding to a transient data queue name, the task has issued a request against an extrapartition transient data queue. Another task is already accessing the same queue, and the waiting task cannot resume until that activity is complete.

If the wait is prolonged, it could be for either of these reasons:

- It is necessary for a task to change TCB mode to open and close a data set. The task must relinquish control while this happens, and, depending on the system loading, this might take several seconds. This contributes to the wait that the second task, suspended on resource type TDEPLOCK, experiences.
- CICS uses the access method QSAM to write data to extrapartition transient data destinations. QSAM executes synchronously with tasks requesting its services. This means that any task invoking a QSAM service must wait until the QSAM processing is complete. If, for any reason, QSAM enters an extended wait, the requesting task also experiences an extended wait.

  The possibility of an extended wait arises whenever QSAM attempts to access an extrapartition data set. QSAM uses the MVS RESERVE volume-locking mechanism to gain exclusive control of volumes while it accesses them, which means that any other region attempting to write to the same volume is forced to wait.

If tasks frequently get suspended on resource type TDEPLOCK, you need to determine which other transactions write data to the same extrapartition destination. You might then consider redefining the extrapartition destinations in the DCT (destination control table).

Resource types TDIPLOCK, KC_ENQ, Any_MBCB, Any_MRCB, MBCB_xxx, and MRCB_xxx

If your task is waiting on any of the resource types TDIPLOCK, KC_ENQ, Any_MBCB, Any_MRCB, MBCB_xxx, or MRCB_xxx, it has made a transient data intrapartition request that cannot be serviced at once. In each case, the resource name identifies the intrapartition queue that the request has been issued against.

Resource type TDIPLOCK

If you have a task suspended on resource type TDIPLOCK, with a resource name corresponding to a transient data queue name, the task has issued a request against an intrapartition transient data queue. Another task is already accessing the same queue, and the waiting task cannot resume until that activity is complete.

If tasks frequently get suspended on resource type TDIPLOCK, you need to determine which other transactions use the same intrapartition destination. You might then consider redefining the intrapartition destinations in the DCT.

You can find further guidance information about the constraints that apply to tasks writing to intrapartition destinations in the *CICS/ESA Application Programming Guide*. For more details of the properties of recoverable transient data queues, see the *CICS/ESA Resource Definition Guide*.

Resource type KC_ENQ

If a transient data queue has been defined in the DCT as intrapartition and logically recoverable, there are further restrictions (in addition to those leading to waits on resource type TDIPLOCK) on the use of the queue by more than one task at a time.

If you have a task suspended on resource type KC_ENQ, it may have issued a request against such a queue. Be aware, however, that KC_ENQ is not exclusively for transient data waits.

Changes to logically recoverable queues are not committed until the end of a logical unit of work. To ensure integrity of data, therefore, other tasks may be forced to wait until the task accessing the queue has reached syncpoint or the end of its activity. Tasks suspended on resource type KC_ENQ , where the current task is trying to access transient data, are very likely to be waiting for the task currently accessing the queue to reach syncpoint or end of task.

There are two types of KC_ENQ wait: one for reads and one for writes. A delete requires both the read and write lock. No task may, therefore, read or write to a queue while a delete is in progress. A delete cannot proceed until any task currently reading has completed its read or any task writing has committed its changes.

A task may read while another task is writing but it can only read already committed records. Uncommitted records can only be read by the task that wrote them.

Whether or not a read request waits on resource type KC_ENQ depends on the coding of the NOSUSPEND option on the READQ command. If coded, QBUSY is returned to the application and the task does not wait. If not coded, the task waits on resource type KC_ENQ.

Resource type Any_MBCB

If your task is waiting on resource type Any_MBCB, the resource name is the name of an intrapartition queue that it is attempting to access. This type of wait shows that all the transient data I/O buffers are in use, and the task resumes only when one becomes available.

Tasks are only likely to wait in this way in a heavily loaded system.

Resource type Any_MRCB

When a transient data I/O buffer has been acquired for a task, a VSAM string must be obtained. If all the VSAM strings available for transient data processing are in use, the task is suspended on resource type Any_MRCB, with a resource name equal to the intrapartition queue name.

Waits on Any_MRCB should not be prolonged, except in a heavily loaded system.

Resource type MRCB_xxx

A resource type of MRCB_xxx, with a resource name equal to an intrapartition transient data queue name, shows that the suspended task has successfully obtained a VSAM string, and is now waiting for VSAM I/O to complete. This should not be a long wait, unless operator intervention is required.

Resource type MBCB_xxx

If a task is waiting on resource type MBCB_xxx, with a resource name equal to the intrapartition queue name, this indicates contention for a transient data I/O buffer. It should not be an extended wait, although it is dependent on VSAM I/O taking place on behalf of another task that has issued a transient data request. If that, for any reason, takes a long time, the wait on resource type MBCB_xxx is correspondingly long. (For descriptions of the waits that might occur during transient data VSAM I/O processing, see “Resource type Any_MRCB” and “Resource type MRCB_xxx”).

The reason for this type of wait is best illustrated by example, as follows:

1. Task #1 issues a transient data request that requires access to an intrapartition queue. Before the request can be serviced, task #1 must be assigned a transient data I/O buffer that is not currently being used by any other task.

   I/O buffers each contain a copy of a control interval (CI) from a data set. Each CI contains records that correspond to elements in an intrapartition queue. A search is made to see if the CI required for task #1 is already in one of the I/O buffers. If it is, that I/O buffer can be used to service the request made by task #1, and no VSAM I/O is involved. If it is not, task #1 is allocated any buffer, so the required CI can be read in. The current contents of the buffer is overwritten.

   An I/O buffer can have a R/O (read only) status or a R/W (read/write) status. If the buffer that is allocated to task #1 has R/W status, it contains a copy of a CI that has been updated by some other task, but not yet written back to the data set. Before the buffer can be used by task #1, the CI it contains must be preserved by writing it back to the data set.

2. A request now arrives from task #2, and the request requires the CI that is currently being written to the data set. No two buffers can contain the same CI, so task #2 is made to wait on resource type MRCB_xxx until the outcome of the VSAM I/O is known.

   If VSAM I/O was successful, task #2 is resumed and assigned some other I/O buffer.

   If VSAM I/O was unsuccessful, task #2 can use the I/O buffer that already contains the CI it needs.

Loader waits

A task is suspended by the loader domain if it has requested a program load and another task is already loading that program. Once the load in progress is complete, the suspended task is resumed very quickly and the wait is unlikely to be detected.

Note that the loader *does not* suspend a task while a program is loaded if it is the first one to ask for that program.

If the requested program is not loaded quickly, the reasons for the wait need to be investigated. The possible reasons for the wait, and the ways you should investigate them are:

1. The system could be short on storage (SOS), so only system tasks can be dispatched.

   To check if the system is short on storage use the CEMT transaction, submitting CEMT I SYS SOSSTATUS. To see if SOS has been reached too often, examine the job log, check the run statistics, or submit CEMT I DSAS. If SOS has been reached too often, take steps to relieve the storage constraints. For guidance about this, see the *CICS/ESA Performance Guide*.

2. There could be an I/O error on a library.

   Check for messages that might indicate this. If you find one, investigate the reason why the I/O error occurred.

3. There could be an error within MVS.

   Has there been any sort of message to indicate this? If so, it is likely that you need to refer the problem to the IBM Support Center.

File control waits

Most file control waits are associated with resource types starting with the characters FC. Some are associated with resource type KC_ENQ, but KC_ENQ is not used exclusively for file control waits.

Here is a list of identifiable resource types associated with file control waits, and all the possible reasons for waits:

- FCBFWAIT. The task is waiting for a VSAM buffer.
- FCDWWAIT. The task is waiting for VSAM upgrade set activity to complete.
- FCFSWAIT. The task is waiting to change the state of a file.
- FCIOWAIT. The task is waiting for I/O on a disk volume.
- FCPSWAIT. The task is waiting for a private string.
- FCRBWAIT. The task is waiting because file recovery failed to complete.
- FCSRSUSP. The task is waiting for a shared resource string.
- FCTISUSP. The task is waiting for a VSAM transaction ID.
- FCXCWAIT. The task is waiting for exclusive control of a VSAM control interval.
- KC_ENQ. The task is waiting for a record lock in a recoverable VSAM file.
- KC_ENQ. The task is waiting for exclusive control of a record in a BDAM file.

The implications of waits on any of these file control resource types are dealt with in the sections that follow.

Resource type FCBFWAIT – waits for VSAM buffers

If your task is waiting on resource type FCBFWAIT, it means that a VSAM buffer is not currently available. You can specify the number of VSAM data buffers and VSAM index buffers in the FILE resource definition using the DATABUFFERS and INDEXBUFFERS parameters, respectively.

Consider increasing the numbers of these buffers if you find that tasks are frequently having to wait on this resource type.

If there are insufficient data and index buffers for a single task, the task is suspended indefinitely. This might happen unexpectedly if you have a base cluster and one or more paths in the upgrade set, and your application references only the base. VSAM upgrades the paths whenever changes are made to the base. There could then be too few buffers defined in the LSRPOOL for both base and paths.

Resource type FCDWWAIT – wait for VSAM upgrade set activity

If your task is waiting on resource type FCDWWAIT, it has received a VSAM response which might indicate that your task is trying to read a record via a VSAM path while this record is being updated by another request. This other request is updating the record either via the base or via another path. If VSAM has not yet completed the update, the content of the alternate index currently in use is no longer the same as the content of the base data set.

This is a transient condition. CICS waits for all current update operations for this VSAM data set to complete and retries the request once only. If the error continues after the request is retried, CICS assumes that there is a genuine error and returns a response of ILLOGIC to the application. Since ILLOGIC is a response to all unexpected VSAM errors, CICS also returns the VSAM response and reason codes (X'0890') in bytes 2 and 3 of EIBRCODE. These identify the cause of the ILLOGIC response.

Resource type FCFSWAIT – waits for file state changes

If your task is waiting on resource type FCFSWAIT, it means that it has attempted to change the state of a file—for example, to CLOSE or DISABLE it—but another task is still using the file. This can happen, for example, if a long-running transaction, possibly conversational, is using a recoverable file. The long-running transaction has the lock on the file, and might not release it until it terminates. In such a case, consider changing the programming logic so that the file is released more quickly.

Only one task at a time waits on FCFSWAIT. If any other tasks attempt to change the state of the same file, they are suspended on resource type KC_ENQ. See “Task control waits” on page 101.

Resource type FCIOWAIT – wait for VSAM I/O

If you have a task waiting on resource type FCIOWAIT, it means that the task is waiting ***within*** VSAM for I/O to take place. VSAM uses MVS RESERVE volume locking, and it is likely that another job has at present got the lock on the volume. See if there are any messages on the MVS console to explain the error.

A wait on resource type FCIOWAIT occurs when the exclusive control conflict is deferred internally by VSAM and not returned as an error condition to CICS. An example of this is when a request against an LSR file is made for exclusive control of a control interval (for example, by WRITE or READ UPDATE) and either this task or another task already holds shared control of this control interval (for example, by STARTBR).

Exclusive control waits are discussed further in “Resource type FCXCWAIT – VSAM exclusive control wait” on page 125.

Resource types FCPSWAIT and FCSRSUSP – waits for VSAM strings

If your task is waiting on either of resource types FCPSWAIT or FCSRSUSP, it means that it cannot get a VSAM string. FCPSWAIT shows that the wait is for a private string, and FCSRSUSP shows that the wait is for a shared resource string. You can purge the task from the system, if the task is purgeable.

The number of strings defined for a VSAM data set (STRINGS parameter in the FILE resource definition) determines how many tasks can use the data set concurrently. STRINGS can have a value in the range 1–255. When all the strings are in use, any other task wanting to access the data set must wait until a string has been released.

If tasks are being caused to wait unduly for strings, consider whether you can increase the value of STRINGS, or change the programming logic so that strings are released more quickly.

An example of programming logic that can hold onto strings (and other VSAM resources) for too long is when a conversational transaction issues a STARTBR or READNEXT and then enters a wait for terminal input without issuing an ENDBR. The browse remains active until the ENDBR, and the VSAM strings and buffers are retained over the terminal wait. Also, for an LSR file, the transaction continues to hold shared control of the control interval and causes transactions that attempt to update records in the same control interval to wait.

Similarly, transactions hold VSAM resources for too long if a READ UPDATE or WRITE MASSINSERT is outstanding over a wait for terminal input.

Resource type FCRBWAIT – file recovery failed to complete

If your task is waiting on FCRBWAIT, it means that file recovery failed to complete. Refer the problem to the IBM Support Center.

Resource type FCTISUSP – wait for a VSAM transaction ID

If your task is waiting on resource type FCTISUSP, it means that there are no VSAM transaction IDs available. Transaction IDs are retained by a task for the duration of a MASSINSERT session.

Waits on FCTISUSP should not be prolonged, and if your task stays suspended on this resource type, it could indicate any of the following:

- There could be a system-wide problem. CICS could have stopped running, or it might be running slowly. Turn to Chapter 2, “Classifying the problem” on page 11 for advice if you suspect this.
- There could be a performance problem. Guidance about dealing with performance problems is given in Chapter 8, “Dealing with performance problems” on page 153.
- The logic of your applications might need changing, so that tasks don’t retain VSAM transaction IDs for too long. If the task does other processing during the session, perhaps even involving input from an operator, code to release the VSAM transaction ID should be included each time.

Resource type FCXCWAIT – VSAM exclusive control wait

If your task is waiting on resource type FCXCWAIT, it means that it cannot get exclusive control of a VSAM control interval at the present time. Another task already has shared or exclusive control of the control interval, so your task is suspended pending the release of that control interval. An exclusive control wait on resource type FCXCWAIT occurs within CICS, unlike the similar wait on FCIOWAIT, which occurs within VSAM. See page 124.

If you find that exclusive control conflicts occur too often in your system, consider changing the programming logic so that applications are less likely to have exclusive control for long periods.

The possibility that a task is deadlocked, waiting on itself or another task for release of the control interval, is dealt with in the next section.

Exclusive control deadlock

Prior to CICS/ESA 4.1, a task could be made to wait on itself for exclusive control of a VSAM control interval. The task would be deadlocked, and could neither release exclusive control nor reacquire it.

Alternatively, a task could be made to wait on another task which has exclusive or shared control of a VSAM control interval. If this second task was, itself, waiting for exclusive control of a resource of which the first task has exclusive or shared control, then both tasks would be deadlocked.

At CICS/ESA 4.1, however, a mechanism exists to avoid exclusive control deadlock. If a task is waiting on resource type FCXCWAIT and causing the task to wait, causing a deadlock, the task is abended either with abend code AFCF or AFCG at the time that it makes the request for exclusive control.

A task that is abended with abend code AFCF would have been waiting for exclusive control of a VSAM control interval of which another task has shared or exclusive control.

A task that is abended with abend code AFCG would have been waiting for exclusive control of a VSAM control interval of which it has shared control.

See the *CICS/ESA Messages and Codes* manual for further details of these abend codes.

To resolve the problem, you must determine which program caused the potential deadlock. Find out which programs are associated with the abended task, and attempt to find the one in error. It is likely to be one that provides successive browse and update facilities. When you have found the programs associated with the task, turn to “How tasks can become deadlocked waiting for exclusive control” for guidance about finding how the error might have occurred.

How tasks can become deadlocked waiting for exclusive control

Tasks can become deadlocked waiting for exclusive control of a CI only when they have shared control of the CI and then attempt to get exclusive control without relinquishing shared control first. This can only occur for VSAM shared resource data sets.

For the deadlock to occur, a transaction must first issue a VSAM READ SEQUENTIAL request via EXEC CICS STARTBR. This is a VSAM “shared control” operation. It must then issue some VSAM request requiring exclusive control of the CI without first ending the shared control operation.

The requests that require exclusive control of the CI are:

- VSAM READ UPDATE, via EXEC CICS READ UPDATE and, subsequently, EXEC CICS REWRITE.

  Exclusive control of the CI is not acquired until after the initial read is complete, but it happens automatically after that and the CI is not released until the record has been rewritten.
- VSAM WRITE DIRECT, via EXEC CICS WRITE.
- VSAM WRITE SEQUENTIAL, via EXEC CICS WRITE MASSINSERT.

VSAM handles requests requiring exclusive control on a data set that is already being used in shared control mode by queueing them internally. VSAM returns control to CICS, but transactions waiting for exclusive control remain suspended.

Example of code causing an exclusive control deadlock

The following sequence of EXEC commands would cause an exclusive control deadlock to occur.

The first command causes shared control to be acquired:

```
EXEC CICS STARTBR
     FILE(myfile)
     RIDFLD(rid-area)
```

This causes no problems. The next command at first acquires shared control while the record is read into “input-area”. When an attempt is subsequently made to get exclusive control, deadlock occurs because the task that wants exclusive control is also the task that is preventing it from being acquired.

```
EXEC CICS READ
     FILE(myfile)
     INTO(input-area)
     RIDFLD(rid-area)
     UPDATE
```

The following sequence of commands would not cause deadlock to occur, because the transaction relinquishes its shared control of the CI by ending the browse before attempting to get exclusive control of it.

The first command causes shared control to be acquired:

```
EXEC CICS STARTBR
     FILE(myfile)
     RIDFLD(rid-area)
```

The next command causes shared control to be relinquished:

```
EXEC CICS ENDBR
     FILE(myfile)
```

The next command initially causes shared control to be acquired. The record is read into “input-area”, and then exclusive control is acquired in place of shared control.

```
EXEC CICS READ
     FILE(myfile)
     INTO(input-area)
     RIDFLD(rid-area)
     UPDATE
```

The transaction now resumes. Exclusive control is relinquished following the next REWRITE or UNLOCK command on file “myfile”.

Resource type KC_ENQ – wait for a record lock in a recoverable VSAM file

When an application updates a record in a recoverable VSAM file, locking occurs at two levels. VSAM locks the CI when the record has been read, and CICS locks the record.

The CI lock is released as soon as the REWRITE (or UNLOCK) request is completed. However, the record is not unlocked by CICS until the updating transaction has reached a syncpoint. This is to ensure that data integrity is maintained if the transaction fails before the syncpoint and the record has to be backed out.

If a second transaction attempts to update the same record while it is still locked, it is suspended on resource type KC_ENQ until the lock is released. This can be a long wait, because the update might depend on a terminal operator typing in data. Also, the suspended transaction relinquishes its VSAM string and, perhaps, its exclusive control of the CI, and has to wait once more for those resources.

If transactions are commonly made to wait for this reason, you should review the programming logic of your applications to see if the record-locking time can be minimized.

Note that CICS only locks the recoverable record for update. Other transactions are allowed to read the record, and this presents a potential read integrity exposure. Thus, a transaction might read a record after an update has been made, but before the updating transaction has reached its syncpoint. If the reading transaction takes action based on the value of the record, the action is incorrect if the record has to be backed out.

There is some more information about read integrity in Chapter 9, “Dealing with incorrect output” on page 163.

Resource type KC_ENQ – wait for exclusive control of a BDAM record

BDAM does not use the “control interval” concept. When a task reads a record for update, the record is locked so that concurrent changes cannot be made by two transactions. The lock is released at the end of the current logical unit of work. If a second task attempts to update the same record while the first has the lock, it is suspended on resource type KC_ENQ.

Interval control waits

Read this section if you have a task that is not running, and interval control seems to be involved. The following is a list of possible cases, and suggestions to consider before you carry out a detailed investigation. If these do not give you enough information in order to solve the problem, turn to “Finding the reason for a DELAY request not completing” on page 129 for further guidance.

If, in the course of your preliminary investigations, you find that the task is waiting because the terminal where it is due to start is unavailable, turn to “Terminal waits” on page 108.

- A terminal task that should have been initiated with an EXEC CICS START command did not start when you expected it to. CEMT INQ TASK does not recognize the task, because it has not yet been attached.

  One approach is to identify the terminal where the subject task should have started, and see if that terminal is, for some reason, unavailable. You can use CEMT INQ TERMINAL to find the status of the terminal.

- You have found that a task is waiting on resource type ICGTWAIT. This means that the task has issued an EXEC CICS RETRIEVE WAIT command,

and the data to be retrieved is, for some reason, not available. The resource name gives you the name of the terminal running the task in the ICGTWAIT wait and therefore the target TERMID for other tasks issuing EXEC CICS START commands to supply more data. If there are no tasks in the system that would issue START commands for this TERMID, you need to determine whether this is reasonable. If there are such tasks in the system, check to see why they are not issuing the required START commands. They might, for example, be waiting for terminal input.

Look, too, at the deadlock time-out interval (DTIMOUT) and the system purge value (SPURGE) for the task issuing the EXEC CICS RETRIEVE WAIT command. If there is no DTIMOUT value or SPURGE=NO has been specified, the task will wait indefinitely for the data.

Note: The task waiting on resource ICGTWAIT might not be the one that you first set out to investigate. Any AID task scheduled to start at the same terminal cannot do so until the current task has terminated.

- You have found that the task is waiting on resource type ICWAIT. This means that the task issued an EXEC CICS DELAY command that has not yet completed. Check that the interval or time specified on the request was what you intended. If you believe that the expiry time of the request has passed, that suggests a possible CICS error.

  Consider, too, the possibility that the task was the subject of a long DELAY that was due to be canceled by some other task. If the second task failed before it could cancel the delay, the first would not continue until the full interval specified on DELAY had expired.

- A task that issued EXEC CICS POST did not have its ECB posted when you expected it to.

  Check to make sure the interval or time you specified was what you intended.

- A task that issued EXEC CICS WAIT EVENT was not resumed when you thought it should have been.

  Assuming the WAIT was issued sometime after a POST, first check to make sure that the interval or time specified on the POST was what you intended. If it was, next check to see whether the ECB being waited on was posted. If it was, that indicates a possible CICS error.

If none of the simple checks outlined here help you to solve the problem, read the next section.

Finding the reason for a DELAY request not completing

If your preliminary investigations have not shown the reason for the wait, you need to look in greater detail at the evidence available. Take a system dump, and format it using the keywords CSA, ICP, and AP. These get you the common system area, the interval control program control blocks, and the task control areas, respectively. You might also find information given by the formatting keywords KE (kernel storage areas, including the calling sequence for each task), DS (dispatcher task summary, including details of suspended tasks), and TR (internal trace table) to be useful.

First, locate field CSATODTU in the CSA. Make a note of the value there, which is the current CICS time of day in internal ‘timer units’. Now locate the TCA for your task, and read the value of field TCAICEAD. This gives you the address of the

interval control element for your task. Use this information to find the ICE (interval control element) for the task, and look at field ICEXTOD. Make a note of the value there.

If ICEXTOD is greater than CSATODTU, the ICE has not yet reached the expiry time. These are the possible explanations:

- Your task either did not make the DELAY request you expected, or the interval specified was longer than intended. This could indicate a user error. Check the code of the transaction issuing the request to make sure it is correct.
- Your task's delay request was not executed correctly. This might indicate an error within CICS code, or a corrupted control block.

If ICEXTOD is equal to CSATODTU (very unlikely), you probably took the system dump just as the interval was about to expire. In such a case, attempt to re-create the problem, take another system dump, and compare the values again.

If ICEXTOD is less than CSATODTU, the ICE has already expired. The associated task should have resumed. This indicates that some area of storage might have been corrupted, or there is an error within CICS code.

Using trace to find out why tasks are waiting on interval control

Before using trace to find out why your task is waiting on interval control, you need to select an appropriate trace destination and set up the right tracing options.

By their nature, interval control waits can be long, so select auxiliary trace as the destination, because you can specify large trace data sets for auxiliary trace. However, the data sets do not have to be large enough to record tracing for the whole interval specified when you first detected the problem. That is because the error is likely to be reproducible when you specify a shorter interval, if it is reproducible at all. For example, if the error was detected when an interval of 20 seconds was specified, try to reproduce it specifying an interval of 1 second.

As far as tracing selectivity is concerned, you need to capture level-2 trace entries made by dispatcher domain, timer domain, and interval control program. Use the CETR transaction to set up the following tracing options:

1. Specify special tracing for the level-2 trace points for components DS (dispatcher domain), TI (timer domain), and IC (interval control program).
2. Select special tracing for the task causing the problem, by specifying special tracing both for the transaction and for the terminal where it is to be run.
3. Set the master system trace flag off, to turn off all standard tracing. This helps minimize the number of trace entries not connected with the problem.
4. Make sure that auxiliary tracing is active, then set the transaction running. When the problem appears, format the auxiliary trace data set and either print it or view it online.

The sort of trace entries that you can expect in normal operation are shown in the figures that follow. They show the flow of data and control following execution of the command EXEC CICS DELAY INTERVAL(000003). A similar set of trace entries would be obtained if TIME had been specified instead of INTERVAL, because TIME values are converted to corresponding INTERVAL values before timer domain is called.

Figure 11 shows the first two entries that you get following execution of the EXEC CICS DELAY INTERVAL(000003) command.

```
AP 00E1 EIP ENTRY DELAY                              REQ(0004) FIELD-A(0034BD70 ....) FIELD-B(08001004 ....)
             TASK-00163 KE_NUM-0007 TCB-009F3338 RET-8413F43E TIME-16:31:58.0431533750 INTERVAL-00.0000166250    =000602=
AP 00F3 ICP ENTRY WAIT                               REQ(2003) FIELD-A(0000003C ....) FIELD-B(00000000 ....)
             TASK-00163 KE_NUM-0007 TCB-009F3338 RET-84760B88 TIME-16:31:58.0432681250 INTERVAL-00.0000370000    =000605=
```

Figure 11. Trace entries following EXEC CICS DELAY INTERVAL(000003) invocation

Notes:

1. Trace point **AP 00E1** is on ENTRY to the EIP DELAY routine. The function is stated in the trace header, and the fact that this trace is made on ENTRY can be deduced from the value shown in the request field, REQ(0004).

   The rightmost two bytes of FIELD B give the EIBFN value, in this case X'1004'. This shows that this is an interval control DELAY request.

   The value shown against TASK is the trace task number, and it is unique to the task while the task is in the system. Its purpose is to show which trace entries relate to which tasks. The task number in this example is 00163. As long as the task is in the system, and either running or suspended, trace entries having this task number always relate to it. Use the task number for your task to identify the trace entries associated with it.

2. Trace point **AP 00F3** is on ENTRY to the ICP WAIT routine. The function is given explicitly in the trace header, and both the function and the fact that this represents ENTRY to the routine can be deduced from the request field, REQ(2003).

   The value of FIELD A, X'0000003C', is an important one for problem determination. It shows the interval that has been specified, in this case three seconds. Check the value shown here for your own task, to make sure it is what you expect it to be.

Look next for an entry with point ID DS 0004 showing your task being suspended, as in Figure 12. You might see TI domain trace entries preceding it that show entry and exit for FUNCTION(REQUEST_NOTIFY_INTERVAL), but these don't always appear.

There might also be some intervening entries, but they are unlikely to be of relevance to the problem.

```
TI 0100 TISR  ENTRY - FUNCTION(REQUEST_NOTIFY_INTERVAL) DOMAIN_TOKEN(00E70000 , 00000000) STCK_INTERVAL(00000002DC6C1000)
                  PERIODIC_NOTIFY(NO) NOTIFY_TYPE(TIMER_TASK)
              TASK-00163 KE_NUM-0007 TCB-009F3338 RET-8476352A TIME-16:31:58.0442390000 INTERVAL-00.0000155000    =000614=
                1-0000  00600000 00000006 00000000 00000000  B3B00000 00000000 01000000 00000000  *.-.............................*
                  0020  00000000 00000000 00000000 00E70000  00000000 00000000 00000000 00000002  *.............X.................*
                  0040  DC6C1000 00000000 02020000 00000000  00000000 00000000 00000000 00000000  *.%.............................*
TI 0101 TISR  EXIT  - FUNCTION(REQUEST_NOTIFY_INTERVAL) RESPONSE(OK) TIMER_TOKEN(03B9B058 , 0000001B)
              TASK-00163 KE_NUM-0007 TCB-009F3338 RET-8476352A TIME-16:31:58.0738898750 INTERVAL-00.0296188125*   =000617=
                1-0000  00600000 00000006 00000000 00000000  B3B00000 00000000 01000100 00000000  *.-.............................*
                  0020  00000000 00000000 00000000 00E70000  00000000 03B9B058 0000001B 00000002  *.............X.................*
                  0040  DC6C1000 00000000 02020000 00000000  00000000 00000000 00000000 00000000  *.%.............................*
DS 0004 DSSR  ENTRY - FUNCTION(SUSPEND) SUSPEND_TOKEN(01040034) RESOURCE_NAME(1477) RESOURCE_TYPE(ICWAIT) PURGEABLE(YES)
                  DEADLOCK_ACTION(INHIBIT)
              TASK-00163 KE_NUM-0007 TCB-009F3338 RET-847645CE TIME-16:31:58.0739336250 INTERVAL-00.0000437500    =000618=
                1-0000  00580000 00000014 00000001 00000000  B7050000 00000000 04000100 00000000  *.............................*
                  0020  00000000 01040034 F1F4F7F7 40404040  C9C3E6C1 C9E34040 0000001B 00000002  *........1477    ICWAIT  ........*
                  0040  DC6C1000 00000000 02010003 00000000  00000000 00000000                   *.%.....................       *
```

Figure 12. Trace entries showing interval calculation and task suspension

Notes:

1. Trace point **TI 0100**, if shown, is on ENTRY to the REQUEST_NOTIFY_INTERVAL function of timer domain. This is stated explicitly in the trace header.

   The value shown in the header for STCK_INTERVAL is derived from the machine store clock value calculated for the DELAY interval specified on the EXEC CICS DELAY command. You can find out how store clock values are related to times in hours, minutes, and seconds from the *ESA/370 Principles of Operation* manual.

   If you do the calculation, you find that the value shown is not exactly equal to the interval you specified. An extra microsecond is added, to account for the case where the interval is specified as 0.

   In this example, 3 seconds is exactly equal to a store clock interval of X'00000002DC6C0000'. You can see that the actual store clock value is quoted in the trace entry as X'00000002DC6C1000', which is 3 seconds *plus* 1 microsecond.

   The TIME field of the trace entry shows the time at which the entry was made, in the format hh:mm:ss. The value in this example (ignoring the fractions of a second) is 16:31:58. It follows that the task is due to be resumed when the time is 16:32:01, because the interval is 3 seconds.

2. Trace point **TI 0101**, if shown, is on EXIT from the REQUEST_NOTIFY_INTERVAL function of timer domain.

   You can see from RESPONSE(OK) in the header that the function completed normally.

3. Trace point **DS 0004** is on ENTRY to the dispatcher task SUSPEND/RESUME interface.

   The SUSPEND_TOKEN field in the trace header is significant. It shows the unique suspend token being used for this SUSPEND/RESUME dialog, and it is referred to explicitly again in a later trace entry showing that the task has been resumed. In this example, the suspend token is X'01040034'.

   Any subsequent dispatcher trace entry that shows the suspend token for your task is connected with the suspension or resumption of the task.

   Field RESOURCE_TYPE(ICWAIT) in the trace header shows that the resource type associated with this suspend is ICWAIT. ICWAIT is the resource type that is returned on CEMT INQ TASK for tasks that are waiting on interval control.

Next, get some trace entries recording system activity during the period when your task is suspended. There are likely to be relatively few at the level of tracing detail you have specified, but you need to look further on in the trace to find the next entries of interest.

Add 3 seconds (or whatever interval you specified) to the time shown on the last trace entry you looked at, and turn forward to the trace entries made at around that time. Now look for an entry made from trace point DS 0004. *This does not show the task number for your task*, but it does show its suspend token. When you have found it, go back one entry. You should find there a trace entry made from trace point AP F322. This and the following two trace entries of interest are shown in Figure 13 on page 133.

```
AP F322 APTIX RESUMED - SYSTEM TASK APTIX RESUMED
              TASK-00006 KE_NUM-0009 TCB-009F3338 RET-84773724 TIME-16:32:01.1016870625 INTERVAL-00.0001065000    =000670=
                1-0000  01000000 D7C5D5C4 D5D6E3D7 01107739  00E70000 00000000 03B9B058 0000001B  *....PENDNOTP.....X.............*
                  0020  01080002 00D40000 00000000 03B9B000  00000001 01050002 00000000 00000000  *.....M.........................*
                  0040  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
                  0060  00000000 00000000 00000000 00000000  00000000 00000000 00000000 00000000  *................................*
DS 0004 DSSR  ENTRY - FUNCTION(RESUME) SUSPEND_TOKEN(01040034)
              TASK-00006 KE_NUM-0009 TCB-009F3338 RET-847646D4 TIME-16:32:01.1019761875 INTERVAL-00.0000278125    =000674=
                1-0000  00580000 00000014 00000001 00000000  B4000000 00000000 05000100 00000000  *................................*
                  0020  00000000 01040034 00000000 00E70000  00000000 03B9B058 0000001A 000026EE  *.............X..................*
                  0040  D9AC1000 00000000 00000000 0001632C  00000000 00000000                    *R.....................        *
DS 0005 DSSR  EXIT - FUNCTION(RESUME) RESPONSE(OK)
              TASK-00006 KE_NUM-0009 TCB-009F3338 RET-847646D4 TIME-16:32:01.1019959375 INTERVAL-00.0000197500    =000675=
                1-0000  00580000 00000014 00000001 00000000  B4000000 00000000 05000100 00000000  *................................*
                  0020  00000000 01040034 00000000 00E70000  00000000 03B9B058 0000001A 000026EE  *.............X..................*
                  0040  D9AC1000 00000000 00000000 0001632C  00000000 00000000                    *R.....................        *
```

Figure 13. Trace entries showing your task being resumed

Notes:

1. Trace point **AP F322** is used to report that system task APTIX has been resumed. APTIX has the job of “waking up” your task on expiration of the specified interval.

   The task number for APTIX is, in this case, X'00006', and this value is shown on the trace entry.

2. Trace point **DS 0004** is on entry to the dispatcher SUSPEND/RESUME interface. This function is stated explicitly in the header.

   TASK-00006 indicates that the trace entry is for system task APTIX.

   SUSPEND_TOKEN(01040034) shows that APTIX is requesting dispatcher domain to resume the task that was suspended for the specified interval. You will recall that a suspend token of X'01040034' was given to your task when it was first suspended.

3. Trace point **DS 0005** is on exit from the dispatcher SUSPEND/RESUME interface.

   The trace entry shows RESPONSE(OK), indicating that the task whose suspend token was X'01040034' has successfully been resumed. However, note that this does not necessarily mean that the task has started to run—it has only been made “dispatchable”. For example, it still needs to wait for a TCB to become available.

Now look forward in the trace, and locate a trace entry made from trace point AP 00F3 and showing your task number. This and the next entry conclude the DELAY request for your task. They are shown in Figure 14.

```
AP 00F3 ICP EXIT NORMAL                              REQ(0005) FIELD-A(01000300 ....) FIELD-B(03BD6EE0 ..>.)
              TASK-00163 KE_NUM-0007 TCB-009F3338 RET-84760B88 TIME-16:32:01.1023045625 INTERVAL-00.0000154375    =000688=
AP 00E1 EIP EXIT DELAY OK                            REQ(00F4) FIELD-A(00000000 ....) FIELD-B(00001004 ....)
              TASK-00163 KE_NUM-0007 TCB-009F3338 RET-8413F43E TIME-16:32:01.1024153750 INTERVAL-00.0000235625    =000691=
```

Figure 14. Trace entries showing satisfactory conclusion of the DELAY request

Notes:

1. Trace point **AP 00F3** is on EXIT from interval control program. Field REQ(0005) shows that this is so, and it also shows that the response was normal. Anything other than a normal response would result in a value other than X'00' for the first byte of the REQ field.
2. Trace point **AP 00E1** is on EXIT from the EXEC interface program. This is shown by bits 0–3 of the second byte of the REQ value, X'F4'.

   The values shown for FIELD A and FIELD B show that no exception condition was detected.

That is the end of the DELAY processing, and the task that was suspended should have been resumed.

When you look at your own trace table, be concerned principally with finding the point at which the processing went wrong. Also, watch for bad parameters. If you do find one, it could mean that an application has a coding error, or some field holding a parameter has been overlaid, or an error has occurred in CICS code.

Checking your application code is the easiest option you have. If you find that it is correct and you suspect a storage violation, refer to Chapter 10, “Dealing with storage violations” on page 189. If you think the error is in CICS code, contact the IBM Support Center.

XRF alternate system waits

The XRF takeover process involves several operations. Before each operation can be started, one or more events must have completed. For example:

- Terminals with backup sessions can be switched while the active system is running, provided the CICS availability manager (CAVM) has initiated a takeover.
- Passively-shared data sets must not be opened until it is known that the active system has terminated.

  Note: This is the only way an alternate system can be sure that no more data will be written by an active system.
- Resource managers, such as transient data, temporary storage, and database recovery control (DBRC), rely on the time-of-day clock providing them with a nondecreasing value to ensure the proper management of their resources. The alternate system must not restart a resource manager until the alternate time-of-day clock has been synchronized with the active time-of-day clock.

A system task that issued a takeover request to CAVM waits on the ECB WCSTCECB in the CAVM static control block (DFHWCGPS) until CAVM has decided to accept or reject the request. The DFHKC TYPE=WAIT and DCI=SINGLE requests are issued in DFHWSRTR. The CAVM TCB posts WCSTCECB in either DFHWSTKV (the normal case) or DFHWSSOF (CAVM failure).

The following ECBs each represent an event. The ECBs are located in the static storage for DFHXRP. The ECBs and the events are:

- XRSTIECB – the CAVM has initiated a takeover.

- XRSIAECB – the alternate system is now the incipient active system.
- XRSTCECB – the active system is known to have terminated.
- XRSRAECB – DFHRSD is available for use by DFHCC (catalog control) and DFHRC (recovery control) macros.
- XRSSSECB – the time-of-day clock is synchronized with active system sign off.
- XRSSTECB – the time-of-day clock is synchronized with active system termination.

XRSTIECB

This ECB is posted by DFHXRA, following a successful call to the CAVM to initiate takeover. Once the ECB has been posted, DFHXRA attaches a system transaction to initiate the switch of terminals with backup sessions. DFHXRA is called from either the surveillance task (DFHXRSP), or the console communication task (DFHXRCP). No tasks wait for XRSTIECB to be posted.

XRSIAECB

The XRSIAEB ECB is posted by DFHXRA, following notification by the CAVM that an alternate system is now the incipient active system. DFHXRA is called from the surveillance task (DFHXRSP). No tasks wait for XRSIAECB to be posted.

XRSTCECB

The XRSTCECB ECB is posted by DFHXRA, following notification by the CAVM that an active system has terminated. There can be a delay in posting the ECB if:

- An SDUMP is being taken as part of the active system termination process.
- In a 2-CEC environment, the active CEC has failed, and the operator failed to reply to the messages sent to the console.

DFHXRA is called from the surveillance task (DFHXRSP). Only one task, the system initialization task (DFHSII1), waits for XRSTCECB to be posted. When the ECB is posted, DFHSII1 opens the restart data set, for DFHRC use as well as for DFHCC use, and then calls DFHXRA to post the XRSRAECB.

XRSRAECB

The XRSRAECB ECB is posted by DFHXRA once the restart data set has been opened, for DFHRC use as well as for DFHCC use. DFHXRA is called from the system initialization task (DFHSII1). Two tasks wait for XRSRAECB to be posted:

- The transient data recovery task (DFHTDRP) initializes the entry for the CXRF queue before waiting for XRSRAECB to be posted. When the ECB is posted, DFHTDRP resumes emergency restart processing.
- The terminal control recovery task (DFHTCRP) drains its tracking queue before waiting for XRSRAECB to be posted. When the ECB is posted, DFHTCRP resumes emergency restart processing.

XRSSSECB

The XRSSSECB ECB is posted by DFHXRA following notification by the CAVM that the time-of-day clock is synchronized with active sign off. DFHXRA is called from the surveillance task (DFHXRSP). No tasks wait for XRSSSECB to be posted.

XRSSTECB

The XRSSTECB ECB is posted by DFHXRA, following notification by the CAVM that the time-of-day clock is synchronized with respect to active system termination. There may be a delay in posting the ECB if the time indicated by the active system time-of-day clock is significantly ahead of that indicated by the alternate system time-of-day clock. DFHXRA is called from the surveillance task (DFHXRSP).

Only the system initialization task, DFHSII1, waits for XRSSTECB to be posted. When the ECB is posted, DFHSII1 links to DFHJRCP to restore journal states before attaching the remaining resource manager tasks.

You are only likely to find either of the CICS-supplied transactions CEDA or CESN waiting on a resource type of XRPUTMSG, and only during XRF takeover by the alternate CICS system. It can indicate either of these conditions:

- Data that is required by the transactions is held on a volume subject to MVS RESERVE locking, and another job currently has the lock.
- There is an error in the CICS availability manager.

If it seems clear that MVS RESERVE locking is not implicated, refer the problem to the IBM Support Center.

CICS system task waits

From an analysis of trace, you could have evidence that a CICS system task is in a wait state. You might have seen the task suspended on a SUSPEND call to the dispatcher, but with no corresponding RESUME call. Alternatively, by looking at the dispatcher task summary in a formatted CICS system dump, you might see that a CICS system task is waiting.

Note: You cannot get online information about waiting system tasks from CEMT INQ TASK or EXEC CICS INQUIRE TASK.

If a system task is in a wait state, and there is a system error preventing it from resuming, contact your IBM Support Center. However, do not assume that there is a system error unless you have other evidence that the system is malfunctioning. Other possibilities are:

- Some system tasks are intended to wait for long periods while they wait for work to do. Module DFHSMSY of storage manager domain, for example, can stay suspended for minutes, or even hours, in normal operation. Its purpose is to clean up storage when significant changes occur in the amount being used, and that might happen only infrequently in a production system running well within its planned capacity.
- System tasks perform many I/O operations, and they are subject to constraints like string availability and volume and data set locking. In the case of tape volumes, the tasks can also be dependent on operator action while new volumes are mounted.

If, in addition to the waiting system task, you think you have enough evidence that shows there is a system error, contact your IBM Support Center.

FEPI waits

This section outlines the CICS waits that FEPI issues. Table 22 shows the points at which FEPI issues CICS waits:

Table 22. FEPI waits

Resource name	Resource type	Wait type	Description
FEPI_RQE	ADAPTER	WAIT_MVS	Issued in the FEPI adapter when a FEPI command is passed to the Resource Manager for processing. Ends when the Resource Manager has processed the request.
SZRDP	FEPRM	WAIT_MVS	Issued in the FEPI Resource Manager when it has no work to do. Ends when work arrives (from either the FEPI adapter or a VTAM exit).

It is possible for a FEPI_RQE wait to be outstanding for a long time—such as when awaiting a flow from the back-end system that is delayed due to network traffic. It is recommended that you do *not* cancel tasks that are waiting at this point; to do so could lead to severe application problems.

An SZRDP wait is generated when the FEPI Resource Manager is idle. Consequently, the SZ TCB is also inactive. On lightly loaded systems, this occurs frequently.

If the Resource Manager abends, then any active CICS FEPI transactions are left waiting on the FEPI_RQE resource. Because the Resource Manager is absent, these waits never get posted, so the transactions suspend. You must issue a CEMT SET TASK FORCEPURGE command to remove these suspended transactions from the system.

What to do if CICS has stalled

CICS can stall during initialization, when it is running apparently “normally”, or during termination. If XRF takeover by an alternate CICS system fails to complete satisfactorily, that might also appear to you as a CICS stall.

These possibilities are dealt with separately in:

“CICS has stalled during initialization”
“CICS has stalled during a run” on page 138
“CICS has stalled during termination” on page 141
Chapter 11, “Dealing with XRF errors” on page 199.

CICS has stalled during initialization

If CICS stalls during initialization, on cold start, warm start, or emergency restart, the first place to look is the MVS console log. This tells you how far initialization has progressed.

Note that there might be significant delays at specific stages of initialization, depending on how CICS last terminated.

On cold start, loading the GRPLIST definitions from the CSD data set can take several minutes. For large systems, the delay could be 20 minutes or more while this takes place. You can tell if this stage of initialization has been reached because you get this console message:

```
DFHSI1511 INSTALLING GROUP LIST xxxxxxxx
```

On warm start, there may be a considerable delay while resource definitions are being created from the global catalog. You are not told explicitly that this processing is taking place, but it occurs just after this message is issued:

```
DFH4500 - nn OF nn JOURNALS SUCCESSFULLY OPENED
```

If you find that unexpected delays occur at other times during CICS initialization, consider the messages that have already been sent to the console and see if they suggest the reason for the wait. For example, a shortage of storage is one of the most common causes of stalling, and is always accompanied by a message. The JCL job log is another useful source of information.

You can find out if this has happened by taking an SDUMP of the CICS region. Format the dump using the keywords KE and DS, to get the kernel and dispatcher task summaries.

Consider, too, whether any first- or second-stage program list table (PLT) program that you have written could be in error. If such a program does not follow the strict protocols that are required, it can cause CICS to stall. For programming information about PLT programs, see the *CICS/ESA Customization Guide*.

CICS has stalled during a run

If a CICS region that has been running normally stalls, so that it produces no output and accepts no input, the scope of the problem is potentially system-wide. The problem might be confined exclusively to CICS, or it could be caused by any other task running under MVS.

Look first on your MVS console for any messages. Look particularly for messages indicating that operator intervention is needed, for example to change a tape volume. The action could be required on behalf of a CICS task, or it could be for any other program that CICS interfaces with.

If there is no operator action outstanding, inquire on active users at the MVS console to see what the CPU usage is for CICS. If you find the value is very high, this probably indicates that a task is looping. Read Chapter 7, “Dealing with loops” on page 143 for advice about investigating the problem further.

If the CPU usage is low, CICS is doing very little work. Some of the possible reasons are:

- The system definition parameters are not suitable for your system.
- The system is short on storage, and new tasks cannot be started. This situation is unlikely to last for long unless old tasks cannot, for some reason, be purged.
- The system is at one of the MXT or transaction class limits, and no new tasks can be attached. In such a case, it is likely that existing tasks are deadlocked, and for some reason they cannot be timed out.
- There is an exclusive control conflict for a volume.

- There is a problem with the communications access method.
- There is a CICS system error.

The way you can find out if any of these apply to your system is dealt with in the paragraphs that follow. For some of the investigations, you will need to refer to a system dump of the CICS region. If you don't already have one, you can request one using the MVS console. Make sure that CICS *is* apparently stalled at the time you take the dump, because otherwise it won't provide the evidence you need. Format the dump using the formatting keywords KE and XM, to get the storage areas for the kernel and the transaction manager.

Are the system definition parameters wrong?

It is possible that the system definition parameters for your system are causing it to stall, possibly at a critical loading. Take a look at what has been specified, paying particular attention to these items:

- The CICS maximum tasks (MXT) and transaction class (MAXACTIVE) limits. If these are too low, new tasks could fail to be attached. If you suspect one of these limits is the cause of the stall, read "Are MXT or transaction class limits causing the stall?" on page 140 for a way of getting further evidence.
- ICV, the system region exit time. If this is set too high, CICS might relinquish control to the operating system for longer than intended when it has no work to do, perhaps giving the impression of a stall.
- ICVR, the runaway task time interval. If this is set too high, a runaway task could stop other tasks from running for a relatively long time. It can have a value up to 2 700 000 milliseconds, in which case a runaway task would not time out for 45 minutes. CICS could, in the meantime, be stalled. If the ICVR is set to 0, the runaway task does not time out at all.

  You should already have an indication if the ICVR is the problem, from the CPU usage (see above).
- ICVTSD, the terminal scan delay interval. The effect differs for VTAM and non-VTAM terminals, but if its value is not appropriate for your system the effect could make you think that CICS is stalled.

For more details about the choice of these and other system definition parameters, see the *CICS/ESA Performance Guide*.

Is the system short on storage?

If storage is under stress, storage manager statistics indicate that a storage stress situation has occurred ('Times went short on storage'). In addition, if the SOS is caused by a suspended GETMAIN or if CICS is unable to alleviate the situation by releasing programs with no current user, and slowing the attachment of new tasks:

- A message is sent to the console saying that CICS is short on storage (DFHSM0131I for storage below the 16 megabyte line, DFHSM0133I for storage above the 16 megabyte line)
- The storage manager statistic 'Times went short on storage' is updated.

CICS can go short on storage independently in any DSA. You may see tasks suspended on any of the resource types, CDSA, SDSA, RDSA, UDSA, ECDSA, ESDSA, ERDSA, or EUDSA.

Are MXT or transaction class limits causing the stall?

Before new transactions can be attached for the first time, they must qualify under the MXT and transaction class limits. In a system that is running normally, tasks run and terminate and new transactions are attached, even though these limits are reached occasionally. It is only when tasks can neither complete nor be purged from the system that CICS can stall as a result of one of these limits being reached.

Look first at the transaction manager summary in the formatted system dump.

Investigate the tasks accepted into the MXT set of tasks to see if they are causing the problem. XM dump formatting formats the state of MXT and provides a summary of the TCLASSes and of the transactions waiting for acceptance into each TCLASS.

Now look at the task control queue control area, in the AP section of the dump for a summary of task enqueues and resources. If you find that many tasks are enqueued on the same resource, this could indicate a deadlock involving tasks that are not purgeable. Investigate these tasks to find if they are causing the problem.

Is there an exclusive control conflict on a volume?

Some programs use MVS RESERVE to gain exclusive control of a volume, and nothing else can have access to any data set on that volume until it is released. Watch for operations involving database access, because these could indicate an exclusive control conflict on a volume.

Is there a problem with the communications access method?

If you suspect that there is a communication problem, you can inquire on the status of VTAM from the MVS console, but not on the status of TCAM. To do this, use the command `F cicsname,CEMT INQ VTAM`. Substitute the name of the CICS job for "cicsname". You can only use this command if the MVS console has been defined to CICS as a terminal. The status returned has a value of OPEN or CLOSED.

- If the VTAM status is OPEN, the problem could be associated with processing done in the VTAM part of your system or with processing done in the CICS part of your system. If it appears that there is a communication problem, consider using either CICS VTAM exit tracing or VTAM buffer tracing. For guidance about using these techniques, see Chapter 14, "Using traces in problem determination" on page 219.
- If the VTAM status is CLOSED, CICS cannot use VTAM to perform communication functions.

Is there a CICS system error?

If you have investigated all the task activity, and all the other possibilities from the list, and you have still not found an explanation for the stall, it is possible that there is a CICS system error. Contact the IBM Support Center with the problem.

CICS has stalled during termination

Waits often occur when CICS is being quiesced because some terminal input or output has not been completed. To test this possibility, try using the CEMT transaction to inquire on the tasks currently in the system.

CICS termination takes place in two stages:

1. All transactions are quiesced.
2. All data sets and terminals are closed.

If you find that you cannot use the CEMT transaction, it is likely that the system is already in the second stage of termination. CEMT cannot be used beyond the first stage of termination.

Note: Even if CEMT is not included in the transaction list table (XLT), you can still use it in the first stage of termination.

The action to take next depends on whether you can use the CEMT transaction, and if so, whether or not there are current user tasks.

- **If you can use the CEMT transaction:**
  - If there are user tasks currently in the system, check what they are. A task may be performing a prolonged termination routine, or it might be waiting on a resource before it can complete its processing. It is also possible that a task is waiting for operator intervention.

    Determine what type of terminal is associated with the task. If the terminal is a 3270 device, some keyboard input might be expected. If it is a printer, it might have been powered off or it might have run out of paper.
  - If there are no user tasks in the system, it may be that one or more terminals have not been closed. Use the CEMT transaction to see which terminals are currently INSERVICE, and then use CEMT SET to place them OUTSERVICE.

  If these actions fail, proceed as if you were unable to use the CEMT transaction.
- **If you cannot use the CEMT transaction**, go to the MVS console or the NetView master terminal and display the active sessions. If necessary, close down the network using the `VARY NET,INACT,ID=applid` command. This should enable CICS to resume its termination sequence. If it does not, you might need to cancel the CICS job. If this does happen, consider whether any PLT program running in the second quiesce stage could be in error. If such a program did not follow the strict protocols that are required, it could cause CICS to stall during termination. For programming information about PLT programs, see the *CICS/ESA Customization Guide*.

Chapter 7. Dealing with loops

A loop is a sequence of instructions that is executed repetitively. Loops that are coded into applications must always be guaranteed to terminate, because otherwise you could get any of the symptoms of loops described in Chapter 2, “Classifying the problem” on page 11.

If a loop does not terminate, it could be that the termination condition can never occur, or it might not be tested for, or the conditional branch could erroneously cause the loop to be executed over again when the condition is met.

This chapter outlines procedures for finding which programs are involved in a loop that does not terminate.

If you find that the looping code is in one of your applications, you need to check through the code to find out which instructions are in error. If it looks as if the error is in CICS code, you probably need to contact the IBM Support Center.

Some CICS domains can detect loops in their own routines, and let you know if one is suspected by sending the following message:

DFHxx0004 *applid* **A possible loop has been detected at offset X**'*offset*' **in module** *modname*.

The two characters “xx” represent the two-character domain index. If, for example, monitoring domain had detected the loop, the message number would be DFHMN0004.

If you see this sort of message repeatedly, contact the IBM Support Center.

What sort of loop is indicated by the symptoms?

Unplanned loops can be divided into those that can be detected by CICS, and those that cannot. Figure 15 shows a hierarchical classification of loops.

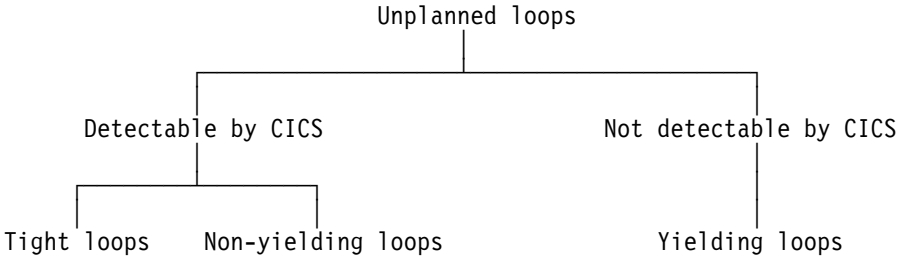


Figure 15. Hierarchical classification of loops

Figure 16 gives a specific example of code containing a simple tight loop.

```
PROCEDURE DIVISION.
    EXEC CICS
        HANDLE CONDITION ERROR(ERROR-EXIT)
              ENDFILE(END-MSG)
    END-EXEC.
ROUTE-FILE.
    EXEC CICS
        ROUTE INTERVAL(0)
              LIST(TERM-ID)
    END-EXEC.
 NEW-LINE-ATTRIBUTE.
    GO TO NEW-LINE-ATTRIBUTE.
    MOVE LOW-VALUES TO PRNTAREA.
    MOVE DFHBMPNL TO PRNTAREA.
```

Figure 16. Example of code containing a tight loop

CICS can detect some looping tasks by comparing the length of time the tasks have been running with the runaway time interval, ICVR, that you code in the system initialization table. If a task runs for longer than the interval you specify, CICS regards it as “runaway” and causes it to abend with an abend code of AICA.

However, in some cases, CICS requests that are contained in the looping code can cause the timer to be reset. Not every CICS request can do this; it can only happen if the request can cause the task to be suspended. Thus, if the looping code contains such a request, CICS cannot detect that it is looping.

The properties of the different types of loop, and the ways you can investigate them, are described in the sections that follow.

Tight loops and non-yielding loops

Tight loops and non-yielding loops are both characterized by the fact that the looping task can never be suspended within the limits of the loop. This makes them detectable by CICS, which compares the time they have been running continually with the runaway time interval, ICVR, that you code in the system initialization table. If the tasks run for longer than the interval you specify, CICS regards them as “runaway” and causes them to abend with an abend code of AICA.

Note: If you make the ICVR value equal to 0, runaway task detection is disabled. Runaway tasks can then cause the CICS region to stall, meaning that CICS must be canceled and brought up again. You might choose to set ICVR to zero in test systems, because of the wide variation in response times. However, it is usually more advisable to set ICVR to a large value in test systems.

A tight loop is one involving a single program, where the same instructions are executed repeatedly and control is never returned to CICS.

Figure 17 on page 145 shows a tight loop, with an indefinite number of instructions contained in the loop. None of the instructions returns control to CICS. In the extreme case, there could be a single instruction in the loop, causing a branch to itself.

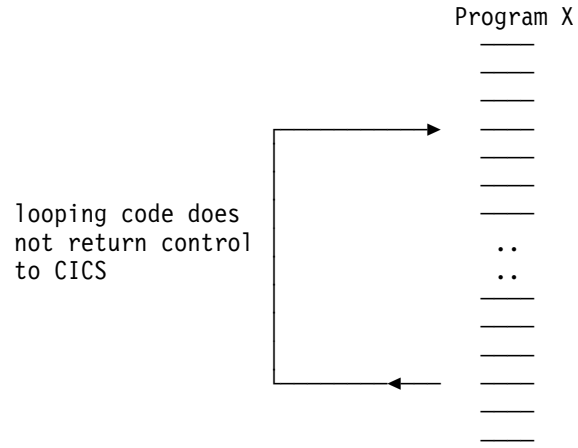


Figure 17. A tight loop

A non-yielding loop is also contained in a single program, but it differs from a tight loop in that control is returned temporarily from the program to CICS. However, the CICS routines that are invoked are ones that can neither suspend the program nor pass control to the dispatcher.[1] There is, therefore, no point at which the task can be suspended, and so the ICVR cannot be reset.

Figure 18 shows a non-yielding loop.

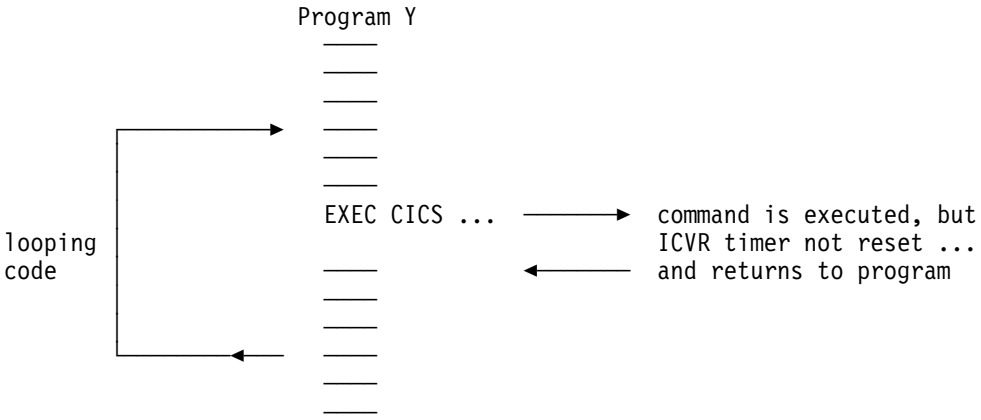


Figure 18. A non-yielding loop

[1] The CICS commands that don't cause tasks to wait include (but are not restricted to) ASKTIME, DEQ, ENQ, ENTER TRACENUM, FREEMAIN, HANDLE, RELEASE, TRACE ON/OFF. Whether a command allows the ICVR to be reset might also depend on other factors. For instance, a FREEMAIN might reset the ICVR if the storage lock is held. A READ might also not wait if the desired record is already in a VSAM buffer.

Figure 19 shows an example of code that contains a simple non-yielding loop. In this case, the loop contains only one CICS command, EXEC CICS ASKTIME.

```
PROCEDURE DIVISION.
    EXEC CICS
        HANDLE CONDITION ERROR(ERROR-EXIT)
               ENDFILE(END-MSG)
    END-EXEC.
ROUTE-FILE.
    EXEC CICS
        ROUTE INTERVAL(0)
              LIST(TERM-ID)
    END-EXEC.
 NEW-LINE-ATTRIBUTE.
    EXEC CICS
        ASKTIME
    END-EXEC.
    GO TO NEW-LINE-ATTRIBUTE.
    MOVE LOW-VALUES TO PRNTAREA.
    MOVE DFHBMPNL TO PRNTAREA.
```

Figure 19. Example of code containing a non-yielding loop

If you have a transaction that repeatedly abends with an abend code of AICA, first make sure the ICVR value has not been set too low. If the value seems reasonable, read "Investigating loops that cause transactions to abend with abend code AICA" on page 148 for advice on determining the limits of the loop.

If you have a stalled CICS region, diagnose the problem using the techniques in "What to do if CICS has stalled" on page 137. Check if the ICVR value has been set to zero. If it has, change the value and try to cause a transaction to abend with a code of AICA.

Yielding loops

Yielding loops are characterized by returning control at some point to a CICS routine that can suspend the looping task. However, the looping task is eventually resumed, and so the loop continues.

CICS is unable to use the runaway task timer to detect yielding loops, because the timer is reset whenever the task is suspended. Thus, the runaway task time is unlikely ever to be exceeded, and so the loop goes undetected by the system.

Yielding loops typically involve a number of programs. The programs might be linked to and returned from, or control might be transferred from one program to another in the loop. A yielding loop can also be confined to just one program, in which case it must contain at least one wait-enabling CICS command.

Figure 20 shows a yielding loop involving two programs, A and B. Program A links to program B, and then program B subsequently returns.

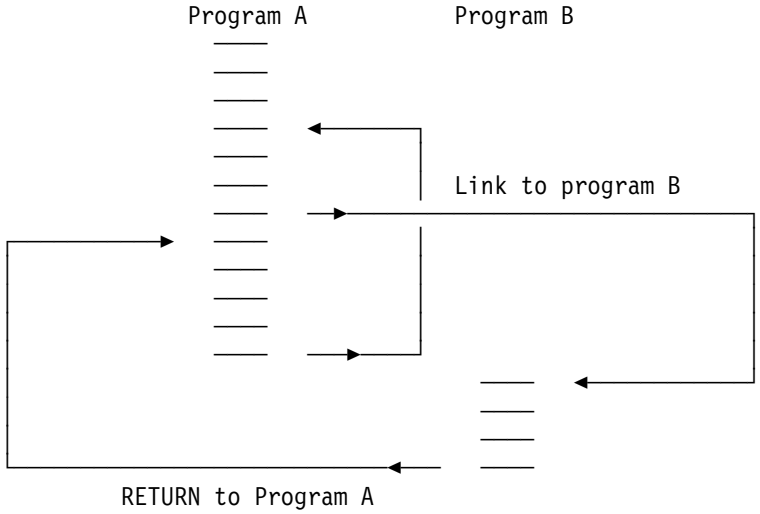


Figure 20. A yielding loop

Figure 21 shows a specific example of a yielding loop within a single program. This code issues the SUSPEND command, which is always a yielding type of command. Every time SUSPEND is issued, the dispatcher suspends the task issuing the request, and sees if any other task of higher priority can run. If no such task is ready, the program that issued the SUSPEND is resumed.

```
PROCEDURE DIVISION.
    EXEC CICS
        HANDLE CONDITION ERROR(ERROR-EXIT)
               ENDFILE(END-MSG)
    END-EXEC.
ROUTE-FILE.
    EXEC CICS
        ROUTE INTERVAL(0)
              LIST(TERM-ID)
    END-EXEC.
 NEW-LINE-ATTRIBUTE.
    EXEC CICS
        SUSPEND
    END-EXEC.
    GO TO NEW-LINE-ATTRIBUTE.
    MOVE LOW-VALUES TO PRNTAREA.
    MOVE DFHBMPNL TO PRNTAREA.
```

Figure 21. Example of code containing a yielding loop

You can detect a yielding loop only by circumstantial evidence such as repetitive output, or excessive use of storage. A fuller description of what to look out for is given in “Loops” on page 17.

If you suspect that you have a yielding loop, turn to “Investigating loops that are not detected by CICS” on page 150 for further guidance.

Investigating loops that cause transactions to abend with abend code AICA

If the loop causes a transaction to abend with abend code AICA, it must either be a tight loop or a non-yielding loop. You do not need to find which type you have, although this is likely to be revealed to you when you do your investigation.

Both a tight loop and a non-yielding loop are characterized by being confined to a single user program. You should know the identity of the transaction to which the program belongs, because it is the transaction that abended with code AICA when the runaway task was detected.

The documentation you need

When investigating loops that cause transactions to abend AICA, you need the CICS system dump accompanying the abend. System dumping must be enabled for dump code AICA.

You can use the system dump to find out:

- Whether the loop is in your user code or in CICS code
- If the loop is in your user code, the point at which the loop was entered.

It is also useful to have trace running, as trace can help you to identify the point in your program where looping started. If you have a non-yielding loop, it can probably also show you some instructions in the loop.

A tight loop is unlikely to contain many instructions, and you might be able to capture all the evidence you need from the record of events in the internal trace table. A non-yielding loop may contain more instructions, depending on the EXEC CICS commands it contains, but you might still be able to capture the evidence you need from the record of events in the internal trace table. If you find that it is not big enough, direct tracing to the auxiliary trace destination instead.

You need to trace CICS system activity selectively, to ensure that most of the data you obtain is relevant to the problem. Set up the tracing like this:

1. Select level-1 special tracing for AP domain, and for the EXEC interface program (EI).
2. Select special tracing for just the task that has the loop, and disable tracing for all other tasks by turning the master system trace flag off.

You can find guidance about setting up these tracing options in Chapter 14, “Using traces in problem determination” on page 219.

Now start the task, and wait until it abends AICA. Format the CICS system dump with formatting keywords KE and TR, to get the kernel storage areas and the internal trace table. (See “Formatting system dumps” on page 277.) You now have the documentation you need to find the loop.

Looking at the evidence

Look first at the kernel task summary. The runaway task is flagged "*YES*" in the ERROR column. The status of the task is shown as "***Running**".

Now use the kernel task number for the looping task to find its linkage stack. If a user task is looping, DFHAPLI, a transaction manager program, should be near the top of the stack. You are likely to find other CICS modules at the top of the stack that have been invoked in response to the abend. For example, those associated with taking the dump. If you find any program or subroutine above DFHAPLI that has not been invoked in response to the error, it is possible that CICS code, or the code of another program, has been looping. If you find that the loop is within CICS code, you need to contact the IBM Support Center. Make sure you keep the dump, because the Support Center staff need it to investigate the problem.

If the kernel linkage stack entries suggest that the loop is in your user program, you next need to identify the loop.

Identifying the loop

There are two approaches to identifying loops in user programs. You can use the trace table, or you can look in the transaction dump.

Using the trace table

Go to the last entry in the internal trace table, and work backward until you get to an entry for point ID AP 1942. The trace entry should have been made when recovery was entered after the transaction abended AICA. Make a note of the task number, so you can check that any other trace entries you read relate to the same abended task.

The entries preceding AP 1942 should have been made either just before the loop was entered (for a tight loop), or within the loop itself (for a non-yielding loop). Watch in particular for trace entries with the point ID AP 00E1. These are made on entry to the EXEC interface program (DFHEIP) whenever your program issues an EXEC CICS command, and again on exit from the EXEC interface program. Field B gives you the value of EIBFN, which identifies the specific command that was issued. (For a list of EIBFN values and their meanings, see the *CICS/ESA User's Handbook*.)

For trace entries made on exit from DFHEIP, field A gives you the response code from the request. Look carefully at any response codes—they could provide the clue to the loop. Has the program been designed to deal with every possible response from DFHEIP? Could the response code you see explain the loop?

If you see a repeating pattern of trace points for AP 00E1, you have a non-yielding loop. If you can match the repeating pattern to statements in the source code for your program, you have identified the limits of the loop.

If you see no repeating pattern of trace points for AP 00E1, it is likely that you have a tight loop. The last entry for AP 00E1 (if there is one) should have been made from a point just before the program entered the loop. You might be able to recognize the point in the program where the request was made, by matching trace entries with the source code of the program.

Using the transaction dump

First you need to find the PSW, and see if it points into your program. This is likely to be the case if you have a tight loop, and it should lead you to an instruction within the loop.

If the next instruction address is not within your code, it is of less value for locating the loop. However, you should attempt to identify the module containing the instruction, as it is likely to be one that was called during the execution of a CICS request made within the loop. Use the module index at the end of the formatted dump to find the module name. If the PSW address is not contained in one of these areas, another program was probably executing on behalf of CICS when the runaway task timer expired.

Note: It is possible that the loop was contained entirely within a module owned by CICS or some other product, and your program was not responsible for it at all. If you find that the loop is contained within CICS code, contact the IBM Support Center.

If the PSW does point to a module outside your application program, you need to find the address of the return point in your program from the contents of register 14 in the appropriate register save area. The return address will lie within the loop, if the loop is not confined to system code.

When you have located a point within the loop, work through the source code and try to find the limits of the loop.

Finding the reason for the loop

When you have identified the limits of the loop, you need to find the reason why the loop occurred. Assuming you have the trace, and EI level-1 tracing has been done, ensure that you can explain why each EIP entry is there. Verify that the responses are as expected.

A good place to look for clues to loops is immediately before the loop sequence, the first time it is entered. Occasionally, a request that results in an unexpected return code can trigger a loop. However, you usually can only see the last entry before the loop if you have CICS auxiliary or GTF trace running, because the internal trace table is likely to wrap before the AICA abend occurs.

Investigating loops that are not detected by CICS

You probably suspect that you have a loop through circumstantial evidence, and CICS has failed to detect it. You might, for example, have seen some sort of repetitive output, or statistics might have shown an excessive number of I/O operations or requests for storage. These types of symptom can indicate that you have a yielding loop.

The nature of the symptoms may indicate which transaction is involved, but you probably need to use trace to define the limits of the loop.

Use auxiliary trace to capture the trace entries, to ensure that the entire loop is captured in the trace data. If you use internal trace, there is a danger that wraparound will prevent you from seeing the whole loop.

Use the CETR transaction to set up the following tracing options. You can use the transaction dynamically, on the running CICS system. For guidance about using the CETR transaction, see Chapter 14, “Using traces in problem determination” on page 219.

1. Select level-1 special tracing for every component, using the CETR transaction. You need to capture as much trace information for the task as possible, because you don’t yet know what functions are involved in the loop.
2. Set all standard tracing off, by setting the master system trace flag off.
3. Select special tracing for just the task containing the loop.
4. Set the auxiliary tracing status to STARTED, and the auxiliary switch status to ALL. As CETR allows you to control trace dynamically, you do not need to start tracing until the task is running and the symptoms of looping appear.

These steps ensure that you get all level-1 trace points traced for just the task you suspect of looping, the trace entries being sent to the auxiliary trace destination.

When you have captured the trace data, you need to purge the looping task from the system. Use the CEMT INQ TASK command to find the number of the task, and then purge it using either the CEMT SET TASK PURGE or the CEMT SET TASK FORCEPURGE command. This causes the transaction to abend, and to produce a transaction dump of the task storage areas.

Note: The use of FORCEPURGE is, in general, not recommended, because it can cause unpredictable system problems. For example, it causes task storage areas to be released, including I/O areas, without notifying any components that might be accessing them. If the FORCEPURGEd task was waiting for input, such an area might be written to after it is released. The storage might even be in use by another task when the input occurs.

The documentation you need

In addition to the auxiliary trace data and the transaction dump, you need source listings of all the programs in the transaction.

The trace data and the program listings should enable you to identify the limits of the loop. You need the transaction dump to examine the user storage for the program. The data you find there could provide the evidence you need to explain why the loop occurred.

Identifying the loop

Examine the trace table, and try to detect the repeating pattern of trace entries. If you cannot do so straightaway, remember that many different programs might be involved, and the loop could be large. Another possibility is that you might not have captured the entire loop in the trace data set. This could be because the loop did not have time to complete one cycle before you purged the transaction, or the trace data sets might have wrapped before the loop was complete.

Consider also the possibility that you might not be dealing with a loop, and the symptoms you saw are due to something else—poor application design, for example.

If you are able to detect a pattern, you should be able to identify the corresponding pattern of statements in your source code.

Note: The PSW is of no value in locating loops that are not detected by CICS. The contents of the PSW are unpredictable, and the PSW is not formatted in the transaction dump for ATCH abends.

Finding the reason for the loop

Look carefully at the statements contained in the loop. Does the logic of the code suggest why the loop occurred? If not, you need to examine the contents of data fields in the task user storage. Look particularly for unexpected response codes, and null values when finite values are expected. Programs can react unpredictably when they encounter these conditions, unless they are tested for and handled accordingly.

What to do if you cannot find the reason for a non-yielding or a yielding loop

If you cannot find the reason for a non-yielding or a yielding loop using the techniques outlined above, there are two more approaches that you can adopt:

1. Use the interactive tools that CICS provides
2. Modify the program, and execute it again.

Investigating loops using interactive tools

If you have a non-yielding or a yielding loop, you can use the execution diagnostic facility (CEDF) to look at the various parts of your program and storage at each interaction with CICS. If you suspect that some unexpected return code might have caused the problem, CEDF is a convenient way of investigating the possibility.

CECI and CEBR are also useful for investigating loops. You can, for example, use them to examine the status of files and queues during the execution of your program. Programs can react unpredictably if records and queue entries are not found when these conditions are not tested for and handled accordingly.

Modifying your program to investigate the loop

If the program is extremely complex, or the data path difficult to follow, you may need to insert additional statements into the source code. Even extra ASKTIME commands allow you to use EDF and inspect the program at more points. You can also request dumps from within your program, and insert user trace entries, to help you to find the reason for the loop.

Chapter 8. Dealing with performance problems

When you have a performance problem, you might be able to find that it is characterized by one of the following symptoms, each of which represents a particular processing bottleneck. If so, turn directly to the relevant section:

1. Some tasks fail to get attached to the transaction manager—see “Finding why tasks fail to get attached to the transaction manager” on page 155.
2. Some tasks fail to get attached to the dispatcher—see “Finding why tasks fail to get attached to the dispatcher” on page 156.
3. Some tasks get attached to the dispatcher, but fail to get dispatched—see “Finding why tasks fail to get an initial dispatch” on page 158.
4. Tasks get attached to the dispatcher and then run and complete, but take a long time to do so—see “Finding out why tasks take a long time to complete” on page 159.

If you are only aware that performance is poor, and you have not yet found which of these is relevant to your system, read “Finding the bottleneck” on page 154.

There is a quick reference section at the end of this chapter (“A summary of performance bottlenecks, symptoms, and causes” on page 160) that summarizes bottlenecks, symptoms, and actions that you should take.

Finding the bottleneck

Four potential bottlenecks can be identified for user tasks, and three for CICS system tasks. The bottlenecks are summarized in Figure 22.

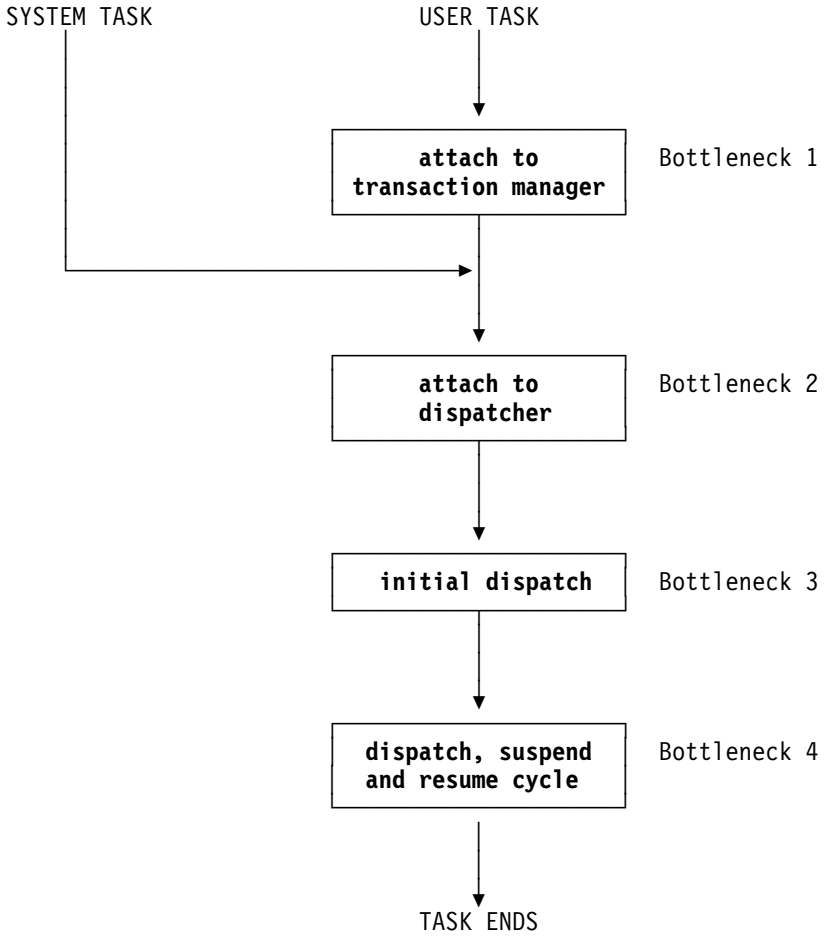


Figure 22. Potential performance bottlenecks in a CICS system

Each bottleneck is affected by a different set of system parameters, and you may find that adjusting the parameters solve the problem. It is useful to determine which bottleneck is causing your performance problem, so you can find out which parameters you need to consider.

If performance is particularly poor for any of the tasks in your system, you might be able to capture useful information about them with CEMT INQ TASK. However, tasks usually run more quickly than you can inquire on them, even though there may be a performance problem. You then need to consider using performance class monitoring or tracing to get the information you need.

Initial attach to the transaction manager: If a task hasn't been attached to the transaction manager, you cannot get any information about its status online. CEMT INQ TASK returns a response indicating that the task is not known. If the task hasn't already run and ended, this response means that it has not been attached to the transaction manager.

Guidance about finding out why tasks take a long time to get an initial attach to the transaction manager is given in “Finding why tasks fail to get attached to the transaction manager” on page 155.

Initial attach to the dispatcher: If a task has been attached to the transaction manager, but has not yet been attached to the dispatcher, CEMT INQ TASK will show it to be ‘SUSPENDED’ on a resource type of MXT or TCLASS. These are the only valid reasons why a user task, having been attached to the transaction manager, would not be attached to the dispatcher.

If CEMT INQ TASK returns anything other than this, the task is not waiting to be attached to the dispatcher. However, consider whether the MXT limit might be causing the performance problem, even though individual tasks are not being held up long enough for you to use CEMT INQ TASK on them. In such a case, use monitoring and tracing to find just how long tasks are waiting to be attached to the dispatcher.

Guidance about finding whether the MXT limit is to blame for the performance problem is given in “Is the MXT limit preventing tasks from getting attached?” on page 156.

Initial dispatch: A task can be attached to the dispatcher, but then take a long time to get an initial dispatch. In such a case, CEMT INQ TASK returns a status of ‘READY’ for the task. If you keep getting this response and the task fails to do anything, it is likely that the task you are inquiring on is not getting its first dispatch.

The delay might be too short for you to use CEMT INQ TASK in this way, but still long enough to cause a performance problem. In such a case, you need to use tracing or performance class monitoring for the task, either of which would tell you how long the task had to wait for an initial attachment to the dispatcher.

If you think your performance problem could be due to tasks taking a long time to get a first dispatch, read “Finding why tasks fail to get an initial dispatch” on page 158.

The dispatch, suspend, and resume cycle: If performance is poor and tasks are getting attached and dispatched, the problem lies with the dispatch, suspend and resume cycle. Tasks run, but the overall performance is poor. If you are able to show that tasks are getting attached and then dispatched, read “Finding out why tasks take a long time to complete” on page 159.

Finding why tasks fail to get attached to the transaction manager

A task might fail to get attached to the transaction manager for one of the following reasons:

1. The interval specified on an EXEC CICS START command might not have expired, or the time specified might not have been reached, or there might be some error affecting interval control.

   Guidance about investigating these possibilities is given in “Interval control waits” on page 128. You need to consider doing this only if INTERVAL or TIME was specified on the START command.

2. The terminal specified on an EXEC CICS START command might not be available. It could be currently OUTSERVICE, or executing some other task.

You can check its status using CEMT INQ TERMINAL, and perhaps take some remedial action.

Remember that several tasks might be queued on the terminal, some of which might require operator interaction. In such a case, the transaction to be started might not get attached to the transaction manager for a considerable time.

3. A remote system specified on an EXEC CICS START command might not be available, or an error condition might have been detected in the remote system. In such a case, the error would not be reported back to the local system.

   You can use CEMT INQ TERMINAL to inquire on the status of the remote system.

Finding why tasks fail to get attached to the dispatcher

Two valid reasons why a user task might fail to get an initial attach to the dispatcher are[2]:

- The system is at the MXT limit (see "Is the MXT limit preventing tasks from getting attached?").
- The task belongs to a transaction class that is at its MAXACTIVE limit.

Is the MXT limit preventing tasks from getting attached?

Before the transaction manager can attach a user task to the dispatcher, the task must first qualify under the MXT (maximum tasks in the system) and transaction class limits. If a task is not getting attached, it is possible that one or both of these values is too small.

You might be able to use CEMT INQ TASK to show that a task is failing to get attached because of the MXT or transaction class limits. If you cannot use CEMT because the task is held up for too short a time, you can look at either the transaction global statistics, transaction class statistics, or the CICS performance-class monitoring records. Another option is to use CICS system tracing.

Using transaction manager statistics: To find out how often the MXT and transaction class limits are reached, look at the transaction global statistics and transaction class statistics. If you compare the number of times these limits are reached with the total number of transactions, you can see whether the values set for the limits are adversely affecting performance.

To gather statistics relating to the number of times that the MXT or transaction class limits are reached, you need to use, at the start of the run, the command CEMT PERFORM STATISTICS RECORD (or your site replacement) with the keywords TRANSACTION and TRANCLASS.

```
CEMT PERFORM STATISTICS RECORD [TRANCLASS TRANSACTION]
```

The statistics are gathered and recorded in the SMF data set. You can format this data set by using the statistics utility program, DFHSTUP. Please read the 'DFHSTUP' section in the *CICS/ESA Operations and Utilities Guide* for details on how to use this facility.

---

[2] For a system task, there may not be enough storage to build the new task. This sort of problem is more likely to occur near peak system load times.

When formatting the SMF data set using DFHSTUP, you may find the following DFHSTUP control parameters useful:

```
SELECT APPLID=
COLLECTION TYPE=REQ
TIME START=       ,STOP=
DATE START=       ,STOP=
```

See the 'DFHSTUP' section in the *CICS/ESA Operations and Utilities Guide* for details on how to code these parameters. If you correctly code these control parameters, you will avoid the formatting of much information that may well be unnecessary at this point.

If MXT is never reached, or reached only infrequently, it is not affecting performance.

If MXT is reached for 5% of transactions, this might have a noticeable effect on performance. When the ratio reaches 10%, there is likely to be a significant effect on performance, and this could account for some tasks taking a long time to get a first attach.

Consider revising the MXT and transaction class values if the statistics indicate that they are affecting performance. For guidance about the performance considerations when you set these limits, see the *CICS/ESA Performance Guide*.

Using CICS monitoring: You can use monitoring information to find out how long an individual task waits to be attached to the Dispatcher. If you want to summarize the monitoring data of particular transactions to assess the impact across many tasks, you can use products such as Service Level Reporter (SLR) or Enterprise Performance Data Manager/MVS (EPDM/MVS).

Monitoring produces performance-class records (if performance-class monitoring is active) for each task that is executing or has executed in the CICS region. Performance-class records contain a breakdown of the delays incurred in dispatching a task, part of which is the impact on a task of the MXT limit and transaction class limits.

For further information on the data produced by CICS Monitoring see the *CICS/ESA Performance Guide*.

Using trace: You can use trace if you want to find out just how long an individual task waits to be attached to the dispatcher.

If you don't want to do any other tracing, internal trace is probably a suitable destination for trace entries. Because the task you are interested in is almost inactive, very few trace entries are generated.

Select special tracing for the transaction associated with the task, and turn off all standard tracing by setting the master system trace flag off. Define as special trace points the level-1 trace points for transaction manager (XM), and for the CICS task controlling the facility that initiates the task, such as terminal control (TC). Make sure that no other trace points are defined as special. For guidance about setting up these tracing options, see Chapter 14, "Using traces in problem determination" on page 219.

When you have selected the options, start tracing to the internal trace table and attempt to initiate the task. When the task starts, get a system dump with CEMT PERFORM SNAP. Format the dump using the keyword TR, to get the internal trace table.

Look for the trace entry showing terminal control calling the transaction manager with a request to attach the task, and the subsequent trace entry showing the transaction manager calling dispatcher domain with a request to attach the task. The time stamps on the two trace entries tell you the time that elapsed between the two events. That is equal to the time taken for the task to be attached.

Finding why tasks fail to get an initial dispatch

When a task is past the transaction class and MXT barriers, it can be attached to the dispatcher. It must then wait for its initial dispatch. If tasks are made to wait for a relatively long time for their first dispatch, you will probably notice the degradation in the performance of the system.

You can get evidence that tasks are waiting too long for a first dispatch from performance class monitoring. If you do find this to be the case, you need to investigate the reasons for the delay. To calculate the initial dispatch delay incurred by a task use the following fields from the performance-class monitoring record:

DSPDELAY = First dispatch delay
TCLDELAY = Transaction Class delay
MXTDELAY = MXT delay

Using the above names:

Delay in dispatcher = DSPDELAY - (TCLDELAY + MXTDELAY)

If 'Delay in Dispatcher' is significantly greater than 0, the dispatcher could not dispatch the task immediately.

The factors that influence the length of time a task must wait before getting its first dispatch are:

- The priority of the task
- Whether the system is becoming short on storage.

Priorities of tasks: Normally, the priorities of tasks determine the order in which they are dispatched. Priorities can have any value in the range 1–255. If your task is getting a first dispatch (and, possibly, subsequent dispatches) too slowly, you might consider changing its priority to a higher value.

You have no control over the priorities of CICS system tasks.

One other factor affecting the priorities of tasks is the priority aging multiplier, PRTYAGE, that you code in the system initialization parameters. This determines the rate at which tasks in the system can have their priorities aged. Altering the value of PRTYAGE affects the rate at which tasks are dispatched, and you probably need to experiment to find the best value for your system.

Storage stress conditions are detailed in the storage manager statistics. CICS attempts to alleviate the situation by releasing programs with no current user, and

by not attaching new tasks. If these actions fail to eliminate storage stress or if the
SOS condition is caused by a suspended GETMAIN, one or both of these
messages is sent to the console:

DFHSM0131 *applid* **CICS is under stress (short on storage below 16MB)**

DFHSM0133 *applid* **CICS is under stress (short on storage above 16MB)**

If you don't observe the SOS messages, you can find out how many times CICS has gone SOS from the storage manager statistics ('Times went short on storage'). You can also get this information from the storage manager domain DSA summary in a formatted system dump.

Note: Release of the storage cushion is not the only cause of CICS going SOS. The condition is also raised if a task makes an unconditional request for storage greater than the storage cushion size when the system is approaching SOS. In such a case, the cushion is not released, but the task making the unconditional request is suspended and message DFHSM0131I or DFHSM0133I may be issued. CICS resumes the suspended tasks immediately if storage is made available by CICS releasing unused programs. The short-on-storage condition remains until all the previously suspended tasks have obtained the storage they requested.

Two other conditions are recognized by the dispatcher on the approach to SOS, namely ‘storage getting short’ and ‘storage critical’. The two conditions affect the chance of new tasks getting a first dispatch.

From the ‘storage getting short’ point, through ‘storage critical’ and right up to SOS, the priorities of new user tasks are reduced in proportion to the severity of the condition[3]. At first, you are not likely to notice the effect, but as ‘storage critical’ is approached, new tasks might typically be delayed by up to a second before they are dispatched for the first time.

It is likely that ‘storage getting short’ and ‘storage critical’ occur many times for every occasion SOS is reached. If you want to see how often these points are reached, select level-2 tracing for the dispatcher domain and look out for trace point IDs **DS 0038** (‘storage getting short’) and **DS 0039** (‘storage critical’). Trace point **DS 0040** shows that storage is OK.

A summary of the effects of ‘storage getting short’, ‘storage critical’, and SOS is given in Table 23.

Table 23. How storage conditions affect new tasks getting started

State of storage	Effects on user tasks
Storage getting short	Priority of new user tasks reduced a little
Storage critical	Priority of new user tasks reduced considerably

[3] This sentence is untrue if the SIT parameter PRTYAGE is set to 0.

Finding out why tasks take a long time to complete

When a ready task is dispatched, it becomes a running task. It is unlikely to complete without being suspended at least once, and it is likely to go through the 'READY - RUNNING - SUSPENDED' cycle several times during its lifetime in the dispatcher.

The longer the task spends in the non-running state, either 'ready' or 'suspended', the greater your perception of performance degradation. In extreme cases, the task might spend so long in the non-running state that it is apparently waiting indefinitely. It is not likely to remain 'ready' indefinitely without running, but it could spend so long suspended that you would probably classify the problem as a wait.

The purpose of this section is to deal not with waiting tasks, but instead with tasks that complete more slowly than they should.

Here are some factors that can affect how long tasks take to complete:

- System loading
- Time-out interval for tasks
- Distribution of data sets on DASD volumes.

Each of these factors is considered in turn.

The effect of system loading on performance

The most obvious factor affecting the time taken for a task to complete is system loading. For more information, see the *CICS/ESA Performance Guide*. Note in particular that there is a critical loading beyond which performance is degraded severely for only a small increase in transaction throughput.

The effect of task time-out interval on performance

The time-out interval is the length of time a task can wait on a resource before it is removed from the suspended state. A transaction that times out is normally abended.

Any task in the system can use resources and not allow other tasks to use them. Normally, a task with a large time-out interval is likely to hold on to resources longer than a task with a short time-out interval. Such a task has a greater chance of preventing other tasks from running. It follows that task time-out intervals should be chosen with care, to optimize the use of resources by all the tasks that need them.

The distribution of data sets on DASD volumes

CICS uses QSAM to write data to extrapartition transient data destinations, and QSAM uses the MVS RESERVE mechanism. If the destination happens to be a DASD volume, any other CICS regions trying to access data sets on the same volume are held up until the TD WRITE is complete.

Other system programs also use the MVS RESERVE mechanism to gain exclusive control of DASD volumes, making the data sets on those volumes inaccessible to other regions.

If you notice in particular that tasks making many file accesses take a long time to complete, check the distribution of the data sets between DASD volumes to see if volume locking could be the cause of the problem.

A summary of performance bottlenecks, symptoms, and causes

Table 24 contains a summary of potential performance bottlenecks, the symptoms you get if they are restricting the performance of your system, and the specific causes of the delays at each point.

Table 24. A summary of performance bottlenecks, symptoms and causes

Bottleneck	Symptoms	Possible causes
Initial attach to transaction manager	CEMT INQ TASK does not know task. Tracing shows long wait for attach to transaction manager.	Interval on EXEC CICS START too long Terminal not available Remote system not available
Initial attach to dispatcher	CEMT INQ TASK shows wait on MXT or transaction class. Tracing shows long wait for attach to dispatcher.	MXT or transaction class limits set too low
First dispatch	Performance class monitoring shows long wait for first dispatch. Storage statistics show CICS has gone SOS.	MXT or transaction class limits set too low Priority of task set too low Insufficient storage System under stress, or near it
SUSPEND / RESUME cycle	Tasks take a long time to complete.	System loading high Task time-out interval too large CICS data sets are on volumes susceptible to MVS RESERVE locking

Chapter 9. Dealing with incorrect output

The term “incorrect output” can be interpreted in many different ways, and its meaning for the purpose of problem determination with this book is explained in Chapter 2, “Classifying the problem” on page 11.

The various categories of incorrect output are dealt with in this chapter:

- “Trace output is incorrect”
- “Dump output is incorrect” on page 167
- “Wrong data has been displayed on a terminal” on page 171
- “Incorrect data is present on a VSAM data set” on page 179
- “An application didn’t work as expected” on page 179
- “Your transaction produced no output at all” on page 180
- “Your transaction produced some output, but it was wrong” on page 185.

Trace output is incorrect

If you have been unable to get the trace output you need, you can find guidance about solving the problem in this section. You can be very selective about the way CICS does tracing, and the options need to be considered carefully to make sure you get the tracing you want.

There are two main types of problem:

- Your tracing might have gone to the wrong destination. This is dealt with in “Tracing has gone to the wrong destination.”
- You might have captured the wrong data. This is dealt with in “You have captured the wrong trace data” on page 164.

Tracing has gone to the wrong destination

In terms of destinations, CICS system trace entries belong to one of three groups:

- CICS trace entries, other than CICS VTAM exit traces and exception traces, go to any of the following trace destinations that are currently active:
  - The internal trace table
  - The current auxiliary trace data set
  - The GTF trace data set.
- CICS VTAM exit traces that are *not* exception traces go only to the GTF trace data set, if GTF tracing is active.
- CICS VTAM exit traces that *are* exception traces go to the internal trace table and, if GTF tracing is active, to the GTF trace data set.
- All other CICS exception traces go to the internal trace table and to any other trace destination that is currently active.

For CICS system tracing other than exception traces and CICS VTAM exit traces, you can inquire on the current destinations and set them to what you want using the CETR transaction.

Figure 43 on page 240 illustrates what you might see on a CETR screen, and indicates how you can change the options by overtyping the fields. From that illustration you can see that, from the options in effect, a normal trace call results in a trace entry being written to the GTF trace destination. If an exceptional condition occurred, the corresponding exception trace entry would be made both to the GTF data set and to the internal trace table, even though the internal trace status is STOPPED.

Note that the master system trace flag value only determines whether standard tracing is to be done for a task (see Figure 41 on page 234). It has no effect on any other tracing status.

Internal tracing goes to the internal trace table in main storage. The internal trace table is used as a buffer in which the trace entries are built no matter what the destination. It, therefore, always contains the most recent trace entries, even if its status is STOPPED—if at least one of the other trace destinations is currently STARTED.

Auxiliary tracing goes to one of two data sets, if the auxiliary tracing status is STARTED. The current data set can be selected from the CETR screen by overtyping the appropriate field with A or B, as required. What happens when the data set becomes full is determined by the auxiliary switch status. Make sure that the switch status is correct for your system, or you might lose the trace entries you want, either because the data set is full or because they are overwritten.

GTF tracing goes to the GTF trace data set. GTF tracing must be started under MVS, using the TRACE=USR option, before the trace entry can be written. Note that if GTF tracing has not been started in this way, the GTF tracing status can be shown as STARTED on the CETR screen and yet no trace entries are made, and no error condition reported.

The way in which trace entries are directed to the required destinations is illustrated in Figure 44 on page 241.

You have captured the wrong trace data

There are several ways in which you might capture the wrong trace data. The following are some sets of symptoms that suggest specific areas for attention:

1. You are not getting the right task tracing, because:
   - Tasks do not trace the right trace points for some components.
   - Transactions are not being traced when they are started from certain terminals.
   - There is no tracing for some terminals that interest you.

   If you are aware of symptoms like these, it is likely that you do not have the right task tracing options set up. Turn to “You are not getting the correct task tracing” on page 165 for further guidance.
2. You are getting the wrong amount of data traced, because:
   - Tracing is not being done for all the components you want, so you are getting too little information.
   - Tracing is being done for too many components, so you are getting more information than you want.

- You are not getting the right trace points (level-1 or level-2) traced for some of the components.
- Tasks are not tracing the component trace points you want. This evidence suggests CICS component tracing selectivity is at fault.

If your observations fit any of these descriptions, turn to "You are not getting the correct component tracing" for guidance about fixing the problem.

3. The data you want is missing entirely from the trace table.

   If you have this sort of problem, turn to "The entries you want are missing from the trace table" on page 166 for guidance about finding the cause.

It is worth remembering that the more precisely you can define the trace data you need for any sort of problem determination, the more quickly you are likely to get to the cause of the problem.

You are not getting the correct task tracing

If you are not getting the correct task tracing, use the CETR transaction to check the transaction and terminal tracing options, and if necessary change them.

You can define whether you want standard or special CICS tracing for specific transactions, and standard or special tracing for transactions started at specific terminals. You can also suppress tracing for transactions and terminals that don't interest you. The type of task tracing that you get (standard or special) depends on the type of tracing for the corresponding transaction-terminal pair, in the way shown in Figure 39 on page 232.

You can deduce from the table that it is possible to get standard tracing when a transaction is initiated at one terminal, and special tracing when it is initiated from another terminal. This raises the possibility of setting up inappropriate task tracing options, so the trace entries that interest you—for example, when the transaction is initiated from a particular terminal—are not made.

You are not getting the correct component tracing

If you are not getting the correct component tracing, use the CETR transaction to inquire on the current component tracing options and, if necessary, to change them.

First, check that you are only tracing components that interest you. If some other components are being traced, change the options so they are no longer traced for standard tracing or for special tracing, as appropriate.

Next, check that the right tracing levels have been defined for standard tracing and special tracing. Remember that, whenever a task that has standard tracing is running, the trace points that you have defined as standard for a component are traced whenever that component is invoked. Similarly, special trace points are traced whenever special task tracing is being done.

Figure 41 on page 234 illustrates the logic used to determine whether a trace call is to be made from a trace point.

If you are satisfied that the component tracing selectivity is correct but you are still getting too much or too little data, read "You are not getting the correct task tracing."

The entries you want are missing from the trace table

Read this section if one or more entries you were expecting were missing entirely from the trace table.

These cases are considered:

- The trace data you wanted didn't appear at the expected time.
- The earliest trace entry in the table was time-stamped after the activity that interested you took place.
- You could not find the exception trace entry you were expecting.

If the trace entry did not appear at the expected time, consider these possibilities:

- If tracing for some components or some tasks did not appear, you might not have set up the tracing selectivity correctly. For guidance about checking and correcting the options, refer to the immediately preceding sections "You are not getting the correct task tracing" and "You are not getting the correct component tracing."
- If you were using GTF tracing, it might not have been active at the time the trace entry should have been made. GTF tracing must be started under MVS using the TRACE=USR option.
- If CICS VTAM exit trace entries (point IDs AP FCxx) were missing, remember that they are only ever made to the GTF trace data set.
- If you attempted to format the auxiliary trace data set selectively by transaction or terminal, and trace entries for the transaction or terminal were missing entirely, it could be that you did not capture the corresponding "transaction attach" (point ID AP F004, KC level-1) trace entry.

  When you select trace entries by specifying TRANID or TERMID parameters in the DFHTU410 trace control statements, DFHTU410 searches for any transaction attach trace entries that contain the specified TRANID or TERMID. It then formats any associated trace entries, identified by the TASKID found in the transaction attach trace entry data.

  It follows that you must have KC level-1 tracing selected for the task in question at the time it is attached if you want to format the auxiliary trace data set selectively by transaction or terminal.

  For more details about trace formatting using DFHTU410, see the *CICS/ESA Operations and Utilities Guide*.

If the options were correct and tracing was running at the right time, but the trace entries you wanted did not appear, it is likely that the task you were interested in did not run or did not invoke the CICS components you expected. Examine the trace carefully in the region in which you expected the task to appear, and attempt to find why it was not invoked. Remember also that the task tracing options might not, after all, have been appropriate.

If the earliest trace entry was later than the event that interested you, and tracing was running at the right time, it is likely that the trace table wrapped round and earlier entries were overwritten.

Internal trace always wraps when it is full. Try using a bigger trace table, or direct the trace entries to the auxiliary trace or GTF trace destinations.

Note: Changing the size of the internal trace table during a run causes the data that was already there to be destroyed. In such a case, the earliest data would have been recorded after the time when you redefined the table size.

Auxiliary trace switches from one data set to the next when it is full, if the autoswitch status is NEXT or ALL.

If the autoswitch status is NEXT, the two data sets can fill up but earlier data cannot be overwritten. Your missing data might be in the initial data set, or the events you were interested in might have occurred after the data sets were full. In the second case, you can try increasing the size of the auxiliary trace data sets.

If the autoswitch status is ALL, you might have overwritten the data you wanted. The initial data set is reused when the second extent is full. Try increasing the size of the auxiliary trace data sets.

GTF trace always wraps when the data set is full. If this was your trace destination, try increasing the size of the GTF trace data set.

If you cannot find an exception trace entry that you expected, bear in mind that exception tracing is always done to the internal trace table irrespective of the status of any other type of tracing. So, if you missed it in your selected trace destination, try looking in the internal trace table.

Dump output is incorrect

Read this section if you did not get the dump output you were expecting.

The things that can go wrong are:

- The dump you got did not seem to relate to the CICS region in which you were interested.
- You did not get a dump when an abend occurred.
- Some dump IDs were missing from the sequence of dumps in the dump data set.
- You did not get the correct data formatted from a system dump.

The sections that follow give guidance about resolving each of these problems in turn.

The dump you got did not seem to relate to your CICS region

If you have experienced this problem, it is likely that you have dumped the wrong CICS region. It should not occur if you are running a single region.

If you invoked the dump from the MVS console using the MVS MODIFY command, check that you specified the correct job name. It must be the job used to bring up the CICS region in which you are interested.

If you invoked the dump from the CICS master terminal using CEMT PERFORM SNAP, check that you were using the master terminal for the correct region. This is more likely to be a problem if you have a VTAM network, because that allows you to switch a single physical VTAM terminal between the different CICS regions.

You did not get a dump when an abend occurred

Read this section if you have experienced any of these problems:

- A transaction abended, but you did not get a transaction dump.
- A transaction abended and you got a transaction dump, but you did not get the system dump you wanted at the same time.
- A system abend occurred, but you did not get a system dump.

There are, in general, two reasons why dumps might not be taken:

- Dumping was suppressed because of the way the dumping requirements for the CICS region were defined. The valid ways that dumping can be suppressed are described in detail in the sections that follow.
- A system error could have prevented a dump from being taken. Some of the possibilities are:
  - No transaction or system dump data sets were available.
  - An I/O error occurred on a transaction or a system dump data set.
  - The system dump data set was being written to by another region, and the DURETRY time was exceeded.
  - There was insufficient space to write the dump in the dump data set. In such a case, you might have obtained a partial dump.

    Depending on the areas that are missing from the dump, the dump formatting program might subsequently be able to format the data that is there, or it might not be able to format the data at all.

  For each of these system errors, there should be a message explaining what has happened. Use the CMAC transaction or refer to the *CICS/ESA Messages and Codes* manual for guidance about the action to take.

How dumping can be suppressed

If you did not get a dump when an abend occurred, and there was no system error, the dumping that you required must somehow have been suppressed. There are several levels at which dumping can be suppressed:

- System dumps can be globally suppressed.
- System dumps and transaction dumps can be suppressed for specific transactions.
- System dumps can be suppressed for specific dump codes from a dump domain global user exit program.
- System dumps and transaction dumps can be suppressed by dump table options.

You need to find out which of these types of dump suppression apply to your system before you decide what remedial action to take.

Global suppression of system dumping

System dumping can be suppressed globally in two ways:

- By coding a value of NO for the DUMP parameter in the system initialization table.
- By using the system programming command EXEC CICS SET SYSTEM DUMPING, with a CVDA value of NOSYSDUMP.

If system dumping has been suppressed globally by either of these means, any system dumping requirements specified in the transaction dump table and the system dump table are overridden.

You can inquire whether system dumping has been suppressed globally by using the EXEC CICS INQUIRE SYSTEM DUMPING system programming command. If necessary, you can cancel the global suppression of system dumping using EXEC CICS SET SYSTEM DUMPING with a CVDA value of SYSDUMP.

Suppression of system dumping from a global user exit program

System dumping can be suppressed for specific dump codes by an XDUREQ user exit program. For programming information about the XDUREQ global user exit program, see the *CICS/ESA Customization Guide*.

If an exit program that suppresses system dumping for a particular dump code is enabled, system dumping is not done for that dump code. This overrides any system dumping requirement specified for the dump code in the dump table.

The exit program can suppress system dumps only while it is enabled. If you want the system dumping suppression to be canceled, you can issue an EXEC CICS DISABLE command for the program. Any system dumping requirements specified in the dump table then takes effect.

Suppression of dumping for individual transactions

Transaction dumps taken when a transaction abends can be suppressed for individual transactions by using the EXEC CICS SET TRANSACTION DUMPING system programming command, or by using the DUMP attribute on the RDO definition of the transaction. None of the dumping requirements specified in the transaction dump table would be met if a transaction for which dumping is suppressed were to abend.

You can use EXEC CICS INQUIRE TRANSACTION DUMPING to see whether dumping has been suppressed for a transaction, and then use the corresponding SET command to cancel the suppression if necessary.

Suppression of dumping by dump table options

If transaction dumping and system dumping are *not* suppressed by any of the preceding mechanisms, the dump table options determine whether or not you get a dump for a particular dump code. For details, see Chapter 15, “Using dumps in problem determination” on page 249.

You can inquire on transaction and system dump code attributes using CEMT INQ TRDUMPCODE and CEMT INQ SYDUMPCODE, respectively. You must specify the dump code you are inquiring on.

If you find that the dumping options are not what you want, you can use CEMT SET TRDUMPCODE code or CEMT SET SYDUMPCODE code to change the values of the attributes accordingly.

- **If you had no transaction dump when a transaction abended**, look first to see if attribute TRANDUMP or NOTRANDUMP is specified for this dump code. The attribute needs to be TRANDUMP if a transaction dump is to be taken.

  If the attribute is shown to be TRANDUMP, look next at the maximum number of dumps specified for this dump code, and compare it with the current number. The values are probably equal, showing that the maximum number of dumps have already been taken.

- **If you had a transaction dump but no system dump**, use CEMT INQ TRDUMPCODE and check whether there is an attribute of SYSDUMP or NOSYSDUMP for the dump code. You need to have SYSDUMP specified if you are to get a system dump as well as the transaction dump.

  Check also that you have not had all the dumps for this dump code, by comparing the maximum and current dump values.

- **If you had no system dump when a system abend occurred**, use CEMT INQ SYDUMPCODE and check whether you have an attribute of SYSDUMP or NOSYSDUMP for the dump code. You need SYSDUMP if you are to get a system dump for this type of abend.

  Finally, check the maximum and current dump values. If they are the same, you need to reset the current value to zero.

Some dump IDs were missing from the sequence of dumps

CICS keeps a count of the number of times that dumping is invoked during the current run, and the count is included as part of the dump ID given at the start of the dump.

Note: SDUMPs produced by the kernel do not use the standard dump domain mechanisms, and **always** have a dump ID of 0/0000.

If both a transaction dump and a system dump are taken in response to the event that invoked dumping, the same dump ID is given to both. However, if just a transaction dump or just a system dump is taken, the dump ID is unique to that dump.

The complete range of dump IDs for any run of CICS is, therefore, distributed between the set of system dumps and the set of transaction dumps, but neither set of dumps has them all.

Figure 23 on page 171 gives an example of the sort of distribution of dump IDs that might occur. Note that each dump ID is prefixed by the run number, in this case 23, and that this is the same for any dump produced during that run. This does not apply to SDUMPs produced by the kernel; these **always** have a dump ID of 0/0000.

On system dump data set	On transaction dump data set
ID=23/0001	
ID=23/0002	ID=23/0002
	ID=23/0003
ID=23/0004	
	ID=23/0005
ID=23/0006	
ID=23/0007	
	ID=23/0008

Figure 23. Typical distribution of dump IDs between dump data sets

For further discussion of the way CICS manages transaction and system dumps, see Chapter 15, “Using dumps in problem determination” on page 249.

You did not get the correct data formatted from a CICS system dump

If you did not get the correct data formatted from a CICS system dump, these are the most likely explanations:

- You did not use the correct dump formatting keywords. If you don’t specify any formatting keywords, the whole system dump is formatted. However, if you specify any keywords at all, you must be careful to specify keywords for *all* the functional areas you are interested in.
- You used the correct dump formatting keywords, but the dump formatting program was unable to format the dump correctly because it detected an error. In such a case, you should be able to find a diagnostic error message from the dump formatter.
- You have CICS systems at different release levels, and you have used the wrong release level of the CICS dump formatting program (DFHPDxxx). For CICS/ESA 4.1, the dump formatting program is DFHPD410. DFHPDxxx is not compatible between CICS releases, so make sure that you select the correct release level for the dump you want to format. See “The DFHIPCSP CICS exit control data” on page 278 for further details.
- A partial dump might have been specified at the MVS level, for example “without LPA”. This requirement would be recorded in the MVS parameter library.

Wrong data has been displayed on a terminal

There are many reasons why you might get the wrong data displayed, some with system-related causes and some with application-related causes. If you think that it is system-related, read this section for some suggestions on likely areas in which to start your investigations.

For the present purpose, a terminal is considered to be any device where data can be displayed. It might be some unit with a screen, or it could be a printer. Many other types of terminals are recognized by CICS, including remote CICS regions, batch regions, IMS regions and so on, but they are not considered in this section on incorrect output.

Broadly, there are two types of incorrect output that you might get on a screen, or on a printer:

- The data information is wrong, so unexpected values appear on the screen or in the hard copy from a printer.
- The layout is incorrect on the screen or in the hard copy. That is, the data is formatted wrongly.

In practice, you may sometimes find it difficult to distinguish between incorrect data information and incorrect formatting. In fact, you seldom need to make this classification when you are debugging this type of problem.

Sometimes, you might find that a transaction runs satisfactorily at one terminal, but fails to give the correct output on another. This is probably due to the different characteristics of the different terminals, and you should find the answer to the problem in the sections that follow.

The preliminary information you need to get

Before you can investigate the reasons why incorrect output is displayed at a terminal, you need to gather some information about the transaction running at the terminal, and the about terminal itself.

The first things you need to know are:

- The identity of the transaction associated with the incorrect output.
- For an autoinstalled terminal, the model number, to ensure that you inquire on the correct TERMTYPE. You can find this from the autoinstall message in the CADL log.

Depending on the symptoms you have experienced, you probably need to examine the PROFILE definitions for the transaction, and the TYPETERM definitions for the affected terminal. The attributes most likely to be of interest are SCRNSIZE for the PROFILE, and ALTSCREEN, ALTPAGE, PAGESIZE, EXTENDEDDS, and QUERY for TYPETERM. Other attributes might also be significant, but the values you find for the attributes named here can often explain why the incorrect output was obtained.

Specific types of incorrect output, and their possible causes

This chapter contains some suggestions about what to do for specific types of incorrect output, and what might be at fault.

Logon rejection message

If you get a logon rejection message when you attempt to log on to CICS, it could be that the TYPETERM definitions for the terminal are incorrect. A message recording the failure is written to the CSNE log or, in the case of autoinstall, to the CADL log.

You are likely to get a logon rejection if you attempt to specify anything other than QUERY(NO) for a terminal that does not have the structured query field feature. Note that NO is the default value for TYPETERM definitions that you supply, but YES is the value for TYPETERM definitions that are supplied with CICS.

If you have a persistent problem with logon rejection, you can use the VTAM buffer trace to find out more about the reasons for the failure.

Unexpected messages and codes

If the “wrong data” is in the form of a message or code that you don’t understand, look in the appropriate manual for an explanation of what it means.

Messages that are prefixed by DFH originate from CICS—use the CMAC transaction or look in the *CICS/ESA Messages and Codes* manual for these. For codes that appear in the space at the bottom of the screen where status information is displayed, look in the appropriate guide for the terminal.

The following are examples of common errors that can cause messages or codes to be displayed:

- SCRNSIZE(ALTERNATE) has been specified in a PROFILE, and too many rows have been specified for ALTSCREEN and ALTPAGE in the TYPETERM definition for the terminal.
- An application has sent a spurious hex value corresponding to a control character in a data stream. For example, `X'11'` is understood as “set buffer address” by a 3270 terminal, and the values that follow are interpreted as the new buffer address. This eventually causes an error code to be displayed.

  If you suspect this may be the cause of the problem, check your application code carefully to make sure it cannot send any unintended control characters.
- EXTENDEDDS(YES) has been specified for a device that does not support this feature. In such a case, a message is sent to the screen, and a message might also be written to the CSMT log.

  The default value for EXTENDEDDS is NO, but check to make sure that YES hasn’t been specified if you know your terminal is not an extended data stream device.

Unexpected appearance of uppercase or lowercase characters

If the data displayed on your terminal has unexpectedly been translated into uppercase characters, or if you have some lowercase characters when you were expecting uppercase translation, you need to look at the options governing the translation.

These are the significant properties of the various translation options you have:

- The ASIS option for BMS or terminal control specifies that lowercase characters in an input data stream are *not* to be translated to uppercase.

  ASIS overrides the UCTRAN attributes for both TYPETERM and PROFILE definitions.
- The UCTRAN attribute of the TYPETERM definition states whether lowercase characters in 3270 input data streams are to be translated to uppercase for terminals with this TYPETERM definition.

  The UCTRAN attribute of TYPETERM is overridden by ASIS, but it overrides the UCTRAN attribute of a PROFILE definition.
- The UCTRAN attribute of a PROFILE states whether lowercase characters in the input data stream are to be translated to uppercase for transactions with

this PROFILE running on VTAM terminals. The PROFILE UCTRAN value is valid only for VTAM terminals.

The UCTRAN option for a PROFILE is overridden by both the UCTRAN option for a TYPETERM definition and the BMS or terminal control ASIS option.

- If the ASIS option is NOT specified, then if either the PROFILE or the TYPETERM definitions specify UCTRAN(YES), the data presented to the transaction IS translated.

Note: User exit XZCIN can also be used to perform uppercase translation.

The UPPERCASE option in the offline utilities (DFHSTUP, DFHDU410, DFHTU410) specify whether all lowercase characters are to be translated to uppercase characters.

Figure 24 and Figure 25 summarize whether or not you get uppercase translation, depending on the values of these options.

ASIS option **not** specified		**TYPETERM**	
		UCTRAN(YES)	UCTRAN(NO)
PROFILE	UCTRAN(YES)	YES	YES
	UCTRAN(NO)	YES	NO

Figure 24. Uppercase translation truth table – ASIS option not specified

ASIS option **is** specified		**TYPETERM**	
		UCTRAN(YES)	UCTRAN(NO)
PROFILE	UCTRAN(YES)	NO	NO
	UCTRAN(NO)	NO	NO

Figure 25. Uppercase translation truth table – ASIS option is specified

The UPPERCASE option in the offline utilities (DFHDUP, DFHSTUP, and DFHTUP) specifies whether all lowercase characters are to be translated to uppercase.

CRTE and uppercase translation

Initiating a CRTE session

The input required to start a CRTE routing session is of the form:

```
CRTE SYSID(xxxx),TRPROF(yyyyyyyy)
```

Translation to uppercase is dictated by the typeterm of the terminal at which CRTE was entered and CRTE's transaction profile definition as shown in Table 25.

Table 25. Uppercase translation on CRTE session initiation

TYPETERM UCTRAN	CRTE PROFILE UCTRAN	INPUT TRANSLATED TO UPPERCASE
YES	YES/NO	ALL OF THE INPUT
NO	NO	NONE OF THE INPUT. See note.
NO	YES	ALL OF THE INPUT EXCEPT THE TRANSID. See note.
TRANID	YES	ALL OF THE INPUT
TRANID	NO	TRANSID ONLY

Note: Note that if the transid CRTE is not entered in upper case, it will not be recognized (unless there is a lower/mixed case alias), and message DFHAC2001 will be issued.

Input within the CRTE session

During the CRTE routing session, uppercase translation is dictated by the typeterm of the terminal at which CRTE was initiated and the transaction profile definition of the transaction being initiated (which has to be a valid transaction on the application owning region) as shown in Table 26.

Table 26. Uppercase translation during CRTE session

TYPETERM UCTRAN	TRANSACTION PROFILE (AOR) UCTRAN	INPUT TRANSLATED TO UPPERCASE
YES	YES/NO	ALL OF THE INPUT
NO	NO	NONE OF THE INPUT. See note.
NO	YES	ALL OF THE INPUT EXCEPT THE TRANSID. See note.
TRANID	YES	ALL OF THE INPUT
TRANID	NO	TRANSID ONLY

Note: Note that if the transid CRTE is not entered in upper case, it will not be recognized (unless there is a lower/mixed case alias defined on the AOR) and message DFHAC2001 will be issued.

During a CRTE routing session, if the first six characters entered at a screen are `CANCEL`, CICS will recognize this input in upper, lower or mixed case and end the routing session.

For more information on the ALIAS attribute of the transaction definition, see the 'Transaction' section of the *CICS/ESA Resource Definition Guide*.

Be aware that when transaction routing from CICS/ESA 4.1 to an earlier release that does not support transaction based uppercase translation, uppercase translation only occurs if it is specified in the typeterm.

EXEC CICS SET TERMINAL and uppercase translation

In a single system, if the EXEC CICS SET TERMINAL command is issued for a terminal while it is running a transaction performing RECEIVE processing, unpredictable results may occur. This is because the command can override the typeterm definition regarding uppercase translation and RECEIVE processing interrogates the uppercase translate status of the terminal in order to establish whether translation is required.

In a transaction routing environment, the system programmer who issues the EXEC CICS SET TERMINAL command should be aware (for VTAM terminals) that the TOR terminal uppercase translate status is copied to the AOR surrogate terminal on every flow across the link from the TOR to the AOR. Consequently:

- The EXEC CICS SET TERMINAL change of uppercase translate status will only take effect on the AOR on the next flow across the link.
- Any AOR typeterm definition used to hard code remote terminal definitions will be overridden with the TOR values for uppercase translate status.
- EXEC CICS INQUIRE TERMINAL issued on the AOR can return misleading uppercase translation status of the terminal, since the correct status on the TOR may not yet have been copied to the AOR.
- The processing of RECEIVE requests on the TOR and AOR can interrogate the uppercase translate status of the terminal. Therefore unpredictable results can also occur if the system programmer issues the EXEC CICS SET TERMINAL command during receive processing.

Katakana terminals – mixed English and Katakana characters

If you are using a Katakana terminal, you might see some messages containing mixed English and Katakana characters. That is because Katakana terminals cannot display mixed-case output. Uppercase characters in the data stream appear as uppercase English characters, but lowercase characters appear as Katakana characters. If you have any Katakana terminals connected to your CICS system, specify MSGCASE=UPPER in the system initialization table to ensure that messages contain uppercase characters only.

The offline utilities DFHSTUP, DFHDU410 and DFHTU410 have an extra parameter to ensure all output is translated to uppercase. See the *CICS/ESA Operations and Utilities Guide* for details on how to use these parameters.

Wrong data values are displayed

If the data values are wrong on the user's part of the screen (the space above the area used to display status information to the operator), or in the hard copy produced by a printer, it is likely that the application is at fault.

Some data is not displayed

If you find that some data is not being displayed, consider these possibilities:

- The SENDSIZE value for the TYPETERM definition could be too large for the device receiving the data. Its receiving buffer could then overflow, with some data being lost.
- SCRNSIZE(ALTERNATE) might be specified in the PROFILE definition for the transaction running at the terminal, while default values for ALTSCREEN and ALTPAGE are allowed in the TYPETERM definition for the terminal.

  The default values for ALTSCREEN and ALTPAGE are 0 rows and 0 columns, so no data could then be displayed if SCRNSIZE(ALTERNATE) were specified.
- EXTENDEDDS(YES) is specified for a device that does not support this feature.

Early data is overlaid by later data

Early data can be overlaid by later data, so that data appears in the wrong order, when the SENDSIZE value of the TYPETERM definition is too large for the device receiving the data. This is because the buffer can wrap when it is full, with the surplus data overlaying the first data that was received.

The data is formatted wrongly

Incorrect formatting of data can have a wide range of causes, but here are some suggestions of areas that can sometimes be troublesome:

- BMS maps are incorrect.
- Applications have not been recompiled with the latest maps.
- Different numbers of columns have been specified for ALTSCREEN and ALTPAGE in the TYPETERM definitions for the terminal. This can lead to unpredictable formatting errors. However, you will not see them unless SCRNSIZE(ALTERNATE) has been specified in the PROFILE for the transaction running at the terminal.
- The PAGESIZE values included in the TYPETERM definitions must suit the characteristics of the terminal, or you get formatting errors.

  For a screen display, the number of columns specified must be less than or equal to the line width. For a printer, the number of columns specified must be *less than* the line width, or else both BMS (if you are using it) and the printer might provide a new line and you will get extra spacing you don't want.

  The default values for PAGESIZE depend on the value you specify for the DEVICE keyword.
- If you get extra line feeds and form feeds on your printer, it could be that an application is sending control characters that are not required because the printer is already providing end of line and end of form operations.

  If your application is handling the buffering of output to a printer, make sure that an "end of message" control character is sent at the end of every buffer full of data. Otherwise, the printer might put the next data it receives on a new line.

Tools for debugging terminal output in a VTAM environment

Amongst the debugging tools you have, two are likely to be of particular use for investigating terminal incorrect output errors in a VTAM environment. They are:

- VTAM buffer trace. This is a function of VTAM itself, and you need to refer to the appropriate manual in the VTAM library to find out how to use it.
- CICS VTAM exit trace. This is a function of CICS, and you can control it from the CETR panel.

For a description of the use of these two types of tracing in CICS problem determination, see Chapter 14, “Using traces in problem determination” on page 219.

Incorrect data is present on a VSAM data set

If READ UPDATE is not used, an error can occur because VSAM allows a record to be read by one transaction while another transaction is updating it.

If the first transaction were to take some action based on the value of the record, the action would probably be erroneous.

For example, in inventory control, a warehouse has 150 items in stock. 100 items are sold to a customer, who is promised delivery within 24 hours. The invoice is prepared, and this causes a transaction to be invoked that is designed to read the inventory record from a VSAM data set and updates it accordingly.

In the meantime, a second customer also asks for 100 items. The salesperson uses a terminal to inquire on the number currently in stock. The “inquire” transaction reads the record that has been read for update but not yet rewritten, and returns the information that there are 150 items. This customer, too, is promised delivery within 24 hours.

Errors of this kind are prevented by the use of READ UPDATE.

An application didn’t work as expected

It is not possible to give specific advice on dealing with this sort of problem, but the points and techniques that follow should help you to find the area where the failure is occurring.

General points for you to consider

1. Make sure you can define exactly what happened, and how this differs from what you expected to happen.
2. Check the commands you are using for accuracy and completeness. For programming information about EXEC CICS commands, see the *CICS/ESA Application Programming Reference* manual. Are any default values the ones you really want? Does the description of the effect of each command match your expectations?
3. Can you identify a failing sequence of commands? If so, can it be reproduced using CECI?

4. Consider the resources required by the application. Are they defined as expected?
5. Are the required functions in the failing functional area available in this system?
6. For “input” type requests, does the item exist? You can verify this using offline utilities.
7. For “output” type requests, is the item created? Verify that the before and after images are as expected.

Using traces and dumps

Traces and dumps can give you valuable information about unusual conditions that might be causing your application to work in an unexpected way.

1. If the path through the transaction is indeterminate, insert user trace entries at all the principal points.
2. If you know the point in the code where the failure occurs, insert a CICS system dump request immediately after it.
3. Use CETR to select special tracing for the level-1 trace points for all components. Select special tracing for the failing task only, and disable all standard tracing by setting the master system trace flag off.
4. Run the transaction after setting the trace options, and wait until the system dump request is executed. Format the internal trace table from the dump (formatting keyword TR), and examine the trace entries before the failure. Look in particular for unusual or unexpected conditions, possibly ones that the application is not designed to handle.

Your transaction produced no output at all

If your transaction produced no output at all, you need to carry out some preliminary checks before looking at the problem in detail. You might be able to find a simple explanation for the failure.

Are there any messages explaining why there is no output?

Look carefully in each of the transient data destinations CSMT, CSTL, and CDBC for any messages that might relate to the task. You could find one there that explains why you received no output.

If you can find no such message, the next step is to get some information about the status of the transaction that produced no output, your terminal, and the CICS system.

Can you use the terminal where the transaction should have started?

Go to the terminal where the transaction should have started, and note whether the keyboard is locked. If it is, press RESET. Now try issuing CEMT INQ TASK (or your site replacement) from the terminal.

If you can’t issue CEMT INQ TASK from the terminal, one of these explanations applies:

- The task that produced no output is still attached to the terminal.
- The terminal where you made the inquiry is not in service.

- There is a system-wide problem.
- You are not authorized to use the CEMT transaction. (This may be because you have not signed on to the terminal and the CEMT transaction is not authorized for that terminal. If you ***have*** signed on to the terminal, you are probably authorized to use CEMT.)

Try to find a terminal where you can issue CEMT INQ TASK. If no terminal seems to work, there is probably a system-wide problem. Otherwise, see if the task you are investigating is shown in the summary.

- If the task is shown, it is probably still attached, and either looping or waiting. Turn to "No output – what to do if the task is still in the system" on page 181 to see what to do next.
- If the task is not shown, there is a problem with the terminal where you first attempted to issue CEMT INQ TASK.

If you are able to issue CEMT INQ TASK from the terminal where the transaction was attached, one of these explanations applies:

- The transaction gave no output because it never started.
- The transaction ran without producing any output, and terminated.
- The transaction started at another terminal, and might still be in the system. If it is still in the system, you can see it in the task summary that you got for CEMT INQ TASK. It is probably looping or waiting. Turn to "No output – what to do if the task is still in the system" on page 181 for advice about what to do next. If you don't see the task in the summary, turn to "No output – what to do if the task is not in the system" on page 181.

No output – what to do if the task is still in the system

If you obtained no output and the task is still in the system, it is either waiting for a resource, or looping. You should get an indication of which of these two conditions is the most likely from the status for the task returned by CEMT INQ TASK.

You have a suspended task, treat this as a "wait" problem. Use the techniques of Chapter 6, "Dealing with waits" on page 57 to investigate it further.

You have a running task, it is likely to be looping. Turn to Chapter 7, "Dealing with loops" on page 143 to find out what to do next.

No output – what to do if the task is not in the system

If you have obtained no output and CEMT INQ TASK shows the task is not in the system, one of two things could have happened:

- Your transaction never started.
- Your transaction ran, but produced no output.

  Note: If you're not getting output on a printer, the reason could be simply that you are not setting on the START PRINTER bit in the write control character. You need to set this bit to get printed output if you have specified the STRFIELD option on a CONVERSE or SEND command, which means that the data area specified in the FROM option contains structured fields. Your application must set up the contents of the structured fields.

Your task might have been initiated by direct request from a terminal, or by automatic task initiation (ATI). Most of the techniques apply to both sorts of task, but there are some extra things to investigate for ATI tasks. Carry out the tests which apply to all tasks first, then go on to the tests for ATI tasks if you need to.

Did the task run? Techniques for all tasks

There are many different techniques for finding out if a transaction started, or if it ran but produced no output. Use the ones that are most convenient at your installation.

Using CICS system trace entry points:

CICS system tracing is probably the most powerful technique for finding out whether a transaction ever started. You might need to direct the trace output to the auxiliary trace destination, depending on how certain you can be about the time the task is expected to start. Even a large internal trace table might wrap and overlay the data you want to see if you are not too sure about when the task should start.

You need to use the CETR transaction to set up the right tracing options. You may need to refer to Chapter 14, “Using traces in problem determination” on page 219 for guidance about setting up trace options.

Select special tracing for just your task, and disable tracing for all other tasks by setting the master system trace flag off. Set up special tracing for the level one trace points for the components that are likely to be used during the invocation of the task. The components you choose will depend on how the task is initiated—by direct request from a terminal, or by automatic transaction initialization—but they should include loader domain (LD), program manager (PG), transaction manager (XM), and dispatcher domain (DS). Make sure that special tracing is disabled for all other components, to minimize the amount of trace data that is collected and the tracing overhead.

Now turn tracing on, and attempt to start your task. When you are sure that the time has passed when the output should have appeared, stop tracing, and format the trace data set.

If your transaction ran, you should see the following types of trace entries for your task and the programs associated with it:

1. Loader domain, when it loaded your program, if the program was not already in main storage.
2. Transaction manager, when it attached your task to the dispatcher.
3. Dispatcher domain, when your task got its first dispatch. You might also see subsequent entries showing your task being suspended, and then resumed.
4. Program manager, for any program management functions associated with your task.

If trace entries for any of these processes are missing, that should help you to find where the failure occurred.

Using EDF

If the transaction being tested requires a terminal, you can use EDF. You need two other terminals for input, as well as the one that the transaction requires ("tttt"). Use one of these others to put the transaction terminal under control of EDF, with:

```
CEDF tttt
```

At the remaining terminal, enter whatever transaction or sequence of transactions causes the one under test to be initiated. Wait long enough for it to start. If no output appears at the second terminal, the transaction hasn't started. If you haven't yet done so, consider using trace to get more information about the failure.

Using statistics

If no one else is using the transaction in question, you can tell from CICS statistics whether the program has been executed or not.

Use the command CEMT PERFORM STATISTICS RECORD (or your site replacement) before you test your transaction, using the TRANSACTION option:

```
CEMT PERFORM STATISTICS RECORD
[TRANSACTION]
```

This causes statistics on transactions that have been executed to be recorded in the SMF data set.

Now initiate the transaction and wait until it should have been executed. Repeat the CEMT PERFORM STATISTICS RECORD command, to get a new set of statistics written to the SMF data set. Format the data from the SMF data set for the APPLID that interests you, and look at the statistics recorded before and after you attempted to execute the transaction. If the count for your transaction increased by 1, it was executed. If it remained the same, it was not executed.

Alternatively, if no one else is using the transaction, you can tell, using CEMT, whether the program is being executed. Use the command CEMT INQUIRE PROGRAM(xxxxxxxx) where xxxxxxxx is the program name. The screen presented to you includes a USECOUNT value. This value is the number of times that the program has been executed since the start of the current CICS session.

Now initiate the transaction and wait until it should have been executed. Repeat the CEMT INQUIRE PROGRAM(xxxxxxxx) and the USECOUNT value will have been incremented if the program has been executed.

Formatting the SMF data set. The statistics utility program, DFHSTUP, prepares and prints reports offline using the data recorded in the SMF data set. Read the 'DFHSTUP' section in the *CICS/ESA Operations and Utilities Guide* for details on how to use this facility.

When you format the SMF data set using DFHSTUP in order to look at the statistics relating to executed transactions and programs, you may find the following DFHSTUP control parameters useful:

SELECT APPLID=

COLLECTION TYPE=REQ

TIME START= ,STOP=

DATE START= ,STOP=

See the 'DFHSTUP' section in the *CICS/ESA Operations and Utilities Guide* for details on how to code these parameters. If you correctly code these control parameters, you avoid the formatting of much information that might be unnecessary at this point.

Using CEBR

You can use CEBR to investigate your transaction if the transaction reads or writes to a transient data queue, or writes to a temporary storage queue. A change in such a queue is strong evidence that the transaction ran, provided that the environment is sufficiently controlled that nothing else could produce the same effect. You need to be sure that no other transaction that might be executed while you are doing your testing does the same thing.

The absence of such a change does not mean that the transaction didn't run—it may have run incorrectly, so that the expected change was not made.

Using CECI

If your transaction writes to a file, you can use CECI before and after the transaction to look for evidence of the execution of your transaction. A change in the file means the transaction ran. If no change occurred, that doesn't necessarily mean that the transaction failed to run—it could have worked incorrectly, so that the changes you were expecting weren't made.

Disabling the transaction

If your transaction requires a terminal, you can do the following. Use CEMT to disable the transaction under test, then do whatever causes the transaction to be initiated. You should get this message at the terminal where it is due to run:

DFHAC2008 *date time applid* **Transaction** *tranid* **has been disabled and cannot be used**

If you don't get this message, it is likely that your transaction didn't start because of a problem with that terminal.

Investigating tasks initiated by ATI

In addition to the general techniques for all tasks described above, there are some additional ones for tasks that should have started by ATI.

Tasks to be started by ATI can be invoked in any of these ways:

- By issuing EXEC CICS START commands, even if no interval is specified
- By BMS ROUTE operations
- By writing to transient data queues with nonzero trigger levels.

There are many reasons why automatically initiated tasks could fail to start. Even when the CICS system is operating normally, an ATI transaction might fail to start for any of the following reasons:

- It might require a resource that is not available. The resource is usually a terminal, although it could be a queue.
- It might not be scheduled to start until some time in the future. START commands and output sent with BMS ROUTE are both subject to this sort of scheduling, but transactions started when transient data trigger levels are reached are not.

CICS maintains two chains for scheduling transactions that have been requested, but not started. They are the interval control element (ICE) chain, and the automatic initiate descriptor (AID) chain. The information contained in one or other of the chains can sometimes indicate why your task has failed to start.

The ICE chain

The ICE chain is used for tasks scheduled to start after some specified interval, for example on an EXEC CICS START command. You can locate it in the formatted system dump by looking at the ICP section. Look in field ICETRNID of each ICE (the 4-character transaction ID) to see if it relates to your task.

If you find an ICE for your task, look in field ICEXTOD. That will show you the expiration time of day. Does it contain the value you expect? If not, either the task which caused this one to be autoinitiated was in error, or there is a system problem.

The AID chain

The AID chain is used for tasks that are due to start immediately. Tasks are moved from the ICE chain to the AID chain as soon as the scheduled time expires, and they are placed there directly if there is no time delay requested. If a task needs a resource, usually a terminal, that is unavailable, the task remains on the AID chain until it can use the resource.

You can see the AIDs in the TCP section of the formatted system dump. Look in field AIDTRNID (the 4-character transaction ID) of each AID, to see if it relates to your task.

If you do find an AID that relates to your task, your task is scheduled to start, but cannot do so because the terminal is unavailable. Look in field AIDTRMID to find the symbolic ID of the terminal, and then investigate why the terminal is not available. One possibility is that the terminal is not in ATI status, because ATI(YES) has not been specified for it in the TYPETERM definition.

Your transaction produced some output, but it was wrong

If your transaction produced no output at all, read “Your transaction produced no output at all” on page 180. For other types of wrong terminal output, read this chapter.

The origins of corrupted data

You get incorrect output to a terminal if data which is the object of the transaction becomes corrupted at some stage.

Figure 26 on page 186 illustrates how data flows between various CICS resources when a transaction is executed, and shows the points at which the data might become invalid.

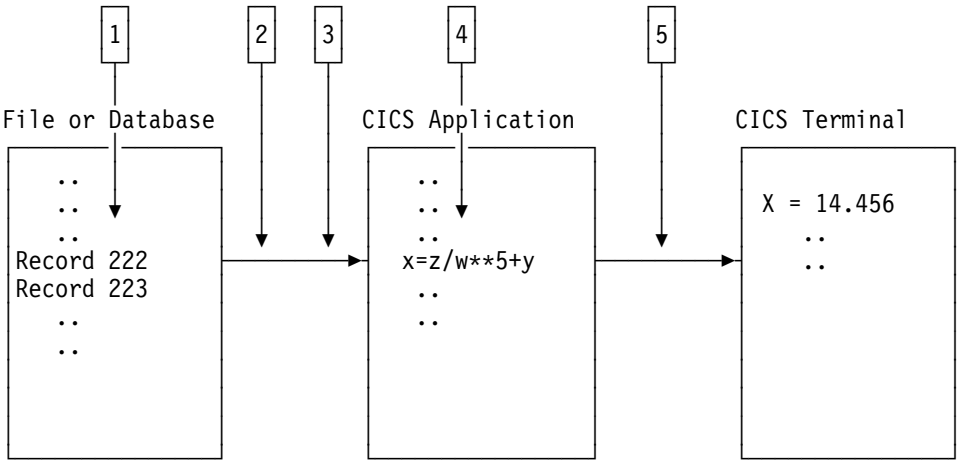


Figure 26. Where data might be corrupted in a transaction

The data might be corrupted at any of points 1 through 5, as it flows from file to terminal.

1. Data records might be incorrect, or they could be missing from the file.
2. Data from the file might be mapped into the program incorrectly.
3. Data input at the terminal might be mapped into the program incorrectly.
4. Bad programming logic might corrupt the data.
5. The data might be mapped incorrectly to the terminal.

Each of these possibilities will be dealt with in turn.

Were records in the file incorrect or missing?

You can check the contents of a file or database either by using CECI or by using a utility program to list off the records in question.

If you find bad data in the file or data set, the error is likely to have been caused by the program that last updated the records containing that data. If the records you expected to see are missing, make sure that your application can deal with a 'record not found' condition.

If the data in the file is valid, it must have been corrupted later on in the processing.

Was the data mapped correctly into the program?

When a program reads data from a file or a database, the data is put into a field described by a symbolic data declaration in the program.

Is the data contained in the record that is read compatible with the data declaration in the program?

Check each field in the data structure receiving the record, making sure in particular that the type of data in the record is the same as that in the declaration, and that the field receiving the record is the right length.

If the program receives input data from the terminal, make sure that the relevant data declarations are correct for that, too.

If there seems to be no error in the way in which the data is mapped from the file or terminal to the program storage areas, the next thing to check is the program logic.

Is the data being corrupted by bad programming logic?

To find out if data is being corrupted by bad programming logic in the application, consider the flow of data through the transaction.

You can determine the flow of data through your transaction by “desk checking”, or by using the interactive tools and tracing techniques supplied by CICS.

Desk checking your source code is sometimes best done with the help of another programmer who is not familiar with the program. It is often possible for such a person to see weaknesses in the code which you have overlooked.

Interactive tools allow you to look at the ways in which the data values being manipulated by your program change as the transaction proceeds.

- CEDF is, perhaps, the most powerful interactive tool for checking your programming logic. You can use it to follow the internal flow from one CICS command-level statement to another. If necessary, you can add CICS statements such as ASKTIME at critical points in your program, to see if certain paths are taken, and to check program storage values.
- CECI allows you to simulate CICS command statements. Try to make your test environment match the environment in which the error occurred as closely as possible. If you don’t, you might find that your program works with CECI, but not otherwise.
- CEBR enables you to look at temporary storage and transient data queues, and to put data into them. This can be useful when many different programs use the queues to pass data.

  Note: When you use CEBR to look at a transient data queue, the records you retrieve are removed from the queue before they are displayed to you. This could alter the flow of control in the program you are testing. You can, however, use CEBR to copy transient data queues to and from temporary storage, as a way of preserving the queues if you need to.

User tracing allows you to trace the flow of control and data through your program, and to record data values at specific points in the execution of the transaction. You could, for example, look at the values of counters, flags, and key variables during the execution of your program. You can include up to 4000 bytes of data on any trace entry, and so this can be a powerful technique for finding where data values are being corrupted.

For programming information about how you can invoke user tracing, see the *CICS/ESA Application Programming Reference* manual.

CSFE storage freeze can be used to freeze the storage associated with a terminal or a transaction so that it is not FREEMAINed at the end of processing. This can be a useful tool if, for example, you want to investigate possible storage violations. You need to get a transaction dump to look at the storage after you have run the task with storage freeze on.

For long-running tasks, there is a possibility that a large amount of storage may be consumed because it cannot be FREEMAINed while storage freeze is on. For short-running tasks, however, there should be no significant overhead.

If, after using these techniques, you can find no fault with the logic of the program, the fault either lies with the way data is mapped to the terminal, or you could have missed some important evidence.

Is the data being mapped incorrectly to the terminal?

Incorrect data mapping to a terminal can have both application-related and system-related causes. If you are using BMS mapping, check the items below.

- Examine the symbolic map very carefully to make sure that it agrees with the map in the load module. Check the date and time stamps, and the size of the map.
- Make sure that the attributes of the fields are what they should be. For example:
  - An attribute of DARK on a field can prevent the data in the field from being displayed on the screen.
  - Failing to turn on the modified data tag (MDT) in a field might prevent that field from being transmitted when the screen is read in.

    Note: The MDT is turned on automatically if the operator types data in the field. If, however, the operator does not type data there, the application must turn the tag on explicitly if the field is to be read in.
- If your program changes a field attribute byte, or a write control character, look at each bit and check that its value is correct by looking in the appropriate reference manual for the terminal.

Chapter 10. Dealing with storage violations

"Avoiding storage violations" describes the CICS facilities for preventing storage violations. The body of the chapter describes problem determination techniques for storage violations.

Avoiding storage violations

CICS provides three facilities that help to prevent storage violations.

CICS subsystem storage protection
: prevents user application programs from directly overwriting CICS code and control blocks.

Transaction isolation
: prevents a user transaction from directly overwriting user application storage of other transactions.

Command protection
: prevents CICS, when processing an EXEC CICS command, from overwriting storage that the issuing transaction could not itself directly overwrite.

Even if your system uses all the CICS storage protection facilities, CICS storage violations can occur in certain circumstances in systems using storage protection. For example:

- An application program could contain the necessary instructions to switch to CICS key and modify CICS storage.
- An application program could contain the necessary instructions to switch to the basespace and modify other transactions' storage.
- An application program could be defined with EXECKEY(CICS) and could thus modify CICS storage and other transactions' storage.
- An application could overwrite one or more storage check zones in its own task-lifetime storage.

To gain the full benefit of CICS storage protection, you need to examine the storage needs of individual application programs and control the storage key definitions that are used.

When CICS detects and prevents an attempted storage violation, the name of the abending program and the address of the area it tried to overwrite are passed to the program error program (DFHPEP). For programming information about DFHPEP, refer to the *CICS/ESA Customization Guide*.

If a storage violation occurs in your system, please read the rest of this chapter.

Two kinds of storage violation

Storage violations can be divided into two classes, namely those detected and reported by CICS, and those not detected by CICS. They require different problem determination techniques.

CICS-detected violations are identified by the following message sent to the console:

DFHSM0102 *applid* **A storage violation (code X'***code***') has been detected by module** *modname*.

If you have received this message, turn first to the description of message DFHSM0102 in the *CICS/ESA Messages and Codes* manual to see an explanation of the message, and then to the *CICS/ESA Diagnosis Reference* manual to see an explanantion of the exception trace point ID, X'*code*'. This tells you how CICS detected the storage violation. Then return to this chapter, and read "CICS has detected a storage violation."

Storage violations not detected by CICS are less easy to identify. They can cause almost any sort of symptom. Typically, you may have got a program check with a condition code indicating 'operation exception' or 'data exception', because the program or its data has been overlaid. Otherwise, you might have obtained a message from the dump formatting program saying that it had found a corrupted data area. Whatever the evidence for the storage violation, if it has not been detected by CICS, turn to "Storage violations that affect innocent transactions" on page 196.

CICS has detected a storage violation

CICS can detect storage violations when:

1. The duplicate storage accounting area (SAA) or the initial SAA of a TIOA storage element has become corrupted.
2. The leading storage check zone or the trailing storage check zone of a user-task storage element has become corrupted.

CICS detects storage violations involving TIOAs by checking the SAA chains when it receives a command to FREEMAIN an individual element of TIOA storage, at least as far as the target element. It also checks the chains when it FREEMAINs the storage belonging to a TCTTE after the last output has taken place. CICS detects storage violations involving user-task storage by checking the storage check zones of an element of user-task storage when it receives a command to FREEMAIN that element of storage. It also checks the chains when it FREEMAINs all the storage belonging to a task when the task ends.

The storage violation is detected *not at the time it occurs*, but *only when the SAA chain or the storage check zones are checked*. This is illustrated in Figure 27 on page 191, which shows the sequence of events when CICS detects a violation of a user task storage element. The sequence is the same when CICS detects a violation of a TIOA storage element.

The fact that the SAA or storage check zone is overlaid some time before it is detected does not matter too much for *user* storage where the trailing storage

check zone has been overlaid, because the transaction whose storage has been violated is also very likely to be the one responsible for the violation. It is fairly common for transactions to write data beyond the end of the allocated area in a storage element and into the check zone. This is the cause of the violation in Figure 27.

The situation could be more serious if the leading check zone has been overlaid, because in that case it could be that some other unrelated transaction was to blame. However, storage elements belonging to individual tasks are likely to be more or less contiguous, and overwrites could extend beyond the end of one element and into the next.

If the leading storage check zone was only overwritten by chance by some other task, the problem might not be reproducible. On other occasions, other parts of storage might be affected. If you have this sort of problem, you need to investigate it as though CICS had not detected it, using the techniques of “Storage violations that affect innocent transactions” on page 196.

Finding the offending transaction when the duplicate SAA of a TIOA storage element has been overlaid might not be so straightforward. This is because TIOAs tend to have much longer lifetimes than tasks, because they wait on the response of terminal operators. By the time the storage violation is detected, the transaction that caused it is unlikely to still be in the system. However, the techniques for CICS-detected violations still apply.

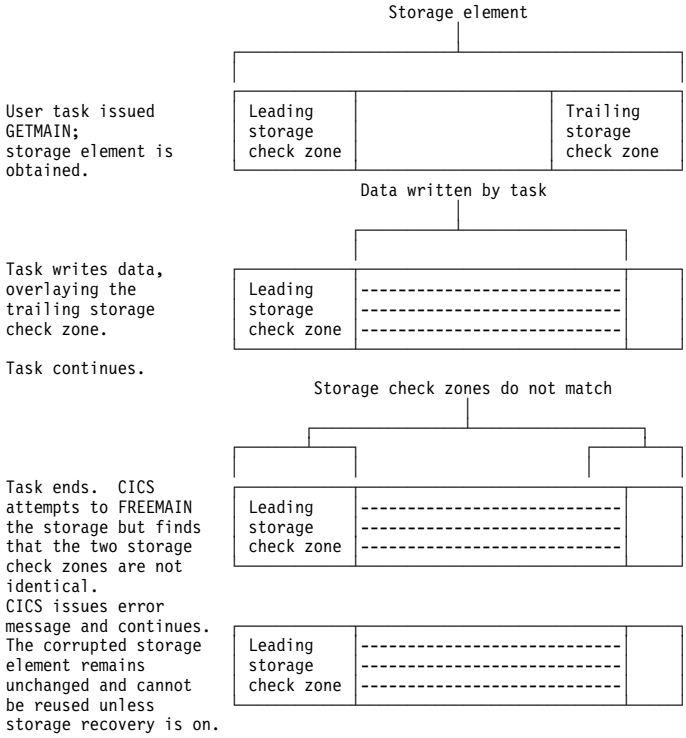


Figure 27. How user-storage violations are committed and detected

Note: For storage elements with SAAs, the address that is returned on the GETMAIN request is that of the leading SAA; for storage elements with storage check zones, the address that is returned is that of the beginning of usable storage.

What happens when CICS detects a storage violation

When CICS detects a storage violation, it makes an exception trace entry in the internal trace table, issues message DFHSM0102 and takes a CICS system dump, unless you have suppressed dumping for system dump code SM0102. If you have suppressed dumping for this dump code, re-enable it and attempt to reproduce the error. The system dump is an important source of information for investigating CICS-detected storage violations.

If storage recovery is on (STGRCVY=YES in the SIT), the corrupted SAAs or check zones are repaired and the transaction continues. See "Storage recovery" on page 198.

If storage recovery is not on, CICS abends the transaction whose storage has been violated (if it is still running). If the transaction is running when the error is detected and if dumping is enabled for the dump code, a transaction dump is taken.

If you received a transaction abend message, read "What the transaction abend message can tell you." Otherwise, go on to "What the CICS system dump can tell you."

What the transaction abend message can tell you

If you get a transaction abend message, it is very likely that CICS detected the storage violation when it was attempting to satisfy a FREEMAIN request for user storage. Make a note of the information the message contains, including:

- The transaction abend code.
- The identity of the transaction whose storage has been violated.
- The identity of the program running at the time the violation was detected.
- The identity of the terminal at which the task was started.

Because CICS doesn't detect the overlay at the time it occurs, the program identified in the abend message probably isn't the one in error. However, it is likely that it issued the FREEMAIN request on which the error was detected. One of the other programs in the abended transaction might have violated the storage in the first place.

What the CICS system dump can tell you

Before looking at the system dump, you must format it using the appropriate formatting keywords. The ones you need for investigating storage violations are:

- TR, to get you the internal trace table
- TCP, to get you terminal-related areas
- AP, to get you the TCAs and user storage.

The dump formatting program reports the damaged storage check zone or SAA chain when it attempts to format the storage areas, and this can help you with diagnosis by identifying the TCA or TCTTE owning the storage.

When you have formatted the dump, take a look at the data overlaying the SAA or storage check zone to see if its nature suggests which program put it there. There are two places you can see this, one being the exception trace entry in the internal trace table, and the other being the violated area of storage itself.

Look first at the exception trace entry in the internal trace table to check that it shows the data overlaying the SAA or storage check zone. Does the data suggest what program put it there? Remember that the program is likely to be part of the violated transaction in the case of user storage. For terminal storage, you probably have more than one transaction to consider.

As the SAAs and storage check zones are only 8 bytes long, there might not be enough data for you to identify the program. In this case, find the overlaid data in the formatted dump. The area is pointed to in the diagnostic message from the dump formatting program. The data should tell you what program put it there, and, more importantly, what part of the program was being executed when the overlay occurred.

If the investigations you have done so far have enabled you to find the cause of the overlay, you should be able to fix the problem.

What to do if you can't find what is overlaying the SAA

The technique described in this section enables you to locate the code responsible for the error by narrowing your search to the sequence of instructions executing between the last two successive old-style trace entries in the trace table.

You do this by forcing CICS to check the SAA chain of terminal storage and the storage check zones of user-task storage every time an old-style trace entry is made from AP domain. These types of trace entry have point IDs of the form AP 00xx, "xx" being two hexadecimal digits. Storage chain checking is not done for new-style trace entries from AP domain or any other domain. (For a discussion of old and new-style trace entries, see Chapter 14, "Using traces in problem determination" on page 219.)

The procedure has a significant processing overhead, because it involves a large amount of tracing. You are likely to use it only when you have had no success with other methods.

How you can force storage chain checking

You can force storage chain checking either by using the CSFE DEBUG transaction, or by using the CHKSTSK or CHKSTRM startup override. Tracing must also be active, or CICS will do no extra checking. The CSFE transaction has the advantage that you need not bring CICS down before you can use it.

Table 27 on page 194 shows the CSFE DEBUG options and their effects. Table 28 shows the startup overrides that have the same effects.

Table 27. Effects of the CSFE DEBUG transaction	
CSFE syntax	**Effect**
CSFE DEBUG, CHKSTSK=ALL	This checks storage check zones of all storage areas on the transaction storage chain for every task. You need to use CSFE DEBUG,CHKSTSK=ALL if one task's storage check zone is being overlaid by another task.
CSFE DEBUG, CHKSTSK=CURRENT	This checks storage check zones for all storage areas on the transaction storage chain for the current task only. If a task is overlaying one of the storage check zones of its own user storage, use CSFE DEBUG,CHKSTSK=CURRENT
CSFE DEBUG, CHKSTRM=CURRENT	This checks SAAs for all TIOAs linked off the current TCTTE. Use this if the SAA of a TIOA has been overlaid.
CSFE DEBUG, CHKSTSK=NONE	This turns off storage zone checking for transaction storage areas.
CSFE DEBUG, CHKSTRM=NONE	This turns off SAA checking for TIOAs.

Table 28. Effects of the CHKSTSK and CHKSTRM overrides	
Override	**Effect**
CHKSTSK=ALL	As CSFE DEBUG,CHKSTSK=ALL
CHKSTSK=CURRENT	As CSFE DEBUG,CHKSTSK=CURRENT
CHKSTRM=CURRENT	As CSFE DEBUG,CHKSTRM=CURRENT
CHKSTSK=NONE	As CSFE DEBUG,CHKSTSK=NONE. This override is the default.
CHKSTRM=NONE	As CSFE DEBUG,CHKSTRM=NONE. This override is the default.

Your strategy should be to have the minimum tracing that will capture the storage violation, to reduce the processing overhead and to give you less trace data to process. Even so, you are likely to get a large volume of trace data, so direct the tracing to the auxiliary trace data sets. For general guidance about using tracing in CICS problem determination, see Chapter 14, "Using traces in problem determination" on page 219.

You need to have only level-1 tracing selected, because no user code is executed between level-2 trace points. However, you do not know which calls to CICS components come before and after the offending code, so you need to trace all CICS components in AP domain. (These are the ones for which the trace point IDs have a domain index of "AP".) Set level-1 tracing to be special for all such

components, so that you get every AP level-1 trace point traced using special task tracing.

If the trailing storage check zone of a user-storage element has been overlaid, select special tracing for the corresponding transaction only. This is because it is very likely to be the one that has caused the overlay.

If the duplicate SAA of a TIOA has been overlaid, you need to select special tracing for all tasks associated with the corresponding terminal, because you are not sure which has overlaid the SAA. It is sufficient to select special tracing for the terminal and standard tracing for every transaction that runs there, because you get special task tracing with that combination. (See Figure 39 on page 232.)

Your choice of terminal tracing depends on where the transaction is likely to be initiated from. If it is only ever started from one terminal, select special tracing for that terminal alone. Otherwise, you need to select special tracing for every such terminal.

When you have set up the tracing options and started auxiliary tracing, you need to wait until the storage violation occurs.

What happens after CICS detects the storage violation?

When the storage violation is detected by the storage violation trap, storage checking is turned off, and an exception trace entry is made. If dumping has not been disabled, a CICS system dump is taken. The following message is sent to the console:

DFHSM0103 *applid* **STORAGE VIOLATION (CODE X**'*code*'**) HAS BEEN DETECTED BY THE STORAGE VIOLATION TRAP. TRAP IS NOW INACTIVE**.

The value of '*code*' is equal to the exception trace point ID, and it identifies the type of storage that was being checked when the error was detected. A description of the exception trace point ID, and the data it contains, is in the *CICS/ESA Diagnosis Reference* manual.

Format the system dump using the formatting keyword TR, to get the internal trace table. Locate the exception trace entry made when the storage violation was detected, near the end of the table. Now scan back through the table, and find the last old-style trace entry (AP 00xx). The code causing the storage violation was being executed between the time that the trace entry was made and the time that the exception trace entry was made.

If you have used the CHKSTSK=CURRENT option, can locate the occurrence of the storage violation only with reference to the last old-style trace entry for the current task. For greater accuracy in locating the occurrence of the violation, you might need to use the CHKSTSK=ALL option. However, keep in mind the additional overhead of running with CHKSTSK=ALL, and use it only if you need it.

You need to identify the section of code that was being executed between the two trace entries from the nature of the trace calls. You then need to study the logic of the code to find out how it caused the storage violation.

For suggestions on programming errors that might have caused your particular problem, look at the list of common ones given in “Programming errors that can cause storage violations” on page 197.

Storage violations that affect innocent transactions

Storage violations that affect innocent transactions—that is, transactions that do not cause the violation—usually go undetected by CICS. However, occasionally CICS detects that the initial SAA of a TIOA element or the storage check zone of a user-storage element has been overlaid by a task that does not own it.

If they are reproducible, storage violations of this type typically occur at specific offsets within structures. For example, the start of an overlay might always be at offset 30 from the start of a field.

The most likely cause of such a violation is a transaction writing data to a part of the DSAs that it does not own, or possibly FREEMAINing such an area. The transaction might previously have GETMAINed the area and then FREEMAINed it before writing the data, or addressability might otherwise not have been correctly maintained by an application. Another possible reason is that an ECB might have been posted by a transaction after the task that was waiting on it had been canceled.

Storage violations affecting innocent transactions are, in general, more difficult to resolve than those that are detected by CICS. Often, you become aware of them long after they occur, and then you need a long history of system activity to find out what caused them.

A strategy for storage violations affecting innocent transactions

The storage violation has been caused by a program writing to an area it does not own, but you probably have no idea at the outset which program is at fault. Look carefully at the content of the overlay before you do any other investigation, because it could help you to identify the transaction, program, or routine that caused the error. If it doesn't provide the clue you need, your strategy should be to use CICS tracing to collect a history of all the activities that reference the affected area.

The trace table must go back as far as task attach of the program causing the overlay, because that trace entry relates the transaction's identity to the unit of work number used on subsequent entries. This could mean that a very large trace table is needed. Internal trace is not suitable, because it wraps when it is full and it then overwrites important trace entries.

Auxiliary trace is a suitable destination for recording long periods of system activity, because it is possible to specify very large auxiliary trace data sets, and they don't wrap when they are full.

If you have no idea which transaction is causing the overlay, you need to trace the activities of every transaction. This impacts performance, because of the processing overhead.

A procedure for resolving storage violations affecting innocent transactions

Ensure that level-1 trace points are in the special set for all CICS components. Select special tracing for all user tasks, by setting up standard tracing for all user transactions and all terminals, and disable standard tracing by setting the master system trace flag off.

Use the CETR transaction to set up the tracing options, and select auxiliary trace as the trace destination. When you get the symptoms that tell you that the storage violation has occurred, take a system dump—unless the error causes a system dump to be taken.

Format the system dump, and format and print the auxiliary trace data set. If you know which area of storage the violation occurred in, you can use appropriate dump formatting keywords. Otherwise, you need to format the entire system dump. The dump formatting program may report that it has found some incorrect data. If not, you need to find the overlaid area by other means.

The next job is to locate all the entries in the trace table that address the overlaid area. Operations involving GETMAIN and FREEMAIN in particular are likely pointers to the cause of the error.

When you have found a likely trace entry, possibly showing a GETMAIN or FREEMAIN addressing the area, you need to find the ID of the associated transaction by locating the trace entry for TASK ATTACH. Rather than locating this manually, it is probably better to reformat the auxiliary trace data set selectively to show just trace entries corresponding to the task's unit of work.

Having found the identity of the transaction, take a look at all the programs belonging to the transaction. It is likely that one of these caused the overlay, and you need to consider the logic of each to see if it could have caused the error. This is a long job, but it is one of the few ways of resolving a storage violation affecting an innocent transaction.

What to do if you still can't find the cause of the overlay

If you are unable to identify the cause of the storage violation after carrying out the procedures of the preceding section, contact your IBM Support Center. They may suggest coding a global trap/trace exit to detect the storage violation.

Programming errors that can cause storage violations

The purpose of this section is to outline a number of commonly occurring programming errors that can cause storage violations.

1. Failing to GETMAIN sufficient storage. This is often caused by failure to recompile all the programs for a transaction after a common storage area has been redefined with a changed length.
2. Runaway subscript. Make sure that your tables can only grow to a finite size.
3. Writing data to an area after it has been FREEMAINed.

   When a task FREEMAINs an area that it has been addressing, it can no longer write data to the area without the risk of overwriting some other data that might subsequently be there.

4. Hand posting an ECB for a canceled task.

   If a task waiting on a CICS ECB is canceled, and then a transaction attempts to hand post the ECB when the resource being waited on becomes available, it may corrupt data belonging to some unrelated activity if the area once occupied by the ECB has been reused.

Storage recovery

The STGRCVY system initialization parameter enables you to vary the action taken by CICS on detection of a storage violation.

In normal operation, CICS sets up four task-lifetime storage subpools for each task. Each element in the subpool starts and ends with a check zone that includes the subpool name. At each FREEMAIN, and at end of task, CICS inspects the check zones and abends the task if either has been overwritten.

Terminal input-output areas (TIOAs) have similar check zones, each of which is set up with the same value. At each FREEMAIN of a TIOA, CICS inspects the check zones and abends the task if they are not identical.

If CICS is initialized with STGRCVY(YES), the overwriting of check zones is treated differently. After the system dump has been taken for the storage violation, CICS resets the check zones to their initial value and the task continues execution.

STGRCVY(NO) is the default.

Chapter 11. Dealing with XRF errors

If you have a problem when using XRF, it might not be easy to determine which system is at fault. In diagnosing these problems, the message content and sequence can be very useful. The CICS availability manager data sets hold these messages.

Symptoms of problems in an XRF complex

This section gives examples of the symptoms that can occur in your CICS system when using the extended recovery facility (XRF). For each symptom, a list of questions leads to the possible causes, indicating the nature of the error. If the error is a CICS error, rather than a user error, the module in error is shown.

The symptoms are divided as follows:

- **Active doesn't initialize**
- **Alternate doesn't initialize**
- **No backup sessions established**
- **Takeover starts, but doesn't complete**
- **Not all terminals recover after a takeover**
- **Unexpected takeover**
- **Takeover didn't occur when expected**
- **Related systems not taken over**
- **Alternate terminated unexpectedly**
- **Takeover took a long time**
- **Overseer didn't restart a job that had failed**
- **Unable to log on to CICS**.

Active doesn't initialize

1. Does MVS indicate that the active system is waiting for data sets?

   This could be because you have not specified DISP=SHR for data sets that need to be shared with the alternate system.

2. Have the CAVM data sets (the control and message data sets) been set up correctly?

3. For a two-CEC configuration, have the JES shared spool requirements been met?

4. Has the active system signed on to the CAVM?

5. If not, is another job signed on to the CAVM as an active system, or is takeover in progress?

6. If the active system has signed on to the CAVM, is an alternate system in the process of taking over and awaiting a response to a message?

7. If the IMS/VS Resource Lock Manager (IRLM) is used, is CICS waiting for the IRLM to be started?

   See the IMS/VS message DFS039I.

Alternate doesn't initialize

1. Does MVS indicate that the alternate system is waiting for data sets?

   This could be because you have not specified DISP=SHR for data sets that need to be shared with the active system.

2. Have the CAVM data sets (the control and message data sets) been set up correctly?

3. For a two-CEC configuration, have the JES shared spool requirements been met?

4. Is an alternate system already signed on to CAVM?

5. Have you started VTAM?

   Unlike the active system, the alternate system cannot initialize without VTAM.

No backup sessions established

1. Has the active system initialized successfully?

   See the appropriate section above.

2. Has the alternate system initialized successfully?

   See the appropriate section above.

3. Has the active system detected the alternate system signon to CAVM? This is indicated by the message DFHXG6403.

4. Was transaction CXCU attached?

   You can confirm this by message DFHXG6499: applid XRF CATCH-UP STARTED, which is issued by DFHZXCU as it starts.

5. Did the active system finish sending catch-up message DFHxxxx?

   Messages DFHXG6494, DFHXG6495, DFHXG6496, and DFHXG6497 are issued during normal execution of CXCU. Messages DFHXG6476 and DFHXG6492 are issued for an abnormal end of CXCU. Message DFHXG6498 indicates that CXCU has finished.

6. Did the alternate system fully receive catch-up messages DFHXGxxxx?

   Check for the following messages:

   - Message DFHTC1041 is issued at the start of tracking.
   - Message DFHTC1044 is issued when the start of catch-up is received.
   - Message DFHTC1040 is issued as records are received from the active system.
   - Message DFHTC1045 is issued when the end of catch-up is received.
   - Message DFHTC1024 is issued when an error occurs in tracking.

   If DBCTL is being used, DFHDX83xx messages are also issued for catch-up.

7. Do the terminals have backup sessions?

   You can use the VTAM command DISPLAY NET,ID=applid to display the status of active and backup sessions.

   CICS cannot control whether a terminal has, or has not, backup sessions. To qualify for a backup session (which means that it can support both an active session and a backup session, and that CICS can use the Session Control=Switch command to switch the session):

   - The APPLID definition must have HAVAIL=YES.
   - The terminal must be a remote SNA device.
   - The terminal must be connected to a 3725.
   - The NCP must be at a release that supports XRF.
   - The VTAM must be at a release that supports XRF.
   - The terminal must not use session-level cryptography.
   - The terminal must not be ISC (for example, LUTYPE6.1 or APPC).
   - The terminal must not use a negotiable bind (like an APPC single-session terminal, such as Scanmaster).

8. Are the necessary VTAM APPLID definitions and XDOMAIN definitions active?
9. Does the alternate system CSMT log indicate that XRF backup sessions are being established and then unbound?

   (See under “Not all terminals recovered after a takeover” on page 202.)

Takeover starts, but doesn’t complete

1. Has the takeover been accepted by the alternate system?
2. Is the active system still dumping?
3. Is the active system waiting because of a time-out during DBCTL disconnection processing?
4. Is the alternate system waiting for confirmation that the active system has ended (particularly in a two-CEC, CEC failure case)?

   If the alternate system has successfully issued the CANCEL, the active system may be unable to terminate abnormally; for example it may be necessary for the system operator to issue the FORCE command.

   If the active system has ended, the alternate system may be unable to determine from JES that it has. If the active system CEC has failed, JES may still indicate that the active job is running.

   In these cases, a reply is probably outstanding to message DFHXG6561 on the system operator console, or to message DFHXG6577 or DFHXG6578 if an error has been detected with the CLT contents. Refer to the *CICS/ESA Messages and Codes* manual for information about the responses to these messages.

5. If IRLM is used, is CICS waiting for the IRLM to be started?

6. Did the alternate system attempt to cancel the active system?

   When the alternate system issues the CANCEL command, this should be displayed on both system consoles in a two-CEC configuration. If MVS on the alternate system CEC did not accept the command, message DFHXG6560 is displayed by the alternate system.

   If there is a failure of JES communication between the active and alternate CECs, the CANCEL command may not have been routed.

   If there are errors in the CLT, the alternate system may not be able to construct the appropriate CANCEL command.

   In each of these cases a reply to messages DFHXG6561, DFHXG6577 or DFHXG6578 may be required. Refer to the *CICS/ESA Messages and Codes* manual for information about the responses to these messages.

7. Do you have the correct definitions in the CLT?

   A previous message should be displayed on the alternate system CEC to report the error with the CLT contents. If CLT errors were detected, the alternate system may not be able to construct the CANCEL command.

   A reply to messages DFHXG6561, DFHXG6577 or DFHXG6578 may be required. Refer to the *CICS/ESA Messages and Codes* manual for information about the responses to these messages.

Not all terminals recovered after a takeover

Terminals with backup sessions

1. Was the terminal definition successfully restored in the alternate system?

   On the alternate system, either of these messages might be issued if a tracking record is dropped for some reason:

   DFHTC1022 *applid* **ERROR FOR XRF TRACKING RECORD - TYPE:** *type* **- KEY:** *key*

   DFHTC1046I *applid* **FLUSHING TERMINAL CONTROL TRACKING**

2. Was a backup bind established for this terminal?

   The following message is issued to the transient data destination CSNE for each terminal that has a backup bind established:

   DFHZC6593I *date time applid termid tranid* **Node** *netname* **backup session started**. *sense ((instance)* **Module name: {DFHZOPX}***)*

   Note that subsequently the following message might have been issued to the same destination:

   DFHZC3462I *date time applid termid tranid* **Node** *netname* **session terminated.** *sense ((instance)* **Module name: {DFHZCLS}***)*

   This means that there was a backup session that was terminated for some reason.

   You could also consult the CSMT log for other messages, for example, network services error.

3. Was the conversation successfully restarted for this terminal?

   This message is issued for each terminal that has been switched to the alternate system:

   DFHZC6590 *tttt*, *pppp*, *time* **NODE** *netname* **CONVERSATION RESTARTED**

   The message is not issued at the time of the switch, but from DFHZNAC when that terminal begins its recovery process.

   You could also consult the CSNE log.

4. Is there any indication in the CSMT log that recovery failed and that this terminal was unbound during takeover?

   This depends upon:

   - The terminal type
   - Its session characteristics as presented in the bind
   - How it is defined in its recovery action parameter
   - The activity on the session at the moment the switch was issued.

   Depending upon the above points, it is possible that the switch command was issued from the active system to transfer the terminal session to the alternate system, but then the alternate system issued CLSDST (UNBIND) to release that session. If that happened, the alternate system attempts to reacquire the session if the terminal supports SIMLOGON.

5. Are the necessary VTAM XDOMAIN definitions active/available?

6. Is RECOVACT=NONE specified for this terminal?

   If RECOVACT=NONE is specified, the logon/logoff state of the terminal in the active system is not tracked by the alternate system, and no backup session is established.

7. At takeover, was the following message issued?

   DFHTC1042I *applid* **WAITING FOR TERMINAL CONTROL TRACKING TO DRAIN**

   This message indicates that there is a backlog of messages in the CAVM message file. It's likely that the information about the terminal that failed to recover was not known to the alternate system.

8. At takeover, did the following message occur?

   DFHTC1046I *applid* **FLUSHING TERMINAL CONTROL TRACKING**

   Some event concerning the establishment of the backup bind was held up, and this terminal is not switched. It is treated as a terminal without a backout session, and the alternate system attempts to reacquire its session.

9. Did the CSNE log indicate that conversation was restarted for one terminal, but not for the rest?

Remember, too, that TCAM terminals are not switched to the alternate system on takeover.

Terminals without backup sessions

1. In a two-CEC configuration, do the control units need switching?
2. Are the necessary VTAM XDOMAIN definitions active and available?
3. Is RECOVACT=NONE specified for this terminal?

   If it is, the logon/logoff state of the terminal in the active system won't have been tracked by the alternate system. The recovery at takeover is the same as that at normal emergency restart; the alternate system honors the AUTOCONNECT option for the terminal.
4. Should the reconnect transaction (CXRE) have run yet?

   If CXRE could not be scheduled, messages DFHXG6485, DFHXG6487, or DFHXG6488 are issued. If CXRE was delayed, to allow physical switching of terminals to be completed, message DFHXG6481 is issued, telling you how long the delay will be. This delay includes the delay specified in the system initialization parameter AUTCONN.
5. Did the reconnect transaction (CXRE) start up on time?

   Message DFHXG6490 is issued at the start of every reconnection pass. When CXRE has reconnected all the terminals that it believes it is supposed to reconnect, it ends the pass with message DFHXG6484. Otherwise, if it is unable to reschedule itself, it issues the message DFHXG6489.

Unexpected takeover

1. Did someone stop the CEC of the active system, with COMMAND=AUTO coded for the alternate system?
2. Does the CICS system have a sufficiently good performance group to allow continuous normal running?
3. Is the ADI value specified in the system initialization table for the alternate system too low?
4. In a two-CEC system, has there been some unexpected processing that stopped the active system from running?

   Note: If such processing is expected, you can suspend surveillance by using the command CEBT SET SURV OFF, or change the alternate system to TAKEOVER=COMMAND by using CEBT SET TAKE COM for the duration of the activity.
5. Did the global user exit XXDFB request a takeover?

Takeover didn't occur when expected

1. Was the correct TAKEOVR value in effect for the alternate system?
2. If the CEBT PERFORM TAKEOVER command was entered, was it accepted?
3. If you are expecting an automatic takeover, is the ADI value set too high?
4. If you have set up for an automatic takeover, was the missing surveillance signal detected by the alternate system?
5. Did the active system sign off abnormally?

   If so, did the alternate system detect this?

6. Did the alternate system have a large backlog of messages in the CAVM message file?

   If the backlog becomes too large, the alternate system becomes unusable. Do you need a larger message file?

7. Is the alternate system clock significantly behind the active system clock? This can cause the alternate system to wait before completing the takeover. See the messages DFHXG6682 and DFHXG6683.

Related systems not taken over

1. Did the master or coordinator alternate system run its CLT successfully?

   If errors were detected in the CLT, message DFHXG6576 should have been issued by the alternate system. Refer to the *CICS/ESA Messages and Codes* manual for an explanation of this message.

2. Do you have the correct definitions in the CLT, so that the other alternate systems are instructed to take over?

   If other alternate systems of related systems should have initiated a takeover as a result of a command entry in the CLT, but did not, the appropriate command entries may not be correctly specified in the CLT, or the set of commands may be incomplete.

   System operator commands issued as part of CLT processing are normally displayed on the alternate system CEC system console. If an error was detected in a command, there should be an associated MVS error message displayed on the alternate system CEC system console. CLT commands are not interpreted by CICS.

   The CLT contents may not be correct for use by this alternate system. For example, the appropriate APPLID may not be correctly specified in the CLT, in which case message DFHXG6567 should be displayed on the system console reporting the error.

Alternate terminated unexpectedly

1. Did the active system shut down normally, thus causing the alternate system to shut down normally?

2. Was the alternate system shutdown caused by a CAVM event?

   This could be because a new active system has been initialized, thus invalidating the existing alternate system.

3. Did the alternate system's VTAM fail, thus causing the alternate system to terminate?

   The alternate system does not complete initialization until VTAM is available and the VTAM ACB has been opened.

   Similarly, if the VTAM in the alternate system fails while the alternate system is tracking the active system, the alternate system cannot continue, and it is abended. Bring up VTAM again, and restart the alternate system.

4. Did the alternate system become unusable because of an unacceptable message backlog in the CAVM message file?

Takeover took a long time

1. Did the active system take a long time to close down after it was canceled?
2. Did either of these messages appear?

   DFHXG6480I *applid* **WAITING FOR BACKUP SIMLOGON PROCESSING TO DRAIN**

   DFHXG6475E *applid iiii* **BACKUP SIMLOGON(S) ABANDONED**
3. Did either of these messages appear?

   DFHTC1042I *applid* **WAITING FOR TERMINAL CONTROL TRACKING TO DRAIN**

   DFHTC1046I *applid* **FLUSHING TERMINAL CONTROL TRACKING**

For an explanation of the above messages, and advice about how to respond to them, refer to the *CICS/ESA Messages and Codes* manual.

Overseer didn't restart a job that had failed

1. Is restart in place disabled?
2. Is restart in place not active for this system?
3. Is the job to be restarted not defined to the overseer?

   If the overseer determines that a takeover is taking place, or is likely to take place, it does not perform a restart in place.
4. Was the overseer unable to open or read the CAVM data sets associated with this job?

Unable to log on to CICS

1. Is the VTAM USERVAR set to the correct value?

   The VTAM commands D NET,USERVAR, and F procname,USERVAR, can be used to inspect, and change, the setting of the USERVAR whose ID is the APPLID used in the failing logon attempts. The VTAM USERVAR is the CICS generic APPLID.
2. Has CICS issued the necessary MODIFY procname,USERVAR command?

   The MODIFY command is logged by both MVS and VTAM. CICS issues the following message if it is unable to issue the MODIFY command:

   DFHXG6479I MODIFY USERVAR ISSUED UNSUCCESSFULLY. RETURN CODE *nn*
3. Has the new USERVAR value been propagated around the network after initialization of the active system, or after a takeover?

   The VTAM commands D NET,USERVAR, and F procname,USERVAR, can be used to inspect and change the setting of the USERVAR.
4. Are the necessary VTAM XDOMAIN definitions not active, or not available?
5. Did you set up the specific and generic APPLIDs correctly in your system initialization parameters? Are they the right way round?

Debugging the overseer sample program

If the overseer detects an error, it may put out a diagnostic message, or take a snap dump, or do both. You can request a snap dump of the overseer's address space using the SNAP command.

The documentation for messages put out by the overseer is in the prolog of the source for the sample program, DFH$AXRO.

Return and reason codes for some types of failure are saved in the RADBS control block, and within individual GENDS control blocks if the error is connected with a particular generic pair of CICS systems.

Notes:

1. Some overseer problems are asynchronous, and post a user ECB when they complete. Be careful not to reuse user areas before the events complete.
2. There is one system key area, and one user key area, managed by the overseer authorized services. You can find the assembler-language DSECTS, and a description of the logic used, in module DFHWOSB.

Chapter 12. External CICS interface

The following CICS messages support the external CICS interface:

DFHIR3799

DFHEX0001 DFHEX0011
DFHEX0002 DFHEX0012
DFHEX0003 DFHEX0013
DFHEX0004 DFHEX0014
DFHEX0005 DFHEX0015
DFHEX0010 DFHEX0016

Messages DFH5502W and DFH5503E include support for the external CICS interface facility.

This facility is also supported by two translator messages, DFH7004I and DFH7005I.

For full details of all CICS messages, see the *CICS/ESA Messages and Codes*.

The external CICS interface outputs trace to two destinations: an internal trace table and an external MVS GTF data set. The internal trace table resides in the non-CICS MVS batch region. Trace data is formatted and included in any dumps produced by the external CICS interface.

Trace entries are issued by the external trace interface destined for the internal trace table and/or an MVS GTF data set. They are listed in the *CICS/ESA Diagnosis Reference* manual.

The external CICS interface produces MVS SYSM dumps for some error conditions and MVS SDUMPs for other, more serious conditions. These dumps contain all the external CICS interface control blocks, as well as trace entries. You can use IPCS to format these dumps.

A user-replaceable program, DFHXCTRA, is available for use under the guidance of IBM service personnel. It is the equivalent of DFHTRAP used in CICS. It is invoked every time the external CICS interface writes a trace entry. The actions of the CICS-supplied DFHXCTRA are, on a pipe FREEMAIN error, to:

1. Make a trace entry
2. Take an SDUMP
3. Skip writing the current trace entry
4. Disable itself.

Chapter 13. Dealing with MRO and shared database problems

This chapter discusses problems with interregion communicaton (IRC) under two main headings:

- MRO problems
- Shared database problems

MRO problems

Suppose an error is suspected in communication between System A and System B. The problem can be determined by looking at either system. The following procedure applies to System A, but could equally well apply to System B. The first step is to look at the field CSACRBA in the CSA optional features list in System A. If CSACRBA is zero, the interregion communication routine is not active in that system.

If CSACRBA is not zero, examine the TCTSEs in System A and find the TCTSE for System B. In this TCTSE, TCSEIRCF is the flag byte that indicates the state of communication between the two systems. If bit TCSEIRNC in this byte is on, there is no communication between System A and System B. This is either because System B has not started interregion services, or because System A is out of service with respect to System B, or because System B is out of service with respect to System A.

If bit TCSEIRNC is off, a session should exist. The next step is to inspect the primary and secondary session(s) between the systems. The first primary session is pointed to by field TCSEVC1 in the TCTSE, and the first secondary session is pointed to by TCSEVC2. If System A initiated the session, look at secondary sessions, otherwise look at primary sessions.

Each session is defined by a TCTTE. The field TCTESCCB in TCTTE is zero if the session is not connected to the other system, otherwise it contains the address of the subsystem connection control block (SCCB) that the interregion SVC routine uses to represent that end of the connection.

Assuming the session is connected, TCTTECA is zero if no task is using the session. Otherwise, TCTTECA points to the TCA of the task that uses the session.

The protocol for interregion SVC transfer is similar to that for VTAM SNA data flow control. Field TCTEIRF1 contains information on the state of the session, field TCTESBRS gives the bracket status, field TCTESRHI is the inbound request header, and field TCTESRHO is the outbound request header.

The field TCTENIBA points to the TCTTE extension for the NIB descriptor. Within this TCTTE extension, TCTEPSQ contains the primary name, and TCTESSQ contains the secondary name. Thus a session in System A can be related to a session in System B.

Shared database problems

The *CICS/ESA Diagnosis Reference* manual describes how the interregion communication (IRC) facility enables IMS/VS batch programs to share an IMS/VS DL/I database with CICS application programs. Briefly, each DL/I call from an IMS/VS batch program is handled by batch region controller modules that invoke an interregion SVC routine to pass the request across to the CICS address space. In CICS, a mirror task handles the request and invokes the same SVC routine to return the response to the batch region. The batch region controller modules handle the response and pass it on to the batch program. The interface between IMS/VS batch programs and IMS/VS DL/I databases is shown in Figure 28.

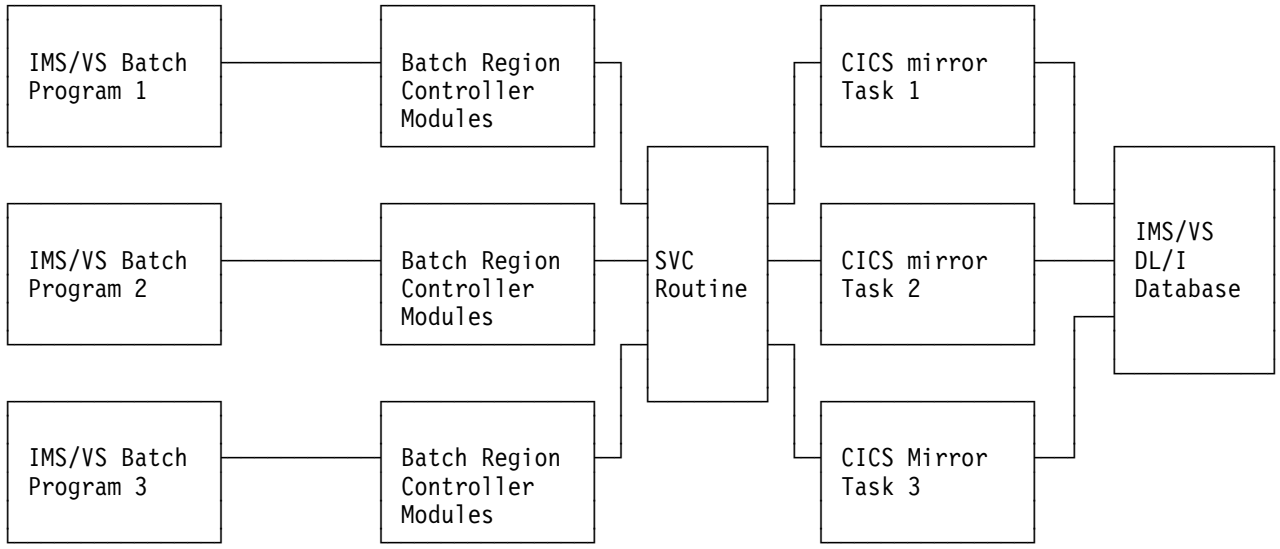


Figure 28. Interfaces between IMS/VS batch programs and IMS/VS DL/I database

This remainder of this chapter outlines suggested approaches to the investigation of problems in a shared database batch region, and in the CICS address space that is sharing database access. All references to “batch region” should be interpreted as “shared database batch region”.

Investigation of problems in a shared database batch region

This section describes how to obtain the following information from a batch region dump:

- The contents of the application program’s registers at the time of abnormal termination or at the time of the last-issued DL/I call
- Information relating to messages caused by unsuccessful invocation of the SVC routine
- Input buffer contents.

IRC information in the batch region dump is contained in the **batch region work area**, which is identified by the following character string:

```
'*****DFHDRWA  -  WORK AREA'
```

The entire area is described in the DSECT named DFHDRWA (copy book DFHDRCA). Figure 29 shows the main diagnostic areas in DFHDRWA.

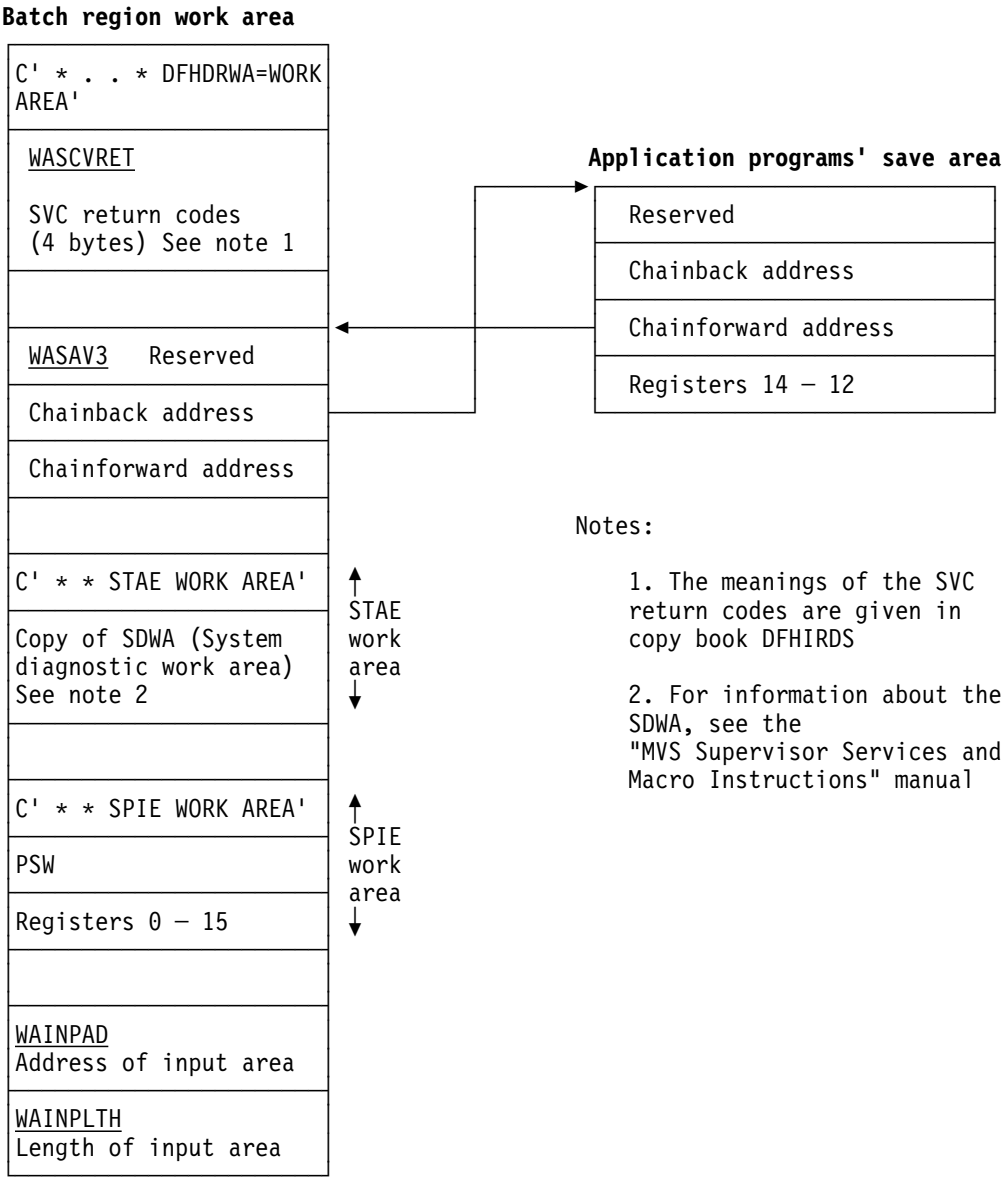


Figure 29. Diagnostic areas in shared database batch region dump

Dealing with a program check (message DFHIR3710)

To determine whether the program check occurred in the application program code or in the batch region controller code, inspect the PSW in the SPIE work area. The SPIE work area is identified by the characters C'**SPIE WORK AREA' (see Figure 29).

If the PSW shows that the program check occurred in the application program code, the contents of the application's registers at that time can be found from the SPIE work area.

If the program check occurred in the controller code (DFHDRP...), the application's registers at the time of the most recent DL/I call can be found from the application's save area (addressed by WASAV3 + X'04'). In particular, register 14 contains the address from which the most recent DL/I call was made.

Dealing with an abnormal termination (message DFHIR3701)

To determine whether the abend occurred in the application program code or in the batch region controller code, inspect the PSW in the STAE work area. The STAE work area is identified by the characters C'**STAE WORK AREA'. (Immediately after these characters is the system diagnostic work area (SDWA) as provided by MVS/XA on entry to the STAE exit routine. In the event of MVS/XA failing to provide an SDWA, work area (SDWA) as provided by MVS on entry to the STAE exit routine. In the event of MVS failing to provide an SDWA, the WANSDWA bit in WAFLG1 of DFHDRWA is set to 1. For information about the SDWA, see the *MVS Supervisor Services and Macro Instructions* manual.)

If the PSW shows that the abend occurred in the application program code, the contents of the application's registers at that time can be found from the STAE work area.

If the abend occurred in the controller code (DFHDRP...), the application's registers at the time of the most recent DL/I call can be found from the application's save area (addressed by WASAV3 + X'04'). In particular, register 14 contains the address from which the most recent DL/I call was made.

Dealing with failed SVC calls (messages DFHIR3706, DFHIR3709, and DFHIR3713)

Each of the messages DFHIR3706, DFHIR3709, and DFHIR3713 is the result of an unsuccessful invocation of the interregion SVC routine. The 4-byte return code returned by the SVC is stored in WASVCRET. The meanings of the SVC return codes are described in the interregion communication subsystem block copybook DFHIRSDS.

Reading contents of application's IRC input buffer

If the problem is suspected to be the result of incorrect data received from CICS, the contents of the application's IRC input buffer can be inspected. In the batch region work area, WAINPAD contains the address and WAINPLTH contains the length of the input area.

Online method of finding mirror task identification

Use the CEMT INQ IRBATCH request. The response to this request lists the task identifications of the mirror transactions for all batch jobs that are currently in communication with CICS.

If the faulty batch region and its mirror transaction have already terminated, they will not appear in the inquiry list. In this event, follow the procedure described below under "Using trace table when mirror task identification is not known" on page 215.

Offline method of finding mirror task identification

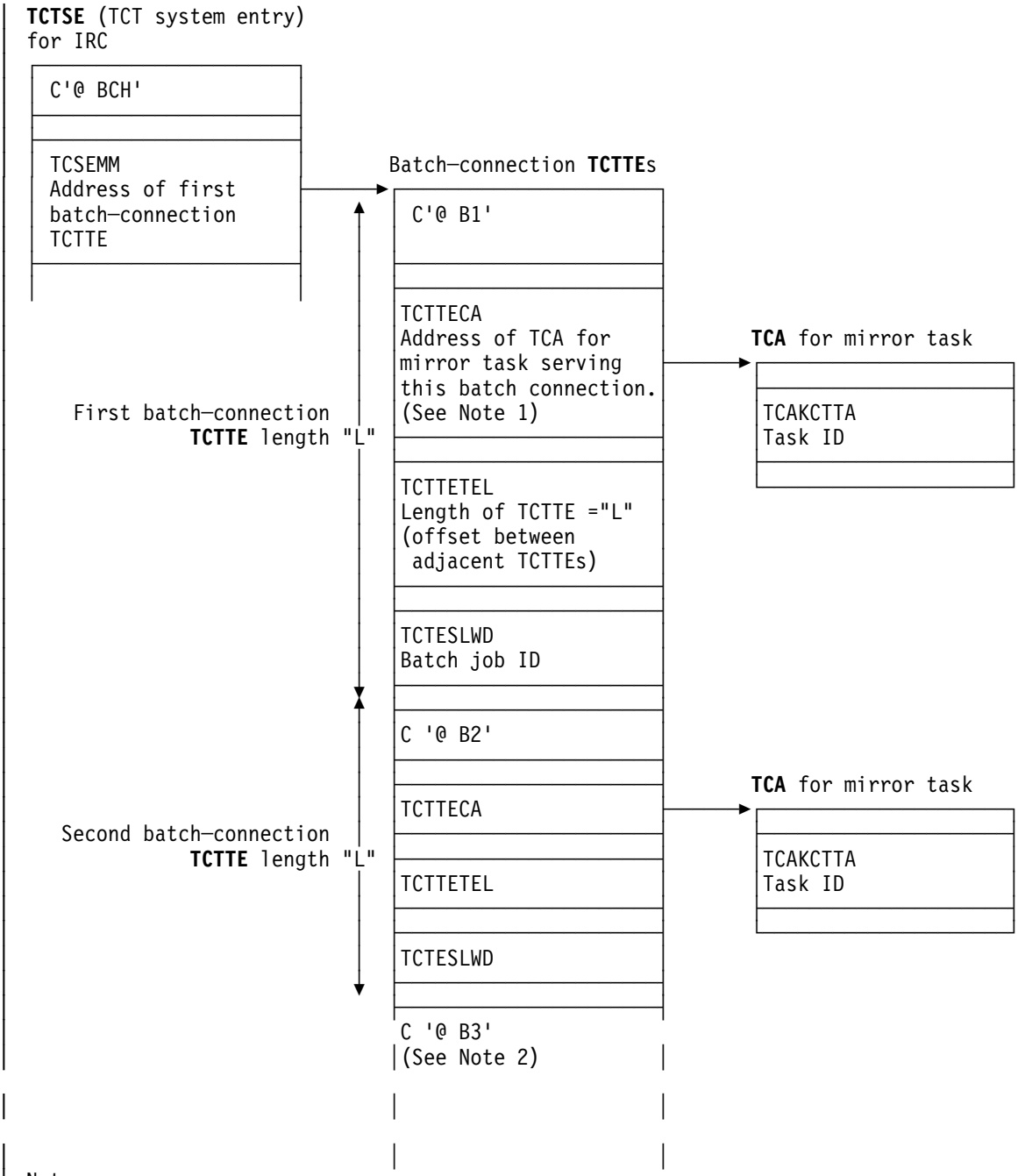
Use the CICS dump as follows:

1. Find the TCT system entry that is identified with the characters C'=@BCH'; see Figure 30 on page 216.
2. From the address in the field TCSEMM, locate the first batch-connection TCTTE, which is identified with the characters C'@B1'. (Subsequent batch-connection TCTTEs are chained together, using the field TCTENEXT as a chain address, and are identified with the characters C'@B2', C'@B3', and so on. All TCTTEs have the same length, which is given in field TCTTETEL.)
3. In the batch-connection TCTTE whose TCTESLWD field contains the identification of the faulty batch job, use the TCTTECA field to locate the TCA for the mirror task. (If no TCTTE contains the required batch job identification, or if the TCTTECA field contains all zeros, the mirror task was not active at the time of the dump. In this event, follow the procedure described below under "Using trace table when mirror task identification is not known.")
4. Assuming the mirror task TCA has been located, obtain the mirror task identification from the TCAKCTTA field.

Using trace table when mirror task identification is not known

Analyze subsequent trace entries that have the same task identification.

Also, DFHCRNP makes entries in the trace table that contains the batch LUWID.



Notes:

1. When a batch-connection TCTTE is not currently allocated to a batch region job (that is, no mirror task is currently allocated to this TCTTE), field TCTTECA has a value of zero.
2. The number of batch-connection TCTTEs is as specified by SESNUMB in DFHTCT=IRCBCH.

Figure 30. TCTTEs for batch job connections

Part 3. Using traces and dumps in problem determination

Part 3 contains:

Chapter 14. Using traces in problem determination

Several types of traces are available to record different aspects of CICS activities. Some of them you have control over, and some you don't. With those you can control, it is important to select just the right amount of tracing activity to give you the evidence you need. If you select too little tracing, you won't get enough evidence. If you select too much, you might find it difficult to identify the significant evidence.

The types of tracing that can be used for CICS systems are:

- "Normal" CICS tracing, is done by the trace domain at predetermined trace points in CICS code during the regular flow of control, and "user" tracing from applications. You get this when you turn on CICS internal tracing, auxiliary tracing, and GTF tracing. You control this type of tracing to suit your needs, except that, when an exception condition is detected by CICS, it always makes an exception trace entry. You cannot turn exception tracing off.
- Tracing from the CICS exit programming interface (XPI), using the TRACE_PUT XPI call from an exit program. You can control this within the exit program, or by enabling and disabling exits.
- CICS XRF tracing, which records CICS XRF-related activities. This is always running if you are operating in a CICS XRF environment.
- Program check and abend tracing, which is used by CICS to record pertinent information when a program check or abend occurs. This is controlled by CICS code.
- CICS VTAM exit tracing. The exits are driven by VTAM when it reaches a particular stage in its asynchronous processing, but the trace points are in CICS code. You can turn CICS VTAM exit tracing on or off.
- VTAM buffer tracing. This is a part of VTAM, but it can be used to record the flow of data between logical units in the CICS environment. You can control this type of tracing to meet your needs.

This chapter starts with a discussion of formatting and interpreting trace entries. The characteristics of the various types of tracing are described in the sections that follow:

"Normal CICS tracing" on page 227.
"CICS exception tracing" on page 242.
"CICS XRF tracing" on page 243.
"Program check and abend tracing" on page 246.
"CICS VTAM exit tracing" on page 246.
"VTAM buffer tracing" on page 247.

Formatting and interpreting trace entries

Before you can look at the trace entries that have been sent to the various trace destinations, you need to do some formatting. The way you do the formatting varies depending on the destination. For more details of trace utility programs, see the *CICS/ESA Operations and Utilities Guide*.

You can specify **abbreviated** or **extended** trace formatting, to give you either a "one line per entry" trace table, or a "many lines per entry" trace table. The structures of the two types of trace entry are described in the sections that follow.

Most of the time, the abbreviated trace table is the most useful form of trace formatting, as you can quickly scan many trace entries to locate areas of interest. However, you might occasionally want to look at the extended format trace entries, to understand fully the information given in the corresponding abbreviated entries, and to be aware of the additional data supplied with many extended trace entries.

The internal trace table can be formatted in one of two ways:

1. From a CICS system dump, using the CICS print dump exit
2. From a transaction dump, using the CICS dump utility program, DFHDU410.

Auxiliary trace can be formatted using the CICS trace utility program, DFHTU410. You can control the formatting, and you can select trace entries on the basis of task, terminal, transaction, time frame, trace point ID (single or range), dispatcher task reference, and task-owning domain. This complements the usefulness of auxiliary trace for capturing large amounts of trace data.

Note: Trace entries can only be formatted selectively by transaction or terminal if the "transaction attach" entry (point ID AP F004, KC level-1) for the transaction is included in the trace data set.

GTF trace can be formatted with the same sort of selectivity as auxiliary trace, using a CICS-supplied routine with the MVS interactive problem control system (IPCS).

Interpreting extended-format CICS system trace entries

CICS system trace entries made to the internal trace table, the auxiliary trace data sets, and the GTF trace data set can all be formatted to give the same type of information.

There are two slightly different extended trace entry formats. One ("old-style") resembles the format used in earlier releases of CICS, and gives FIELD A and FIELD B values. The other ("new-style") uses a completely new format, described below.

Both types of formatted trace entries always show:

- The **trace point ID**. This is an identifier that indicates where the trace point is in CICS code. In the case of application (AP) domain, the request type field included in the entry is also needed to uniquely identify the trace point. For all other domains, each trace point has a unique trace point ID.

  Its format is always a two-character domain index, showing which domain the trace point is in, then a space, then a four-digit (two-byte) hexadecimal number identifying the trace point within the domain.

  The following are examples of trace point IDs:

```
AP 00EE         trace point X'00EE' in Application Domain
DS 0005         trace point X'0005' in Dispatcher Domain
TI 0101         trace point X'0101' in Timer Domain
```

- An **interpretation string**, showing:
  - The **module** where the trace point is located
  - The **function** being performed
  - Any **parameters** passed on a call, and any response from a called routine.
- A **standard information string**, showing:
  - The **task number**, which is used to identify a task uniquely for as long as it is in the system. It provides a simple way of locating trace entries associated with specific tasks, as follows:
    - A five-digit decimal number shows that this is a trace entry for a task with a TCA, the value being taken from field TCAKCTTA of the TCA.
    - A three-character non-numeric value in this field shows that the trace entry is for a system task. You could, for example, see “III” (initialization), or “TCP” (terminal control).
    - A two-character domain index in this field shows that the trace entry is for a task without a TCA. The index identifies the domain that attached the task.
  - The **kernel task number** (KE_NUM), which is the number used by the kernel domain to identify the task. The same KE_NUM value for the task is shown in the kernel task summary in the formatted system dump.
  - The **time** when the trace entry was made. (Note that the GTF trace time is GMT time.)
  - The **interval** that elapsed between this and the previous trace entry, in seconds.

  The standard information string gives two other pieces of useful information:
  - The address of the **MVS TCB** (field TCB) that is in use for this task. This field can help you in comparing a CICS trace with the corresponding MVS trace.
  - The **return address** (field RET), passed in Register 14 to a called routine. This field helps by showing what invoked the module that is making this trace entry.
- A number of **data fields** containing information relevant to the function being performed.

  For old-style trace points, these are shown as fixed length (4-byte) FIELD A and FIELD B values in the same line as the interpretation string. Both the hexadecimal data values and any printable EBCDIC characters that they represent are shown.

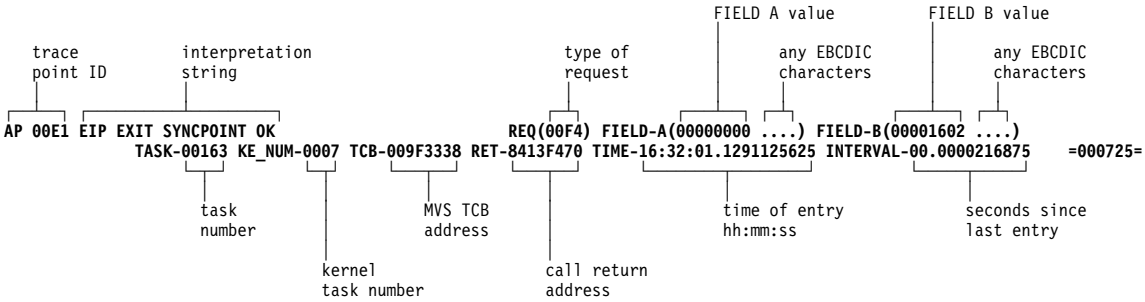
  For new-style trace points, 1–7 variable-length data fields can be given. They are shown immediately below the standard information line. Any printable EBCDIC characters represented by byte values in the data fields are shown on the right of the dump.

  Some of the data fields in the new trace entries contain material intended for use by IBM support personnel and you cannot interpret them directly. However, there is enough information to resolve user errors and for IBM support personnel to resolve most system errors in the interpretation string for the entry. Trace entries from old-style trace points also include a **request type**

field (REQ), which gives the same information as the two-byte request field in the formatted trace entries of CICS/MVS Version 2.

Some old-style trace entries also have a RESOURCE field. When provided, it is usually the name of a resource associated with the request being traced. For example, for program control requests, it is the program name.

Figure 31 shows a trace entry made from an old-style trace point. Its trace point ID is AP 00E1, corresponding to trace ID X'E1' of CICS/MVS Version 2.



Note: For some trace entries, an 8 character resource field appears to the right of FIELD B.

Figure 31. Example of the extended format for an old-style trace entry

If you are migrating from CICS/MVS Version 2, you can probably see the resemblance between this type of format and the one it replaces.

The following is an explanation of the old-style trace entry:

- AP 00E1 shows that this trace entry was made from trace point X'00E1' in the application domain.

  Note that although all old-style trace points are in AP domain, not all AP domain trace points are old-style. Some are new trace points, and they have a similar format to that shown in Figure 32 on page 223. In general, old-style trace points have values less than or equal to X'00FF' and new AP-domain trace points have values greater than or equal to X'0200'

- The interpretation string ‘EIP EXIT SYNCPOINT OK’ gives information about what was going on at the time the trace entry was made.
  - EIP identifies the module where the trace point is located, in this case DFHEIP.
  - EXIT shows that the trace entry was written on completion of processing a request.
  - SYNCPOINT shows the type of function requested.
- REQ(00F4) represents the “request type” of the trace format of CICS/MVS Version 2. In this example, byte 1 bits 0–3 (X'F') show that the trace entry is made on exit from the request.
- FIELD-A and FIELD-B contain the same data as FIELD A and FIELD B in the old-style format.

  FIELD-A bytes 0–3 would contain the secondary response, EIBRESP2. FIELD-B bytes 0–1 would contain the condition number, EIBRESP. In this example, both are zero, indicating that no error response has been returned. FIELD-B bytes 2–3 contain the command code, EIBFN. In this example, this is X'1602', showing that the EXEC CICS command was SYNCPOINT.

- The standard information string shows:
  - The task number for the currently running task is 00163. Any trace entries having the same task number would have been made while this task was running.
  - The kernel task number for the task is 0007. If you were to take a system dump while this task was in the system, you could identify the task in the kernel summary information from this number.
  - The time when the trace entry was made was 16:32:01.1291125625.
  - The interval that elapsed between this and the preceding trace entry was 00.0000216875 seconds.

Look now at Figure 32, which shows a new-style trace entry.

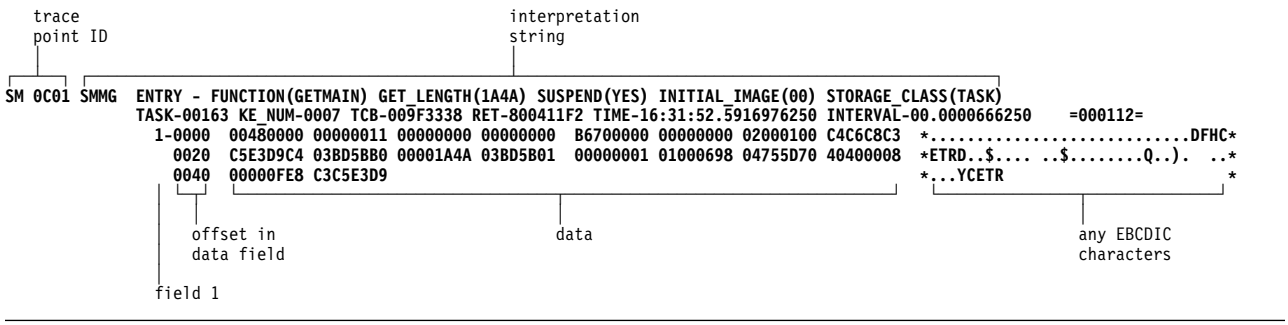


Figure 32. Example of the extended format for a new-style trace entry

The following is an explanation of the new-style trace entry:

- SM 0C01 shows that this trace entry was made from trace point X'0C01' in the storage manager domain.
- The interpretation string provides the following information:
  - SMMG tells you the trace call was made from within module DFHSMMG.
  - ENTRY FUNCTION(GETMAIN) tells you the call was made on entry to the GETMAIN function.
  - GET_LENGTH(1A4A) SUSPEND(YES) INITIAL_IMAGE(00) STORAGE_CLASS(TASK) tells you the parameters associated with the GETMAIN call, as follows:
    - The request is for X'1A4A' bytes of storage.
    - The task is to be suspended if the storage isn't immediately available.
    - The storage is to be initialized to X'00'
    - The storage class is TASK.
- The standard information string gives you this information:
  - The task currently running is task number X'00163'.
  - The kernel task number for the task is 0007.
  - The time when the trace entry was made was 16:31:52.5916976250.
  - The time that elapsed between this and the preceding trace entry was 00.0000666250 seconds.
- The data that is displayed below the standard information was taken from only one data area.

The *CICS/ESA User's Handbook* contains details of trace point ID SM 0C01. The data area is the SMMG parameter list.

Relevant information is formatted from the data area and appears in the trace entry interpretation string.

Note that information about some data areas is intended for use by IBM support personnel, and this means that details of their format and contents might not be available to you. If you reach a point at which you are certain that you cannot continue the problem determination process because you do not have access to information about a data area, you need to contact your IBM Support Center. They may not necessarily tell you the format and contents, but they investigate the problem in the usual way, as described in Chapter 17, "IBM program support" on page 287.

Interpreting abbreviated-format CICS system trace entries

Abbreviated-format CICS trace entries contain much of the information present in the corresponding extended-format trace entries, and they are often sufficient for debugging purposes. There is a one-to-one correspondence between the trace entry numbers for the abbreviated and extended trace entries, so you can easily identify the trace entry pairs.

Abbreviated trace entries show a "TCB index" instead of an MVS TCB address. The TCB index has a value of 1, 2, 3, 4, or 5 and it indicates when CICS task execution switches from one MVS TCB to another.

The first abbreviated trace entry always has a value of 1 for the TCB index. When execution switches to another TCB, the value is shown as 2. When execution switches again, the value shown is either 1 if the previous TCB is used, or 3, 4, or 5 if either of the remaining TCBs is used.

A value of 3, 4, or 5 for the TCB index appears only when certain optional functions are used. These optional functions are concurrent mode (SUBTSKS=1 in the system initialization table or start-up overrides), FEPI, or ONC RPC.

There is no absolute correlation between the type of MVS TCB (quasi-reentrant, resource owning, concurrent, FEPI, or ONC RPC) and the value of the TCB index. A value of 1 is always shown for the TCB that happens to be in use when the first trace entry is made.

Figure 33 gives an example of the abbreviated format for an old-style trace entry.

Note: For some trace entries, an 8-character resource field appears to the right of FIELD B.

Figure 33. Example of the abbreviated format for an old-style trace entry

Abbreviated old-style trace entries are easy to interpret, as you can readily identify the REQ, FIELD A and FIELD B fields. Note that some such trace entries also include a RESOURCE field.

For ZCP trace entries, FIELD B (which contains the TCTTE address) is printed twice on each line. This allows both sides of the output to be scanned for the terminal entries on an 80-column screen without having to scroll left and right.

Figure 34 gives an example of the abbreviated format for a new-style trace entry.

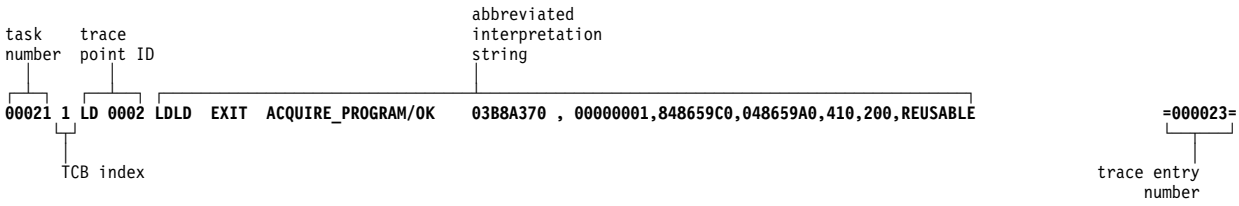


Figure 34. Example of the abbreviated format for a new-style trace entry

Abbreviated-format new-style trace entries are less readily interpreted, because the parameters in the interpretation string are not identified by name. If you are not familiar with the parameters included in the trace entry, you need to look at the corresponding extended-format trace entry to find out what they are. Figure 35 shows the corresponding extended-format trace entry.

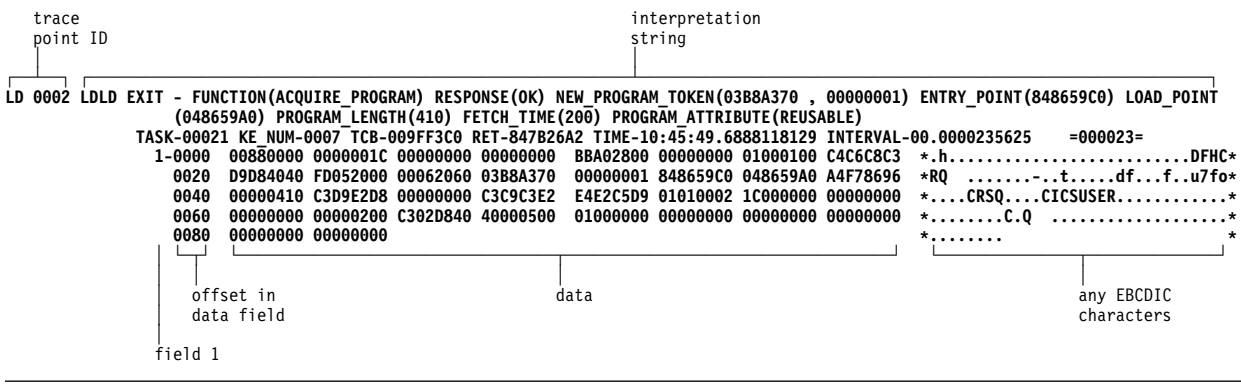


Figure 35. Example of the corresponding extended-format trace entry

LD 0002 shows that this trace entry was made from trace point X'0002' in the loader domain.

The interpretation string provides this information:

- LDLD tells you the trace call was made from within module DFHLDLD.
- EXIT FUNCTION(ACQUIRE_PROGRAM) tells you the call was made on exit from the ACQUIRE_PROGRAM function

The standard information string gives you this information:

- The task currently running has a task number of 00021.
- The kernel task number for the task is 0007.
- The time when the trace entry was made was 10:45:49.6888118129. (Note that the GTF trace time is GMT time.)
- The time that elapsed between this and the preceding trace entry was 00.0000235625 seconds.

The data displayed below the standard information was taken from only one data area. If you look in the *CICS/ESA Diagnosis Reference* manual for details of trace point ID LD 0002, you will see that the data area is the LDLD parameter list.

Interpreting user trace entries

User trace entries have point IDs in the range AP 0000 through AP 00C2, the numeric part of the point ID being specified in the application.

Extended format user trace entries show a user-defined resource field, and a user-supplied data field that can be up to 4000 bytes in length. A typical extended-format entry is shown in Figure 36.

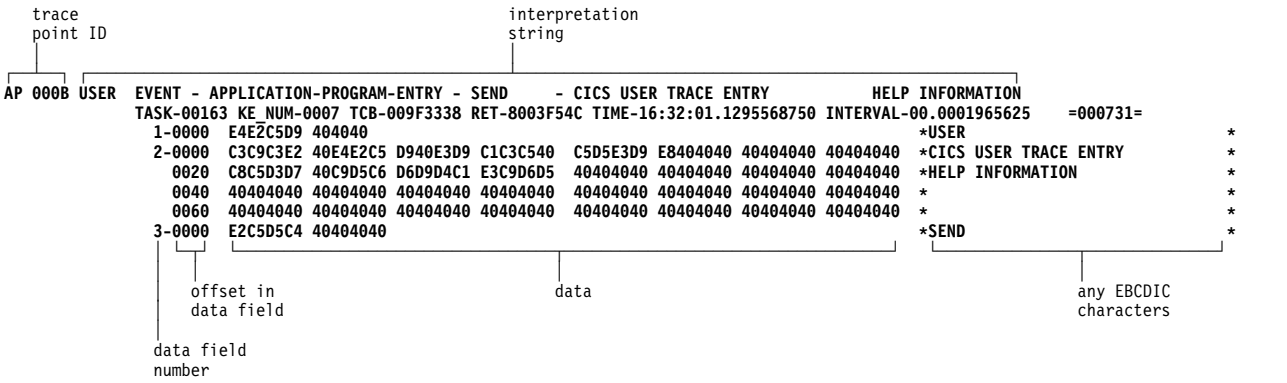


Figure 36. Example of the extended format for a user trace entry

The interpretation string for the entry contains the string "APPLICATION-PROGRAM-ENTRY", to identify this as a user trace entry, and the resource field.

There are three data fields on an extended-format user trace entry:

1. The character string "USER".
2. User data from the area identified in the FROM parameter of the trace command.
3. The resource field value identified in the RESOURCE parameter of the trace command.

The abbreviated-trace entry corresponding to the extended trace entry of Figure 36 is shown in Figure 37.

Figure 37. Example of the abbreviated format for a user trace entry

Abbreviated-format trace entries show the user resource field in the interpretation string. There is also an optional user data field that contains as much user-specified data as can be fitted into the line. If successive user trace entries

have the same resource field value, but different data field values, you might need to refer to the corresponding extended trace entries to assess their significance.

Normal CICS tracing

"Normal" CICS tracing is handled by trace domain. It can be used to trace the flow of execution through CICS code, and through your applications as well. For programming information about how to make trace calls from within your own programs, refer to the *CICS/ESA Application Programming Reference* manual. You can see what functions are being performed, what parameters are being passed, and the values of important data fields at the time trace calls are made. This type of tracing is also used for "first failure data capture", if an exception condition is detected by CICS.

The principal feature of this type of CICS tracing is the control you have over the amount of system tracing that is done. You can select tracing by transaction, by CICS component, and by level of detail.

You can select any combination of internal tracing, auxiliary tracing and GTF tracing to be active at the same time. Your choice has no effect on the selectivity with which system tracing is done, but each type of tracing has a set of characteristic properties. These properties are described in "CICS internal trace" on page 237, "CICS auxiliary trace" on page 238, and "CICS GTF trace" on page 239.

Trace points and trace levels

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 in the *CICS/ESA Diagnosis Reference* manual.

Some trace points are used to make exception traces when exception conditions occur, and some are used to trace the mainline execution of CICS code. Trace points of the latter type each have an associated "level" attribute. The value of this attribute depends on where the trace point is, and the sort of detail it can provide on a trace call.

Trace levels can, in principle, vary in value in the range 1–32, but in practice nearly all mainline trace points have a trace level of 1 or 2.

Level-1 trace points are designed to give you enough diagnostic information to fix "user" errors. The following is a summary of where they are located, and a description of the information they return:

- On entry to, and exit from, every CICS domain. The information includes the domain call parameter list, and data whose address is contained in the parameter list if it is necessary for a high-level understanding of the function to be performed.
- On entry to, and exit from, major internal domain functions. The information includes parameters passed on the call, and any output from the function.
- Before and after calls to other programs, for example, VTAM. The information includes what request is to be made, the input parameters on the request, and the result of the call.

- At many of the points where trace calls were made in CICS/MVS Version 2. The type of information is the same as for that release.

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 won't 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.

Level-3 trace points and above are reserved for special cases. Very few components have trace points higher than 2, and they are only likely to be of use to IBM support staff.

You can select how much CICS system tracing is to be done on the basis of the trace level attributes of trace points. You can make your selection independently for each CICS component, and you can also vary the amount of tracing to be done for each task. This gives you control over what system tracing is done.

Note: In the storage manager component (SM), two levels of tracing, level 3 and level 4, are intended for IBM field engineering staff. These trace levels take effect only if specified in system initialization parameters and modify the internal SM operation for CICS subpools as follows:

SM level 3 trace The quickcell mechanism is deactivated. Every CICS subpool, regardless of quickcelling requirements, will issue domain calls for getmain and freemain services, and these calls will be traced.

SM level 4 trace Subpool element chaining on every CICS subpool is forced. Every CICS subpool, regardless of element chaining requirements, will use element chaining.

A significant performance overhead is introduced into your CICS system if these storage manager trace levels are selected. Note that specifying SM=ALL activates SM trace levels 1, 2, 3, and 4.

Specifying the system tracing you want

For each transaction, you can specify whether *standard* tracing or *special* tracing is to be done, or whether tracing is to be suppressed for that transaction altogether.

For each component, you can specify two sets of trace level attributes. The trace level attributes define the trace point IDs to be traced for that component when standard task tracing is being done and when special task tracing is being done, respectively.

If you are running a test region, you probably have background tracing most of the time. In this case, the default tracing options (standard tracing for all transactions, and level-1 trace points only in the standard set for all components) probably suffice. All you need do is to enable the required trace destinations and set up any related tracing options. Details are given in "Selecting trace destinations and related options" on page 237.

For a production system, background tracing might incur an unacceptable processing overhead. If you find this to be so, you are recommended to set up tracing so that exception traces *only* are recorded on an auxiliary trace data set. There need be no other tracing overhead, and you can be sure that the exception trace will be preserved even when the event invoking the trace does not cause a system dump to be taken. For details, see "CICS exception tracing" on page 242.

When specific problems arise, you can set up special tracing so you can focus on just the relevant tasks and components. A scheme for specifying the tracing you need is outlined in Figure 38 on page 229.

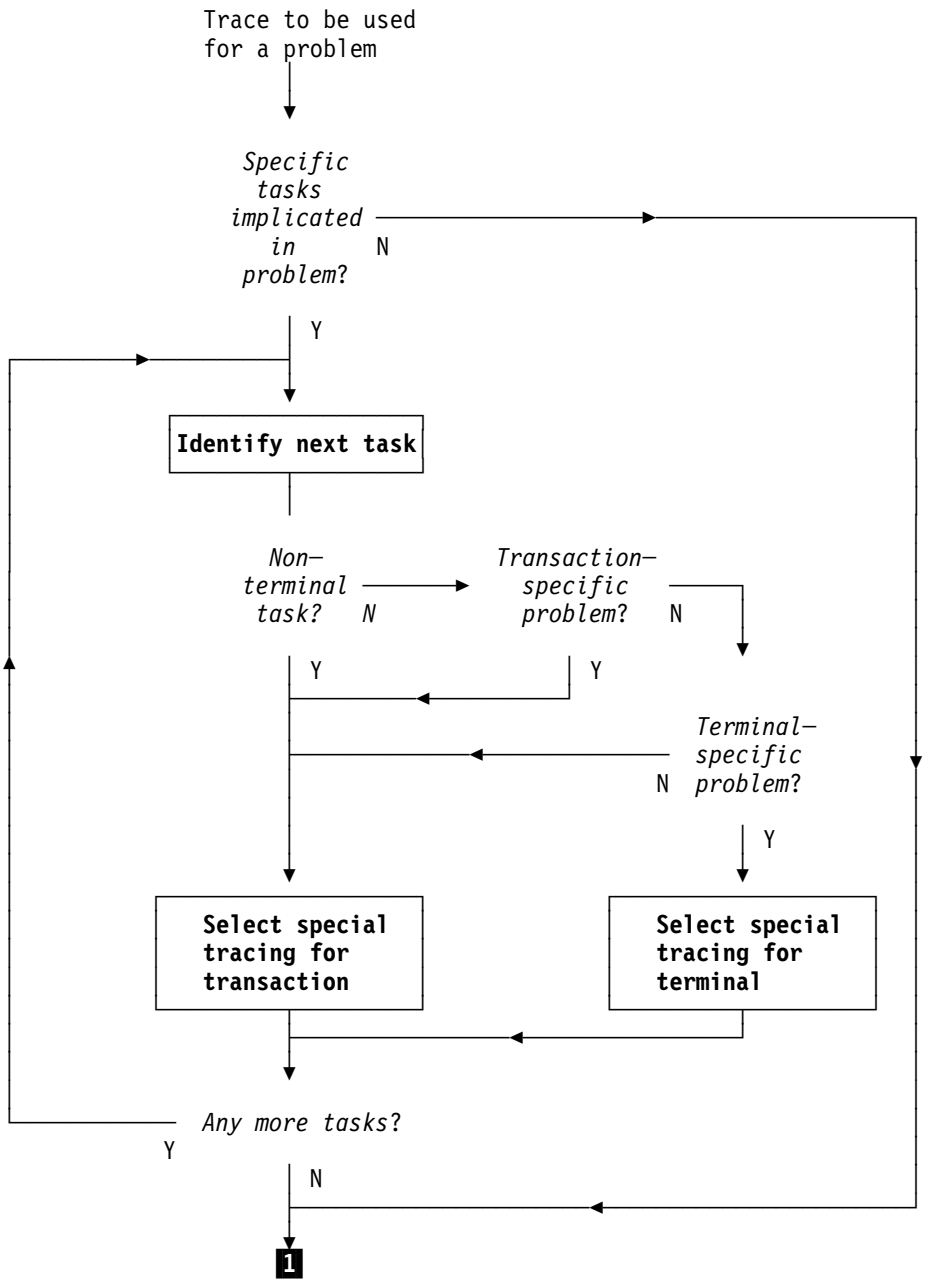


Figure 38 (Part 1 of 3). Outline scheme for setting up special tracing for problems

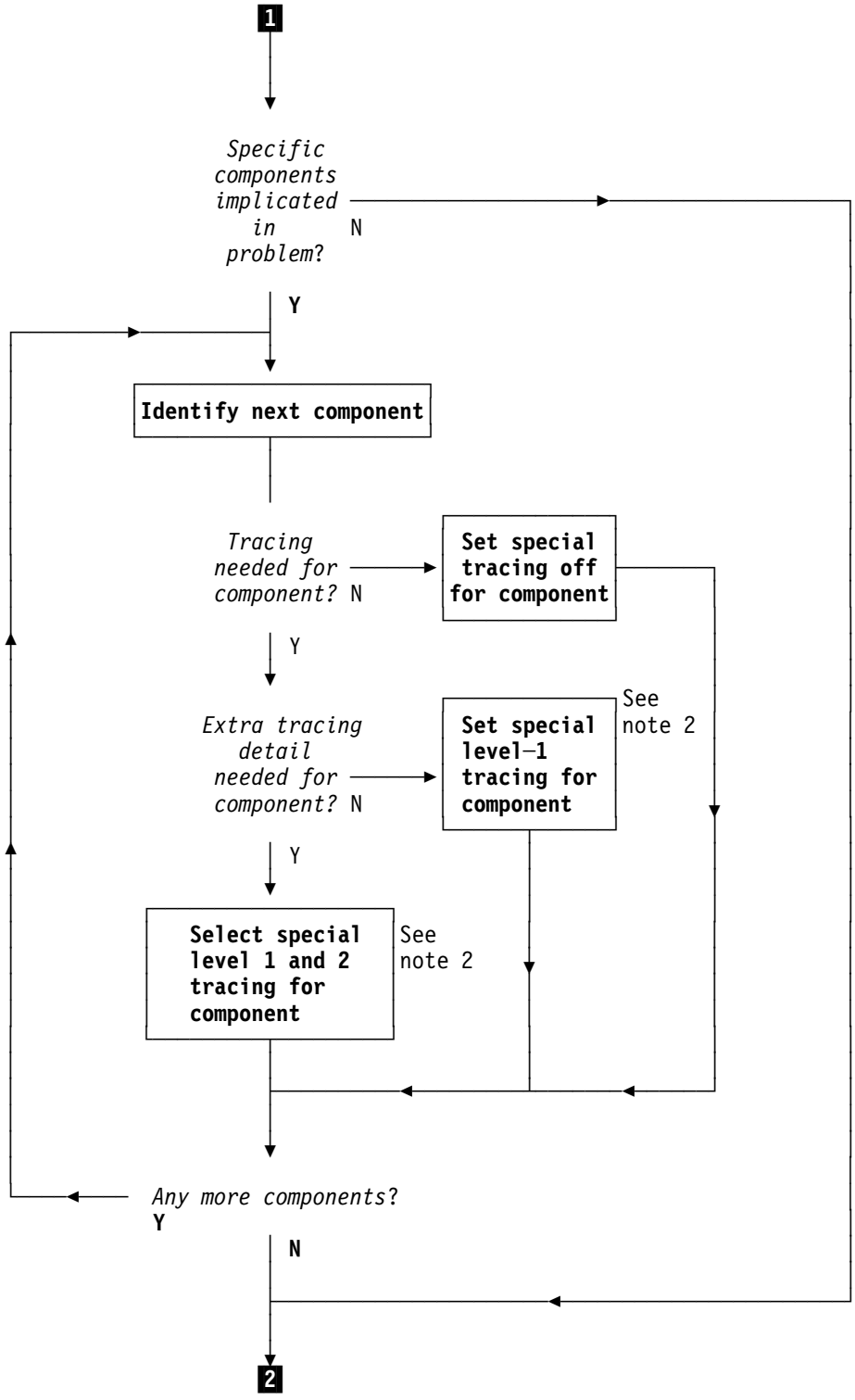


Figure 38 (Part 2 of 3). Outline scheme for setting up special tracing for problems

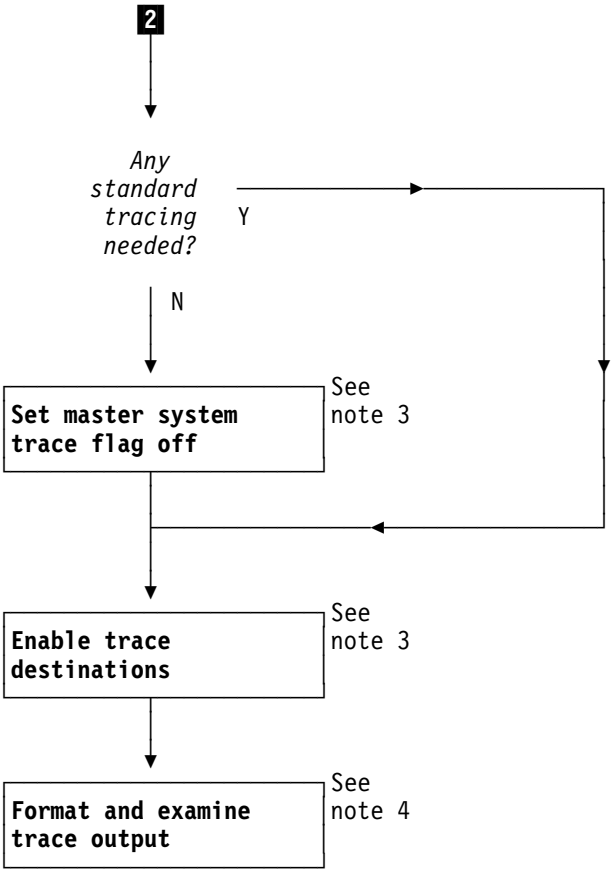


Figure 38 (Part 3 of 3). Outline scheme for setting up special tracing for problems

Notes:

1. See “Tracing for selected tasks”
2. See “Selecting component tracing” on page 235
3. See “Setting the tracing status” on page 239
4. See “Formatting and interpreting trace entries” on page 219; “Interpreting extended-format CICS system trace entries” on page 220; and “Interpreting abbreviated-format CICS system trace entries” on page 224.

Tracing for selected tasks

You can select which tasks are to have standard tracing, which are to have special tracing, and which are to have tracing suppressed. If you specify standard tracing for a task, trace entries are made at all the trace points in the standard set. If you specify special task tracing, you get trace entries at all the trace points in the special set. If you suppress tracing for a task, you don’t get any tracing done (except exception tracing) when that task is running.

For transactions that run at terminals, a task is considered to be an instance of a transaction run at a specific terminal. By defining the type of tracing you want by transaction and terminal, you automatically define what task tracing is to be done.

For non-terminal transactions, a task is just an instance of the transaction. The type of tracing you define for the transaction alone defines the type of task tracing that is to be done.

The type of task tracing you get for the various combinations of transaction tracing and terminal tracing is summarized in the truth table shown in Figure 39.

OPTION on TRANSACTION	**OPTION on TERMINAL**	
	standard tracing	**special tracing**
tracing suppressed	SUPPRESSED	SUPPRESSED
standard tracing	STANDARD	SPECIAL
special tracing	SPECIAL	SPECIAL

Figure 39. The combination of task trace options

You can set up the task tracing you want using the CETR transaction, with the screen shown in Figure 40. You need to type in the transaction ID or the terminal ID or the netname for the terminal, together with the appropriate tracing.

The status can be any one of STANDARD, SPECIAL, or SUPPRESSED for the transaction, and either STANDARD or SPECIAL for the terminal.

This screen can also be used to set up certain other terminal tracing options. You can select ZCP tracing for a named terminal (trace point ID AP 00E6), and you can also select CICS VTAM exit tracing for the terminal. For more details about CICS VTAM exit tracing, see “CICS VTAM exit tracing” on page 246.

```
CETR                  Transaction and Terminal Trace
Type in your choices.
Item                              Choice    Possible choices
Transaction ID             ===>             Any valid 4 character ID
Transaction Status         ===>             STandard, SPecial, SUppressed
Terminal ID                ===>             Any valid Terminal ID
Netname                    ===>             Any valid Netname
Terminal Status            ===>             STandard, SPecial
Terminal VTAM Exit Trace   ===>             ON, OFf
Terminal ZCP Trace         ===>             ON, OFf
VTAM Exit override         ===>  NONE       All, System, None
When finished, press ENTER.
PF1=Help      3=Quit     6=Cancel Exits      9=Error List
```

Figure 40. CETR screen for specifying standard and special task tracing

The CETR transaction can, for example, help you to get standard tracing for a transaction when it is run at one terminal, special tracing when it is run at a second terminal, and no tracing at all when it is run at any other terminal.

Notes:

1. You can turn standard tracing off for *all* tasks by setting the master system trace flag off. You can do this with the CETR transaction, using the screen shown in Figure 43 on page 240, or you can code SYSTR=OFF at system initialization. However, any special task tracing will continue—it is not affected by the setting of the system master trace flag.
2. If you run with standard tracing turned off and you specify levels of tracing for the required components under the "Special" heading in the “Components Trace Options” screen shown in Figure 42 on page 236, you can use CETR to trace a single transaction. To do this, specify the transaction ID and a transaction status of SPECIAL, on the screen shown in Figure 40 on page 232.

The tracing logic used by CICS

The logic used by CICS to decide whether a trace call is to be made from a trace point is shown in Figure 41. It is assumed that at least one trace destination is STARTED.

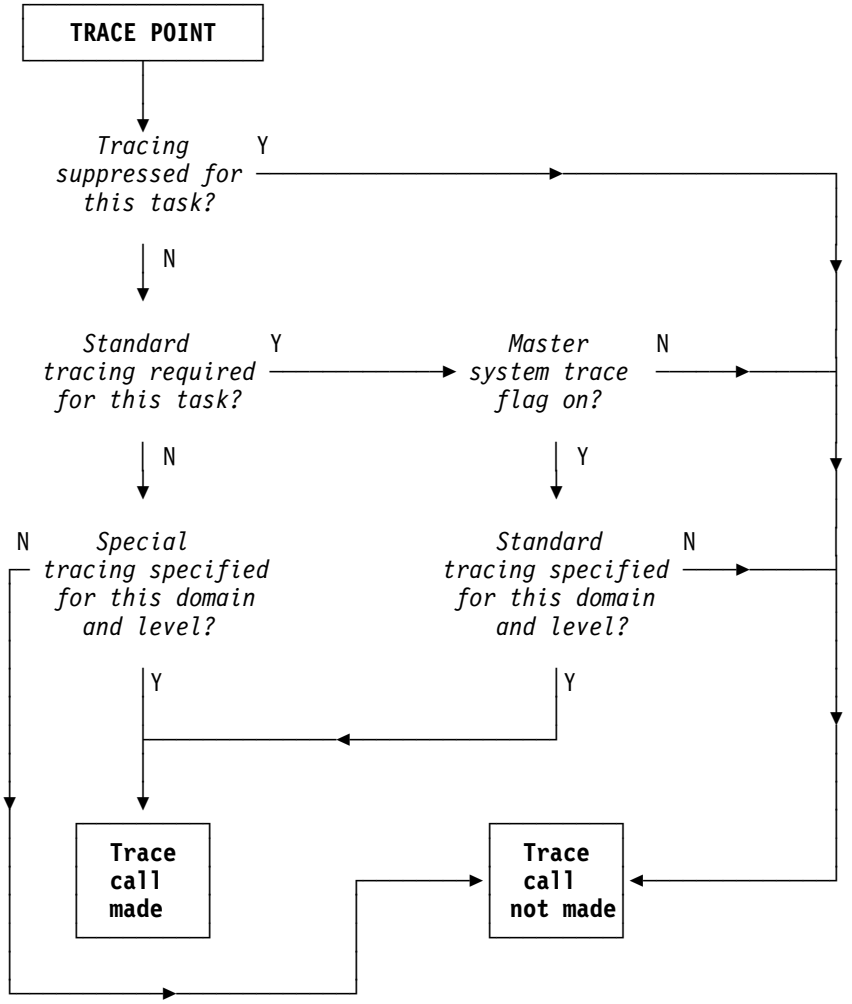


Figure 41. Logic used to determine if a trace call is to be made from a trace point

Selecting component tracing

"Component names and abbreviations" lists the components for which you can select trace levels for standard and special tracing. You can reference this list online through CETR, by pressing PF1 on the component screen (see Figure 42 on page 236).

You need to decide for each component the trace levels to be used for both standard and special tracing. You can define this either during system initialization, or online using the CETR transaction.

Note: The component codes BF, BM, CP, DC, DI, EI, FC, IC, IS, JC, KC, PC, SC, SP, SZ, TC, TD, TS, and UE are subcomponents of the AP domain. The corresponding trace entries are produced with a point ID of AP nnnn.

For example, trace point AP 0471 is a file control level-1 trace point and AP 0472 is a file control level-2 trace point. These trace points are produced only if the trace setting for the FC component is "(1,2)" or "ALL". The component code AP is used for trace points from the AP domain that do not fall into any of the subcomponent areas listed above.

Component names and abbreviations

Code	Component name
AP	Application domain
BF	Built-in function
BM	Basic mapping support
CP	Common programming interface
DC	Dump compatibility layer
DD	Directory manager domain
DI	Batch data interchange
DM	Domain manager domain
DS	Dispatcher domain
DU	Dump domain
EI	Exec interface
FC	File control
GC	Global catalog domain
IC	Interval control
IS	ISC or IRC
JC	Journal control
KC	Task control
KE	Kernel
LC	Local catalog domain
LD	Loader domain
LM	Lock domain
ME	Message domain
MN	Monitoring domain
PA	Parameter domain
PC	Program control
PG	Program manager
SC	Storage control
SM	Storage manager domain
SP	Syncpoint
ST	Statistics domain
SZ	Front End Programming Interface
TC	Terminal control
TD	Transient data
TI	Timer domain
TR	Trace domain
TS	Temporary storage
UE	User exit interface
US	User domain
XM	Transaction manager
XS	Security manager

Defining component tracing at system initialization: You can code any of the following parameters to define component tracing at CICS system initialization time:

- SPCTR, to indicate the level of special tracing required for CICS as a whole.
- SPCTRxx, where xx is one of the two-character component identifiers that specify the level of special tracing you require for a particular CICS component (see “Component names and abbreviations” on page 235).
- STNTR, to indicate the level of standard tracing required for CICS as a whole.
- STNTRxx, where xx is one of the two-character component identifiers that specify the level of standard tracing you require for a particular CICS component (see “Component names and abbreviations” on page 235).

For more information about SIT parameters, see the *CICS/ESA System Definition Guide*.

Defining component tracing when the CICS system is running: You can use the CETR transaction to define component tracing dynamically on the running CICS system.

Figure 42 shows you what the CETR Component Trace Options screen looks like. To make changes, you overtype the settings shown on the screen, and then press ENTER.

```
CETR                    Component Trace Options
Overtype where required and press ENTER.                    PAGE 1 OF 2
Component Standard                          Special
--------  --------------------------------  --------------------------
  AP      1                                 1-2
  BF      1                                 OFF
  BM      1                                 OFF
  CP      1                                 1-2
  DC      1                                 OFF
  DI      1                                 1
  DM      1                                 1-2
  DS      1                                 1-2
  DU      1                                 1-2
  EI      1                                 1
  FC      1                                 1-2
  GC      1                                 1-2
  IC      1                                 1
  IS      1                                 OFF
  JC      1                                 1
  KC      1                                 1
  KE      1                                 1-2
  LC      1                                 1-2
PF:  1=Help   3=Quit   7=Back  8=Forward  9=Messages   ENTER=Change
```

Figure 42. CETR screen for specifying component trace options

With the settings shown, trace entries are made as follows:

- With standard task tracing in effect, from level-1 trace points of all the components listed.

- With special task tracing in effect:
  - From level-1 trace points only for components DI, EI, IC, JC and KC
  - From both level-1 and level-2 trace points for components AP, CP, DM, DS, DU, FC, GC, KE, and LC

  No special task tracing is done for components BF, BM, DC, and IS.

Selecting trace destinations and related options

The trace destinations you select and the options you define for the destinations depends on your specific problem determination requirements. You can select any combination of CICS internal tracing, CICS auxiliary tracing, and CICS GTF tracing. Your decision must be based on the characteristics of the various types of CICS tracing, which are described in “CICS internal trace,” “CICS auxiliary trace” on page 238, and “CICS GTF trace” on page 239. You need to decide how much trace data you need to capture, and whether you want to integrate CICS tracing with tracing done by other programs.

You can control the status and certain other attributes of the various types of CICS tracing either dynamically, using the CETR transaction, or during system initialization, by coding the appropriate system initialization parameters.

CICS internal trace

You can select a status of STARTED or STOPPED for internal trace. If you select STARTED, any trace calls that are made cause trace entries to be directed to the internal trace table. This occupies a sequence of pages in virtual storage above the 16MB line, in MVS GETMAIN storage.

The internal trace table has a minimum size of 16KB, and a maximum size of 1048576KB. The table is extendable from 16KB in 4KB increments. You can change the size of the table dynamically, while CICS is running, but if you do so you lose all of the trace data that was present in the table at the time of the change. If you want to keep the data *and* change the size of the table, take a system dump before you make the change.

The internal trace table wraps when it is full. When the end of the table is reached, the next entry to be directed to the internal trace entry goes to the start, and overlays the trace entry that was formerly there. In practice, the internal trace table cannot be very big, so it is most useful for background tracing or when you don’t need to capture an extensive set of trace entries. If you need to trace CICS system activity over a long period, or if you need many entries over a short period, one of the other trace destinations is likely to be more appropriate.

Note that the internal trace table is always present in virtual storage, whether you have turned internal tracing on or not. The reason is that the internal trace table is used as a destination for trace entries when CICS detects an exception condition. Other trace destinations that are currently selected get the exception trace entry as well, but the entry always goes to the internal trace table even if you have turned tracing off completely. This is so that you get “first failure data capture”.

CICS auxiliary trace

CICS auxiliary trace entries are directed to one or other 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.

You can use the AUXTR system initialization parameter to turn CICS auxiliary trace on or off in the system initialization table.

You can select a status of STARTED, STOPPED, or PAUSED for CICS auxiliary trace dynamically using the CETR transaction. These statuses reflect both the value of the auxiliary trace flag, and the status of the current auxiliary trace data set, in the way shown in Table 29.

Table 29. The meanings of auxiliary trace status values

Auxiliary tracing status	Auxiliary trace flag	Auxiliary trace data set
STARTED	ON	OPEN
STOPPED	OFF	CLOSED
PAUSED	OFF	OPEN

When you first select STARTED for CICS auxiliary trace, any trace entries are directed to the initial auxiliary trace data set. If CICS terminated normally when auxiliary trace was last active, this is the auxiliary trace data set that was not being used at the time. Otherwise, it is the DFHAUXT data set. If you initialize CICS with auxiliary trace STARTED, DFHAUXT is used as the initial auxiliary trace data set.

What happens when the initial data set is full depends on the status of the auxiliary switch (AUXTRSW). This can have a value of NO, NEXT, or ALL, and you can define it either in the system initialization table, using the AUXTRSW system initialization parameter, or dynamically using the CETR transaction.

NO means that when the initial data set is full, no more auxiliary tracing is done.

NEXT means that when the initial data set is full, then the other data set receives the next trace entries. However, when *that* one is full, no more trace data is written to auxiliary trace.

ALL means that auxiliary trace data is written alternately to each data set, a switch being made from one to the other every time the current one becomes full. This means that trace entries already present in the trace data sets start getting overwritten when both data sets become full for the first time.

The advantage of using auxiliary trace is that you can collect large amounts of trace data, if you initially define large enough trace data sets. For example, you might want to do this to trace system activity over a long period of time, perhaps to solve an unpredictable storage violation problem.

A time stamp is included in the header line of every page of abbreviated auxiliary trace output to help match external events with a particular area of the trace, and thus help you to find the trace entries that are of interest.

CICS GTF trace

CICS GTF trace is an invocation of MVS GTF trace, and the trace entries are directed to a destination defined to MVS. This can be either a trace table in main storage, or a single external GTF trace data set.

You can switch CICS GTF trace on or off by using the GTFTR system initialization parameter.

You can select a status of STARTED or STOPPED for CICS GTF trace dynamically using the CETR transaction. MVS GTF trace must be started with the TRACE=USR option before CICS GTF trace is started, because otherwise no trace entries can be written.

In a multi-system environment, trace entries from CICS Version 3 and CICS Version 4.1 can be recorded in the GTF trace data set. Entries from both releases are formatted by the CICS-supplied routine, DFHTG410. See the *CICS/ESA Operations and Utilities Guide* for details of how to format and print the GTF trace data set.

When the GTF trace data set is full, it wraps. The next trace entries are written at the start of the data set, and the entries that were formerly there are overlaid. Thus, you need to define a data set that is big enough to capture all the trace entries that interest you.

The MVS GTF trace destination can be used not only by CICS, but by other programs as well. This gives you the opportunity of integrating trace entries from CICS with those from other programs. A single GTF trace data set could, for example, contain trace entries made by both CICS and VTAM. You can relate the two types of trace entry using a unique task identifier in the trace header, known to both CICS and VTAM.

Because different products are likely to execute asynchronously, be cautious about using the sequence of trace entries in the GTF trace data set as evidence when you investigate problems.

Setting the tracing status

You can set the system tracing status by coding the appropriate system initialization parameters, and you can also set it dynamically, using the CETR transaction. The CETR transaction enables you to make changes in response to contingencies, as they arise.

Setting the tracing status at system initialization: These are the parameters that you can use to set up tracing status at system initialization:

- AUXTR, to specify whether auxiliary trace is to be activated at system initialization.
- AUXTRSW, to define the auxiliary switch status.
- GTFTR, to enable CICS to use MVS GTF tracing.
- INTTR, to specify whether internal tracing is to be activated at system initialization.
- SYSTR, to set the master system trace flag on or off.
- TRTABSZ, to specify the size of the internal trace table.

- USERTR, to set the master user trace flag on or off. It must be on if user trace calls are to be made from your applications.

For more details of SIT parameters, see the *CICS/ESA System Definition Guide*.

Setting the tracing status using CETR: You can use the CICS/ESA trace control transaction (CETR) to set the tracing status. Figure 43 shows you the CETR screen you can use to set the tracing status dynamically.

```
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 Dataset       ===>  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 43. Tracing options shown on a CETR screen

In this example, internal tracing status is STOPPED, and so regular tracing is not directed explicitly to the internal trace table. However, note that the internal trace table is used as a buffer for the other trace destinations, so it always contains the most recent trace entry if at least one trace destination is STARTED. The internal trace table is also used as a destination for exception trace entries, which are made whenever CICS detects an exception condition. If such a condition were detected when the options shown in this example were in effect, you would be able to find the exception trace entry in the internal trace table as well as in the GTF trace data set.

The internal trace table size is 16KB, which is the minimum size it can be. If internal trace were STARTED, the trace table would wrap when it became full.

The current auxiliary trace data set is B, meaning that trace entries would be written to DFHBUXT if auxiliary tracing were started. However, its status is shown to be PAUSED, so no tracing is done to that destination. The auxiliary switch status is ALL, so a switch would be made to the other auxiliary trace data set whenever one became full.

The GTF trace status is shown to be STARTED, which means that CICS trace entries are written to the GTF trace data set defined to MVS. Be aware that no error condition is reported if the CICS GTF status is started but GTF tracing has not been started under MVS. If this happens, the trace entries are not written.

The master system trace flag is OFF. This means that no standard tracing is done at all, even though standard tracing might be specified for some tasks. However, special task tracing is not affected—the master system trace flag only determines whether standard task tracing is to be done.

You can see the role of the master system trace flag in Figure 41 on page 234.

The master user trace flag is OFF, so no user trace entries can be made from applications. You must set the master user trace flag on before any user trace requests in your programs can be serviced. If it were off, any trace call requests in your programs would be ignored.

The logic used to ensure that trace entries are written to the required destinations is shown in Figure 44.

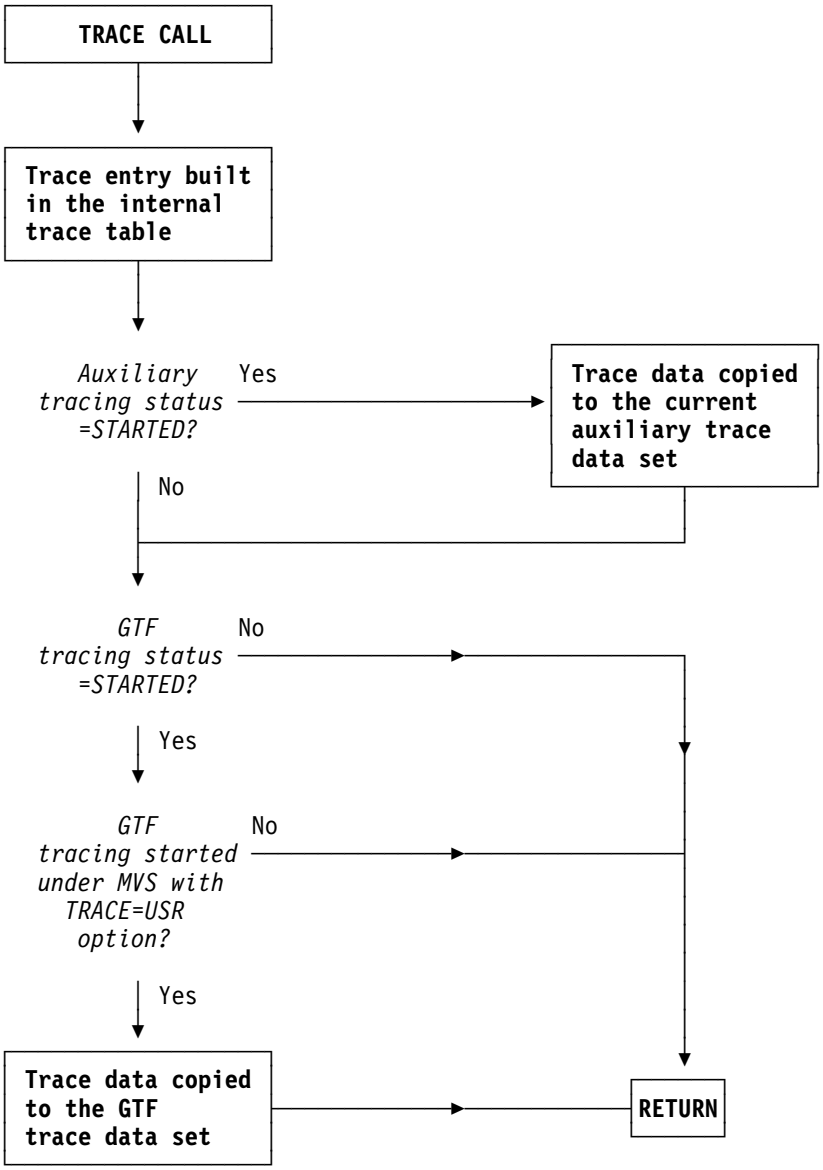


Figure 44. How trace entries are directed to the required destinations

CICS exception tracing

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.

CICS uses a similar mechanism for both exception tracing and “normal” tracing. Exception trace entries are made from specific points in CICS code, and data is taken from areas that might provide information about the cause of the exception. The first data field in the trace entry is usually the parameter list from the last domain call, because this can indicate the reason for the exception.

The exception trace points do not have an associated “level” attribute, and trace calls are only ever made from them when exception conditions occur.

Exception trace entries are always written to the internal trace table, even if no trace destinations at all are currently STARTED. That is why there is always an internal trace table in every CICS region, to make sure there is always somewhere to write exception trace entries. If the other trace destinations are STARTED, the exception trace entries are written there, as well.

You can select tracing options so that exception traces *only* are made to an auxiliary trace data set. This is likely to be useful for production regions, because it enables you to preserve exception traces in auxiliary storage without incurring any general tracing overhead. You need to disable all standard and special task tracing, and enable auxiliary trace:

1. Ensure that special tracing has not been specified for any task.
2. Set the master system trace flag off.
3. Set the auxiliary trace status to STARTED, and the auxiliary trace data set and the auxiliary switch status to whatever values you want.

Exception traces are now made to an auxiliary trace data set, but there is no other tracing overhead.

The format of an exception trace entry is almost identical to that of a normal trace entry. However, you can identify it by the eye-catcher ***EXC*** in the header.

Note: Exception conditions that are detected by MVS—for example, operation exception, protection exception, or data exception—do not cause a CICS exception trace entry to be made directly. However, they do cause a CICS recovery routine to be invoked, and that, in turn, causes a “recovery” exception trace entry to be made.

User exception trace entries

The EXCEPTION option on the EXEC CICS ENTER TRACENUM command enables user programs to write a trace entry to the trace destinations, even when the master user trace flag is off. User exception trace entries are always written to the internal trace table (even if internal tracing is set off), but are written to other destinations only if they are active.

The user exception trace entries CICS writes are identified by the character string *EXCU in any formatted trace output produced by CICS utility programs. For example, an application program exception trace entry generated by an EXEC CICS ENTER TRACENUM() EXCEPTION command appears in formatted trace output as:

```
USER *EXCU - APPLICATION-PROGRAM-EXCEPTION
```

If you use the exit programming interface (XPI) trace control function to write user trace entries, you can use the DATA1 block descriptor to indicate whether the entry is an exception trace entry. Enter the literal 'USEREXC' in the DATA1 field on the DFHTRPTX TRACE_PUT call to identify an exception trace entry. This is interpreted by the trace formatting utility program as follows:

```
USER *EXCU - USER-EXIT-PROGRAM-EXCEPTION
```

See the *CICS/ESA Customization Guide* for programming information about XPI trace control function.

CICS XRF tracing

CICS XRF tracing is always active when you are running with XRF. It is used by the CICS availability manager (CAVM), and you cannot turn it off. However, it only makes about 12 entries every 2 seconds, so the overhead is not great. Note that CICS XRF tracing is quite distinct from the "normal" CICS tracing that can originate from the CAVM, which is identified by trace point IDs AP 00C4 through AP 00C7.

The XRF trace entries are 32 bytes long and are written to a trace table in main storage. The table has a fixed size of 64KB, and it wraps around when it is full.

The table starts with 28 bytes of control information, in the format shown in Table 30.

Table 30. Control information at the start of the XRF trace table

Bytes	Contents
0–15	'*** XRF TRACE **'
16–19	Address of start of trace entries
20–23	Address of end of trace entries
24–27	Address of end of most recent entry

Trace entries are 32 bytes long, and have the format shown in Table 31.

Table 31. Format of an XRF trace entry

Bytes	Contents
0	Type code
1	Subtype
2–3	Process ID of XRF process that made the entry
4–27	Trace data—the format depends on the type or the subtype
28–31	Clock value when entry was made, same format as "normal" CICS trace entries

Process IDs are assigned in order of process ATTACH starting from 1. Some special values are used for processes which are not known to the dispatcher, but which cause trace entries to be made. These are:

Process ID	Function
X'0000'	Initial attach
X'FFFE'	ESPIE/ESTAE error handling
X'FFFF'	Dispatcher activities.

Entry types

The entries are as follows:

```
Module   Type Subtype Description

DFHWLGET  1    1      Module entry
                        4-11  Module name
                       12-15  LIFO allocation address

DFHWLFRE  1    2      Module return
                        4-11  Module name
                       12-15  LIFO allocation address
                       16-27  0

DFHWDATT  2    1      XRF process attach
                        4- 7  Process entry point
                        8-11  Initial data parameter
                       12-15  Address of ESPIE routine
                       16-19  Address of ESTAE routine
                       20-23  Address of attached process XPB
                       24-27  Process ID attached process XPB

DFHWDISP  2    2      XRF process detach
                        4-27  0

DFHWDISP  2    3      XRF process dispatch
                        4- 7  Address of external ECB waited for (if any)
                        8-11  Address of internal ECB waited for (if any)
                       12-15  Awaited broadcast events which were
                               posted
                       16-19  Broadcast events still posted for
                               this process
                       20-23  Address of process XPB
                       24-27  Locks held by this process

DFHWDWAT  2    4      XRF process wait (event data)
                        4- 7  Address of external ECB to wait for (if any)
                        8-11  Address of internal ECB to wait for (if any)
                       12-15  Broadcast events to wait for (if any)
                       16-19  Events to be broadcast to all processes
                       20-23  Events to be reset for this process
                       24-27  0
```

Module	Type	Subtype	Description
DFHWDAT	2	5	XRF process wait (lock data) 4- 7 Locks to be freed 8-11 Locks to be acquired 12-19 0 20-23 Locks held by all other processes at time of call 24-27 Locks held by this process at time of call
DFHWDISP	2	6	Dispatcher termination 4-27 0
DFHDISP	2	7	Dispatcher issuing OS WAIT 4-19 0 20-23 Address of WAIT list 24-27 Number of ECBs in WAIT list
DFHWDISP	2	8	Dispatcher resume after OS WAIT 4-27 0
DFHWMMT	3	1	Message manager issuing VSAM GET 4- 7 RPL address 8-11 RBA of CI to be read 12-27 0
DFHWMMT	3	2	Message manager issuing VSAM PUT 4- 7 RPL address 8-11 RBA of CI to be written 12-27 0
DFHWMMT	3	3	Message manager I/O complete 4- 7 RPL address 8-11 RBA of CI to be read 12 0 13-15 VSAM feedback information 16-27 0
DFHWMQH	4	1	Message manager message received 4- 7 0 8-11 Queue name 12-15 Message sequence number 16-19 Address of message block (contains message copy) 20-27 0
DFHWMWR	4	2	Message manager message sent 4- 7 0 8-11 Queue name 12-15 Message sequence number 16-19 Message file cycle number 20-23 RBA of this message 24-25 0 26-27 Response to PUTMSG request (WMSRESP)

Program check and abend tracing

Program check and abend tracing is carried out by kernel domain. The kernel records information about program checks and abends, including details of the registers and the PSW at the time of failure.

You cannot format the program check and abend trace information directly, but you get a summary of its contents in a formatted CICS system dump when you specify dump formatting keyword KE. The information is provided in the form of a storage report for each task that has had a program check or an abend during the current run of CICS.

An example of such a storage report is given in Figure 3 on page 51.

CICS VTAM exit tracing

CICS VTAM exit tracing gives you a way of tracing VTAM requests made from CICS. You can control it online, using CETR—see Figure 40 on page 232 for an illustration of the screen you need to use.

When CICS issues a VTAM request, VTAM services the request asynchronously and CICS continues executing. When VTAM has finished with the request, it returns control to CICS by driving a CICS VTAM exit. Every such exit contains a trace point, and if CICS VTAM exit tracing is active, a trace entry is written to the GTF trace data set. GTF tracing must be active, but you do not need to start it explicitly from CICS. It is enough to start VTAM exit tracing from the CETR transaction and terminal trace panel.

Note: The GTF trace data set can receive trace entries from a variety of jobs running in different address spaces. You need to identify the trace entries that have been made from the CICS region that interests you. You can do this by looking at the job name that precedes every trace entry in the formatted output.

You can use this type of tracing in any of the cases where you might want to use VTAM buffer tracing, but it has the advantage of being part of CICS and, therefore, controllable from CICS. This means that you don't need a good understanding of VTAM system programming to be able to use it. CICS VTAM exit tracing also has the advantage of tracing some important CICS data areas relating to VTAM requests, which might be useful for diagnosing problems.

Controlling CICS VTAM exit tracing

You can turn CICS VTAM exit tracing on and off using the CETR transaction. You can specify tracing for just a single terminal, or for all the terminals in the VTAM network. However, you cannot select which CICS exits are to be traced. Whenever CICS VTAM exit tracing is running, you get a trace entry every time a CICS exit is driven by VTAM.

If you select "normal" CICS tracing for the affected terminals at the same time as you have CICS VTAM exit tracing running, you can then correlate CICS activities more easily with the asynchronous processing done by VTAM.

If you need to turn on CICS VTAM exit tracing in an application owning region (AOR) while you are signed-on to a terminal in a terminal owning region (TOR), follow these steps:

1. Invoke CETR on the AOR.
2. Press PF5 to call up the CETR transaction and terminal trace screen.
3. Enter the APPLID of the TOR in the NETNAME field.
4. Complete other fields as required.
5. Press Enter.

CICS VTAM trace entries are always written to the GTF trace data set, and you can format them in the usual way—see “Formatting and interpreting trace entries” on page 219. Direct all “normal” CICS tracing to the GTF trace destination as well, so you get the regular trace entries and the CICS VTAM exit trace entries in sequence in a single data set. If you send the normal tracing to another destination, you get only the isolated traces from the exit modules with no idea of related CICS activity.

Interpreting CICS VTAM exit trace entries

CICS VTAM exit trace entries can be identified by their trace point IDs, which are in the range AP FC00 through AP FCFF. Not all the values in the range are used.

The format of the trace entries is similar to that shown in Figure 32 on page 223. The interpretation string contains the netname of the terminal to which the trace entry relates, if the trace entry was made from a terminal-specific trace point. This makes it easy to identify the terminal associated with the VTAM request. The trace entries also contain data from one or more selected CICS data fields, for example from the TCTTE. For guidance on interpreting the data values you might find there, see the *CICS/ESA User's Handbook*.

VTAM buffer tracing

VTAM buffer tracing enables you to look at all the data that is passed between logical units on a VTAM communication link.

The trace entries, which include the netname of the terminal to which they relate, are made to the GTF trace data set. If you want to send “normal” CICS trace entries there, you can rationalize the activities of CICS with the asynchronous activities of VTAM. For details of VTAM buffer tracing, see the appropriate manual in the VTAM library.

Using FEPI trace

For information about using trace to solve FEPI problems, refer to the *CICS/ESA Front End Programming Interface User's Guide*. THIS OVERWRITES CLSS100 DFHS1302 (4.1)

Chapter 15. Using dumps in problem determination

You have the choice of two different types of CICS dump to help you with problem determination. They are the **transaction dump**, of transaction-related storage areas, and the **CICS system dump**, of the entire CICS region.

The type of dump to use for problem determination depends on the nature of the problem. In practice, the system dump is often more useful, because it contains more information than the transaction dump. You can be reasonably confident that the system dump has captured all the evidence you need to solve your problem, but it is possible that the transaction dump might have missed some important information.

The amount of CICS system dump data that you could get is potentially very large, but that need not be a problem. You can leave the data on the system dump data set, or keep a copy of it, and format it selectively as you require.

You can control the dump actions taken by CICS, and also what information the dump output contains.

Controlling dump action

There are two aspects to controlling dump action:

- Setting up the dumping environment, so that appropriate dump action is taken when circumstances arise that might cause a dump to be taken.
- Causing a dump to be taken. Both users and CICS can issue requests for dumps to be taken.

Setting up the dumping environment

There are several levels at which the dumping environment can be set up:

- System dumps (apart from CICS kernel domain dumps) can be globally suppressed or enabled at system initialization by using the DUMP system initialization parameter.
- System dumps (apart from CICS kernel domain dumps) can be globally suppressed or enabled dynamically through the EXEC CICS SET SYSTEM DUMPING command.
- Transaction dumps can be suppressed or enabled dynamically for individual transactions. This is done by using the EXEC CICS SET TRANSACTION DUMPING system programming command, or the CEMT SET TRDUMPCODE command or by using the DUMP attribute of the RDO definition for the transaction.
- System dumps (apart from CICS kernel domain dumps) can be suppressed for specific dump codes from a dump domain XDUREQ global user exit program.
- System dumps (apart from CICS kernel domain dumps) can be suppressed or enabled, and other dumping requirements specified, by means of dump codes.

A dump code defines what action CICS is to take under any of the circumstances in which a dump might be required. Dump codes are kept in one of two **dump tables**, one for "transaction dump codes" and the other for "system dump codes". For details, see "The dump code options you can specify" on page 261.

- Dumps of the builder parameter set at specific stages in the build process of terminal or connection definition can be enabled. This is done using the CSFE ZCQTRACE facility. For details, see "The CSFE ZCQTRACE facility" on page 268.

Detecting and avoiding duplicate system dumps

When more than one CICS system runs under one instance of the MVS operating system, two CICS systems can take duplicate system dumps.

Each CICS/ESA 4.1 system dump header includes a symptom string. The symptom string will be created only if the system dump code has the DAE option specified in the dump table entry. The default action is that symptom strings are not produced. This can, however, be altered by means of the DAE= system initialization parameter.

The symptom strings provide sufficient information to enable the detection of duplicate dumps. You can take advantage of this in either of two ways:

1. Use MVS Dump Analysis Elimination (DAE) to detect and suppress duplicate dumps. (If the symptom string has been suppressed by the dump table option, DAE will not suppress the system dump.)

   You can control DAE with an ADYSETxx parmlib member. Additionally, be aware of the SUPPRESS and SUPPRESSALL options in the ADYSETxx parmlib member. These are controlled by the VRADAE and VRANODAE keys in the SDWA. For information about DAE, SUPPRESS, and SUPPRESSALL, see *MVS ESA 4.1 Planning: Problem Determination and Recovery*, GC28-1629.

2. Manually compare the headers of CICS/ESA 4.1 system dumps, so that you are aware that you have duplicate dumps. Doing it this way, you avoid repeating the same analysis, but still have a separate dump listing for each CICS system.

Events that can cause dumps to be taken

The following are the events that can cause dumps to be taken, if the dumping environment allows dumping under the circumstances:

- Explicit requests for dumps from users
- CICS transaction abends
- CICS system abends.

On most occasions when dumps are requested, CICS references a dump code that is specified either implicitly or explicitly to determine what action should be taken. Dump codes are held in two dump tables, the transaction dump table and the system dump table.

The ways that you can request dumps

You can issue an explicit request for a dump by using the CEMT transaction, by using an EXEC CICS command, or by using an exit programming interface (XPI) call.

- CEMT PERFORM [DUMP|SNAP] enables you to get a CICS system dump from the master terminal, if system dumping has not been globally suppressed. The system dump code that is needed is supplied by CICS, and it has a specific value of “MT0001”.
- You can use EXEC CICS PERFORM DUMP to get a CICS system dump, if system dumping is not globally suppressed. You must specify a system dump code when you use this command and it must be a maximum of 8 characters in length.
- You can use EXEC CICS DUMP TRANSACTION to get a transaction dump. You get a transaction dump even if dumping has been suppressed for the transaction that you identify on the command.

  You must specify a transaction dump code when you use this command and it must be a maximum of 4 characters in length. It could, for example, be TD01.
- You can make a TRANSACTION_DUMP or a SYSTEM_DUMP XPI call from an exit program to get a transaction dump or a system dump, respectively. For programming information about these exits, see the *CICS/ESA Customization Guide*.

You might use these methods of taking dumps if, for example, you had a task in a wait state, or you suspected that a task was looping. However, these methods are not useful for getting information following a transaction abend or a CICS system abend. This is because the evidence you need is almost certain to disappear before your request for the dump has been processed.

The occasions when CICS requests a dump

In general, CICS requests a dump when a transaction or CICS system abend occurs. The dumping environment determines whether or not a dump is actually taken. CICS does not take a transaction dump if a HANDLE ABEND is active at the current logical level[4]. The NODUMP option on an EXECS CICS ABEND command, an internal call, or the transaction definition, prevents the taking of a transaction dump.

The following are the occasions when CICS requests a dump:

- CICS requests a transaction dump, and perhaps a system dump, after a transaction abend occurs. There are two cases:
  - You can include the command EXEC CICS ABEND in one of your applications, causing it to abend when some condition occurs. You must specify a four-character transaction abend code on this command, and this is used as the transaction dump code. It could, for example, be ‘MYAB’.
  - CICS might cause a transaction to abend, for any of the reasons described in the *CICS/ESA Messages and Codes* manual. In this case, the

[4] To make PL/I on units work, PL/I library routines can issue HANDLE ABEND. This is called an implicit HANDLE ABEND and causes the suppression of transaction dumps.

four-character CICS transaction abend code is used as the transaction dump code. It might, for example, be ASRA.

- A CICS system dump could be taken following a CICS system abend. In this situation, the system dump code is often equal to the abend message ID, with the leading letters “DFH” stripped off. In the case of message DFHST0001, for example, the system dump code would be “ST0001”. However, in some cases the system dump code cannot be directly related to a CICS message, either because it does not closely resemble the message, or because no message accompanies the event causing the dump to be invoked. For details of these system dump codes, see the *CICS/ESA Messages and Codes* manual.
- When a transaction dump or system dump is taken, and the dump code includes the RELATED attribute on the DUMPSCOPE option, system dumps are taken of all CICS regions in the sysplex which are related to the CICS region on which this transaction dump or system dump is taken. A related CICS region is one in which the unit of work identifiers, in the form of APPC tokens, of one or more tasks match those in the CICS region that issued the dump request. Typically, these may be regions in a distributed transaction processing environment.
- A CICS system dump can be requested from within the Node Error Program (NEP) when a terminal error is processed for a terminal with no task attached. For information about using NEPs to handle terminal errors, read the entry for message DFHZC3496 in the *CICS/ESA Messages and Codes* manual; also read the programming information on NEPs in the *CICS/ESA Customization Guide*.
- A CICS system dump can also be requested from the global trap/trace exit. In this case, the system dump code is TR1003.

CICS dumping in a sysplex

Simultaneous dump capture from multiple CICS regions across a sysplex is possible, considerably aiding problem determination in XCF/MRO environments where many CICS regions are running. Two types of situation, in particular, benefit from this:

- Where a task involves multiple CICS regions in a sysplex, and one region issues a dump, typically in response to an error.

  Dump data from all CICS regions related to the region issuing the dump request is usually required in order to fully diagnose and solve the problem. This dump data from related regions would also need to be captured at the same time as the dump taken on the region issuing the dump.
- Where an MVS console operator needs to capture, simultaneously, dump data from multiple CICS regions in the sysplex.

Before CICS/ESA 4.1, such automatic and simultaneous dump data capture of CICS data was impossible.

You need MVS/ESA 5.1, the MVS workload manager, and the XCF facility in order to collect dump data in this way. The MVS images in the sysplex must be connected via XCF. The CICS regions must be using MRO supported by the CICS/ESA 4.1 interregion communication program, DFHIRP.

Automatic dump data capture from related CICS regions

It is possible to collect dump data simultaneously from all related CICS regions in a sysplex. Related CICS regions are those containing one or more tasks which have unit of work identifiers, in the form of APPC tokens, that match the unit of work identifiers in the CICS region which initially issued the dump request.

The CICS regions must be connected via XCF/MRO. Connections using VTAM ISC are not eligible to use the related dump facility.

The function is controlled by the DUMPSCOPE option on each CICS dump table entry. You can set this option to have either of the following values:

- RELATED - take dumps for all related CICS regions across the sysplex.
- LOCAL - take dumps for the requesting CICS region only. This is the default.

The DUMPSCOPE option is available on the following master terminal and system programming commands:

- EXEC CICS INQUIRE SYSDUMPCODE
- EXEC CICS SET SYSDUMPCODE
- EXEC CICS INQUIRE TRANDUMPCODE
- EXEC CICS SET TRANDUMPCODE
- CEMT INQUIRE SYDUMPCODE
- CEMT SET SYDUMPCODE
- CEMT INQUIRE TRDUMPCODE
- CEMT SET TRDUMPCODE

If the DUMPSCOPE option is set to RELATED in the CICS region issuing the dump request, a request for a system dump is sent to all MVS images in the sysplex that run related CICS regions.

The local MVS image running the CICS region that initiated the dump request has two dumps - one of the originating CICS region, the other containing the originating CICS region and up to fourteen additional related CICS regions from the local MVS image.

When a dump is requested, the DUMPSCOPE option is tested only in the CICS region issuing the original dump request. If the requesting CICS region has DUMPSCOPE defined as RELATED for the dump code, then all related CICS regions are dumped even if they have DUMPSCOPE defined as LOCAL for the dump code.

There is a maximum of fifteen address spaces in an SDUMP. If there are more than fifteen related CICS regions on an MVS image, then not all of them will be dumped. Related CICS regions may also fail to be dumped if they are swapped out when the dump request is issued. You should consider whether to make certain CICS regions non-swappable as a result.

Operator-requested simultaneous dump data capture

The MVS console operator may want to issue a dump request simultaneously to several CICS regions in the sysplex. There may be a problem in the sysplex where one or more CICS regions is hanging and, to fully diagnose and solve the problem, dump data captured at the same time is required.

Without this facility, such simultaneous dump data capture across multiple CICS regions in the sysplex is impossible.

Use the following command at the console:

```
DUMP COMM=()
R x,REMOTE=(SYSLIST=*),PROBDESC=(SYSDCOND,SYSDLOCL,(DFHJOBN,jobnames))
```

where:

- **REMOTE** controls the issuing of dumps on remote systems.
- **SYSLIST=*** means the request is to be routed to all remote systems.
- **PROBDESC** is problem description information, as follows:
  - SYSDCOND - an MVS keyword. This specifies that a dump is to be taken on remote MVS images if the IEASDUMP.QUERY exit responds with return code 0. CICS supplies DFHDUMPX as the IEASDUMP.QUERY exit.

- SYSDLOCL - an MVS keyword. This drives the IEASDUMP.QUERY exit on the local and remote MVS images. This allows the CICS regions on the local MVS region to be dumped.
- DFHJOBN - a CICS keyword. The operator should include the generic job name. This is used by DFHDUMPX to determine which address spaces to dump.

Refer to the *MVS System Commands* manual, GC28 1626, for a full description of all command options.

If you adopt a suitable naming convention for your CICS regions, this can be used to define suitable generic jobnames to determine which CICS regions to dump. See the *System/390 MVS Sysplex Application Migration* manual for recommendations on naming conventions. If you follow the recommendation in this manual, the generic job name for all CICS regions in the sysplex would be 'CICS*'.

Useful CICS master terminal and MVS console commands in a sysplex

MVS support for remote SDUMPs is available only for images running MVS/ESA 5.1 or later and which are connected by XCF. Related CICS SDUMPs are produced only for CICS regions which are MRO connected using XCF. DFHIRP must be at CICS/ESA 4.1 level. Connections using VTAM ISC are not eligible to use the related dump facility.

If you are unable to produce related system dumps when there are related CICS regions across MVS images, ensure that the regions are MRO connected.

Use CEMT I IRC to ensure that interregion communication is available. If IRC is not available it may be started using the CEMT S IRC OPEN command. Failure to start IRC results in DFHIRxxx messages which may be used to identify the source of the problem.

During IRC start processing, CICS attempts to join XCF group DFHIR000. If this fails, return code yyy is given in the DFHIR3777 message.

The following MVS console commands may be used to monitor activity in the sysplex:

- `D XCF` - to identify the MVS sysplex and list the name of each MVS image in the sysplex.

  An example of a response to this command looks like this:

```
08.14.16 DEV5           d xcf
08.14.16 DEV5           IXC334I  08.14.16  DISPLAY XCF 602
   SYSPLEX DEVPLEX5:   DEV5
```

  This response tells you that MVS image DEV5 has joined sysplex DEVPLEX5.

- `D XCF,GROUP` - to list active XCF groups by name and size (note that CICS, group DFHIR000, is missing from the following example response).

```
16.21.36 DEV5           d xcf,group
16.21.36 DEV5           IXC331I  16.21.36  DISPLAY XCF 877
   GROUPS(SIZE):   COFVLFNO(1)       SYSDAE(2)         SYSGRS(1)
                   SYSIGW00(1)       SYSIGW01(1)       SYSIKJBC(1)
                   SYSMCS(6)         SYSMCS2(3)        SYSWLM(1)
                   WINMVSG(1)
```

- `D XCF,COUPLE` - to list details about the XCF coupling dataset and its definitions. In the following example response, the dataset has a MAXGROUP of 10 and a peak of 10. The response to XCF,GROUP indicates there are currently 10 active groups. Should CICS now attempt to join XCF group DFHIR000, it would be rejected and the IRC start would fail with message DFHIR3777.

```
16.30.45 DEV5           d xcf,couple
16.30.45 DEV5           IXC357I  16.30.45  DISPLAY XCF 883
SYSTEM DEV5 DATA
   INTERVAL   OPNOTIFY     MAXMSG    CLEANUP      RETRY   CLASSLEN
         42         45        500         60         10        956

   SSUM ACTION      SSUM INTERVAL             WEIGHT
            N/A                N/A               N/A
SYSPLEX COUPLE DATA SETS
PRIMARY    DSN: SYS1.XCFDEV5.PXCF
           VOLSER: SYS001     DEVN: 0F0E
           FORMAT TOD         MAXSYSTEM MAXGROUP(PEAK) MAXMEMBER(PEAK)
           08/16/93 08:05:39          8      10   (10)       87    (6)
ALTERNATE  DSN: SYS1.XCFDEV5.AXCF
           VOLSER: SYS001     DEVN: 0F0E
           FORMAT TOD         MAXSYSTEM    MAXGROUP        MAXMEMBER
           08/16/93 08:05:41          8       10              87
```

  This example also indicates that the primary and alternate datasets are on the same volume, thereby giving rise to a single point of failure.

Use the CICS master terminal command CEMT I CONNECTION to display the status of the connections. 'XCF' is displayed for every acquired connection using MRO/XCF for communications.

```
 I CONNECTION
 STATUS:  RESULTS - OVERTYPE TO MODIFY
  Con(FORD) Net(IYAHZCES)     Ins Acq         Xcf
  Con(F100) Net(IYAHZCEC)     Ins Acq         Irc
  Con(F150) Net(IYAHZCED)     Ins Acq         Irc
  Con(GEO ) Net(IYAHZCEG)     Ins Acq         Xcf
  Con(GMC ) Net(IYAHZCEB)     Ins Acq         Xcf
  Con(JIM ) Net(IYAHZCEJ)     Ins Acq         Xcf
  Con(MARY) Net(IYAHZCEM)     Ins Acq         Xcf
  Con(MIKE) Net(IYAHZCEI)     Ins Acq         Xcf
+ Con(RAMB) Net(IYAHZCEE)     Ins Acq         Xcf
                                                    SYSID=CHEV APPLID=IYAHZCET
  RESPONSE: NORMAL                             TIME:  01.28.59  DATE: 06.11.94
PF 1 HELP       3 END                    7 SBH 8 SFH 9 MSG 10 SB 11 SF
```

CICS initialization issues an MVS CSVDYNEX request to add DFHDUMPX as an MVS IEASDUMP.QUERY exit. If you need to change the status of the exit, use the MVS console command SETPROG EXIT.

If you have determined that XCF communication is in use, you can verify that the CICS SDUMP exit has been established using the following MVS commands:

```
D PROG,EXIT,MODNAME=DFHDUMPX
    08.16.04 DEV5          CSV463I MODULE DFHDUMPX IS NOT ASSOCIATED ANY EXIT

D PROG,EXIT,EN=IEASDUMP.QUERY
   08.17.44 DEV5          CSV463I NO MODULES ARE ASSOCIATED WITH EXIT IEASDUMP.QUERY
```

This example indicates that the exit has not been established.

The following displays indicate that DFHDUMPX is active as an IEASDUMP exit with one CICS/ESA 4.1 region active in one MVS image.

```
D PROG,EXIT,MODNAME=DFHDUMPX
01.19.16 DEV5           CSV461I 01.19.16 PROG,EXIT DISPLAY 993
EXIT             MODULE   STATE MODULE   STATE MODULE   STATE
IEASDUMP.QUERY   DFHDUMPX   A

D PROG,EXIT,EN=IEASDUMP.QUERY
01.19.46 DEV5           CSV462I 01.19.46 PROG,EXIT DISPLAY 996
MODULE  DFHDUMPX
EXIT(S) IEASDUMP.QUERY
```

You may issue MVS dump commands from the console to verify that remote dumping is available within the MVS image, without an active CICS region.

```
11.29.59 DEV5           dump comm=(NO CICS)
11.29.59 DEV5          *03 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
11.36.49 DEV5           r 03,remote=(syslist=*),probdesc=(sysdcond,
 sysdlocl,(dfhjobn,iyahzcet))
11.36.49 DEV5           IEE600I REPLY TO 03 IS;REMOTE=(SYSLIST=*),
 PROBDESC=(SYSDCOND,SYSDL
11.36.52 DEV5           IEA794I SVC DUMP HAS CAPTURED:
DUMPID=001 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=NO CICS
```

In the next example, the messages from SDUMP indicate that one dump of the master address space has been taken.

```
*11.37.03 DEV5          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
*DUMPID=001 REQUESTED BY JOB (*MASTER*)
*FOR ASID (0001)
*REMOTE DUMPS REQUESTED
*INCIDENT TOKEN: DEVPLEX5 DEV5     06/28/1994 11:36:49
```

Another test is to issue the dump command specifying the CICS XCF group.

```
11.42.33 DEV5           dump comm=(STILL NO CICS)
11.42.33 DEV5          *05 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
11.43.27 DEV5           r 05,remote=(grplist=dfhir000(*)),
 probdesc=(sysdcond,sysdlocl,(dfhjobn,iyahzcet))
11.43.28 DEV5           IEE600I REPLY TO 05
 IS;REMOTE=(GRPLIST=DFHIR000(*)),PROBDESC=(SYSD
11.43.31 DEV5           IEA794I SVC DUMP HAS CAPTURED:
DUMPID=002 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=STILL NO CICS
```

The messages from SDUMP indicate that one dump of the master address space has been taken.

```
*11.43.42 DEV5          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
*DUMPID=002 REQUESTED BY JOB (*MASTER*)
*FOR ASID (0001)
*REMOTE DUMPS REQUESTED
*INCIDENT TOKEN: DEVPLEX5 DEV5     06/28/1994 11:43:28
```

To verify that the remote dumping function works on the local system, use the following commands:

```
11.45.57 DEV5           dump comm=(TEST REMOTE FUNCTION ON LOCAL SYSTEM
11.45.57 DEV5          *06 IEE094D SPECIFY OPERAND(S) FOR DUMP COMMAND
11.46.57 DEV5           r 06,remote=(grplist=*(*)),probdesc=(sysdlocl)
11.46.59 DEV5           IEE600I REPLY TO 06 IS;REMOTE=(GRPLIST=*(*)),
 PROBDESC=(SYSDLOCL)
11.47.00 DEV5           IEA794I SVC DUMP HAS CAPTURED:
DUMPID=003 REQUESTED BY JOB (*MASTER*)
DUMP TITLE=TEST REMOTE FUNCTION ON LOCAL SYSTEM
11.47.17 DEV5           IEA794I SVC DUMP HAS CAPTURED:
DUMPID=004 REQUESTED BY JOB (DUMPSRV )
DUMP TITLE=TEST REMOTE FUNCTION ON LOCAL SYSTEM
```

The messages from SDUMP indicate two dumps were taken, one for the master address space and a second which contains ASIDs 0101, 0012, 0001, 0005, 000B, 000A, 0008, 0007. Note that the same incident token is used for both dumps.

```
*11.47.39 DEV5          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
*DUMPID=003 REQUESTED BY JOB (*MASTER*)
*FOR ASID (0001)
*REMOTE DUMPS REQUESTED
*INCIDENT TOKEN: DEVPLEX5 DEV5     06/28/1994 11:46:57
*11.47.59 DEV5          *IEA911E COMPLETE DUMP ON SYS1.DUMP04
*DUMPID=004 REQUESTED BY JOB (DUMPSRV )
*FOR ASIDS(0101,0012,0005,0001,000A,000B,0008,0007)
*REMOTE DUMP FOR SYSNAME: DEV5
*INCIDENT TOKEN: DEVPLEX5 DEV5     06/28/1994 11:46:57
```

The following example lists the MVS console messages received when the CICS master terminal command CEMT P DUMP is issued from CICS APPLID IYAHZCET executing in ASID 19 on MVS image DEV6. IYAHZCET has at least one related task in the CICS region executing in ASID 1B on MVS DEV6 and ASIDS 001A, 001C, 001B, 001E, 001F, 0020, 001D, 0022, 0024, 0021, 0023, 0028, 0025, 0029, 0026 on MVS image DEV7.

```
- 22.19.16 DEV6 JOB00029  +DFHDU0201 IYAHZCET ABOUT TO TAKE SDUMP. DUMPCODE: MT0001
- 22.19.23 DEV7           DFHDU0214 DFHDUMPX IS ABOUT TO REQUEST A REMOTE SDUMPX.
- 22.19.23 DEV6           DFHDU0214 DFHDUMPX IS ABOUT TO REQUEST A REMOTE SDUMPX.
  22.19.27 DEV6 JOB00029  IEA794I SVC DUMP HAS CAPTURED:
  DUMPID=001 REQUESTED BY JOB (IYAHZCET)
  DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCET CODE=MT0001   ID=1/0001

  22.19.43 DEV6 JOB00029  IEA794I SVC DUMP HAS CAPTURED:
  DUMPID=002 REQUESTED BY JOB (DUMPSRV )
  DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCET CODE=MT0001   ID=1/0001

- 22.19.43 DEV6 JOB00029  +DFHDU0202 IYAHZCET SDUMPX COMPLETE. SDUMPX RETURN CODE X'00'

 *22.21.00 DEV6          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
 *DUMPID=001 REQUESTED BY JOB (IYAHZCET)
 *FOR ASID (0019)
 *REMOTE DUMPS REQUESTED
 *INCIDENT TOKEN: DEVPLEX1 DEV6     06/10/1994 22:19:16
 *ID = DUMP : APPLID IYAHZCET DUMPCODE MT0001  /1/0001
```

The dump in SYS1.DUMP03 on DEV6 was taken as a result of the CEMT request on IYAHZCET.

```
*22.21.15 DEV6          *IEA911E COMPLETE DUMP ON SYS1.DUMP04
*DUMPID=002 REQUESTED BY JOB (DUMPSRV )
*FOR ASIDS(0019,001B)
*REMOTE DUMP FOR SYSNAME: DEV6
*INCIDENT TOKEN: DEVPLEX1 DEV6     06/10/1994 22:19:16
*ID = DUMP : APPLID IYAHZCET DUMPCODE MT0001  /1/0001
```

The dump in SYS1.DUMP04 on DEV6 was taken as a remote dump by MVS dump services as a result of the CEMT request on IYAHZCET. Note that the incident token and ID are the same.

```
 22.22.35 DEV7 JOB00088  IEA794I SVC DUMP HAS CAPTURED:
 DUMPID=003 REQUESTED BY JOB (DUMPSRV )
 DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCET CODE=MT0001   ID=1/0001
```

```
*22.25.58 DEV7          *IEA911E COMPLETE DUMP ON SYS1.DUMP05
*DUMPID=003 REQUESTED BY JOB (DUMPSRV )
*FOR ASIDS(001A,001C,001B,001E,001F,0020,001D,0022,0024,0021,0023,0028,
*0025,0029,0026)
*REMOTE DUMP FOR SYSNAME: DEV6
*INCIDENT TOKEN: DEVPLEX1 DEV6     06/10/1994 22:19:16
*ID = DUMP : APPLID IYAHZCET DUMPCODE MT0001  /1/0001
```

The dump in SYS1.DUMP05 on DEV7 was taken as a remote dump by MVS dump services as a result of the CEMT request on IYAHZCET. Note that the incident token and ID are the same as those for the dumps produced on DEV6, indicating the originating MVS and CICS IDs.

The following example lists the MVS console messages received when transaction abend SCOP is initiated after having first been added to the transaction dump table in CICS IYAHZCES as requiring related dumps. (CEMT S TRD(SCOP) ADD RELATE).

CICS IYAHZCES (ASID 1A in MVS DEV7) has at least one related task in CICS IYAHZCET (ASID 19 in MVS DEV6).

```
 23.40.41 DEV7 JOB00088  +DFHDU0201 IYAHZCES ABOUT TO TAKE SDUMP. DUMPCODE: SCOP
 23.40.49 DEV7           DFHDU0214 DFHDUMPX IS ABOUT TO REQUEST A REMOTE SDUMPX.
 23.40.55 DEV7 JOB00088  IEA794I SVC DUMP HAS CAPTURED:

 DUMPID=012 REQUESTED BY JOB (IYAHZCES)
 DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCES CODE=SCOP     ID=1/0008

 23.40.49 DEV6           DFHDU0214 DFHDUMPX IS ABOUT TO REQUEST A REMOTE SDUMPX.
 23.40.56 DEV6 JOB00029  IEA794I SVC DUMP HAS CAPTURED:
 DUMPID=007 REQUESTED BY JOB (DUMPSRV )
 DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCES CODE=SCOP     ID=1/0008


 23.41.11 DEV7 JOB00088  IEA794I SVC DUMP HAS CAPTURED:
 DUMPID=013 REQUESTED BY JOB (DUMPSRV )
 DUMP TITLE=CICS DUMP: SYSTEM=IYAHZCES CODE=SCOP     ID=1/0008

 23.41.11 DEV7 JOB00088  +DFHDU0202 IYAHZCES SDUMPX COMPLETE. SDUMPX RETURN CODE X'00'
```

```
*23.41.18 DEV6          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
*DUMPID=007 REQUESTED BY JOB (DUMPSRV )
*FOR ASID (0019)
*REMOTE DUMP FOR SYSNAME: DEV7
*INCIDENT TOKEN: DEVPLEX1 DEV7     06/10/1994 23:40:41
*ID = DUMP : APPLID IYAHZCES DUMPCODE SCOP    /1/0008
```

The dump in SYS1.DUMP03 on DEV6 was taken upon receipt of the remote dump request issued from IYAHZCES. Note the incident token and ID are the same as those for dumps produced on DEV7.

```
*23.41.28 DEV7          *IEA911E COMPLETE DUMP ON SYS1.DUMP03
*DUMPID=012 REQUESTED BY JOB (IYAHZCES)
*FOR ASID (001A)
*REMOTE DUMPS REQUESTED
*INCIDENT TOKEN: DEVPLEX1 DEV7     06/10/1994 23:40:41
*ID = DUMP : APPLID IYAHZCES DUMPCODE SCOP    /1/0008

*23.41.38 DEV7          *IEA911E COMPLETE DUMP ON SYS1.DUMP04
*DUMPID=013 REQUESTED BY JOB (DUMPSRV )
*FOR ASID (001A)
*REMOTE DUMP FOR SYSNAME: DEV7
*INCIDENT TOKEN: DEVPLEX1 DEV7     06/10/1994 23:40:41
*ID = DUMP : APPLID IYAHZCES DUMPCODE SCOP    /1/0008
```

The dump in SYS1.DUMP04 on DEV7 was taken as a remote dump by MVS dump services as a result of the request from IYAHZCES. Note the incident token and ID are the same as those for the dumps produced on DEV6, indicating the originating MVS and CICS IDs. A second dump of ASID 1A is taken because the CICS IEASDUMP does not have information indicating that a dump has already been taken for that address space.

Enabling system dumps for some CICS messages

There are occasions when you might need diagnostic information for an event that causes a message to be sent, but does not normally cause CICS to take a system dump. You can enable system dumping for some CICS messages that do not normally cause CICS to take a system dump. You do this by adding the dump code (constructed by removing the “DFH” prefix from the message number) to the system dump table, and specifying the SYSDUMP option. CICS then causes a system dump to be taken when the message is issued.

To determine which messages you can do this for, look in the *CICS/ESA Messages and Codes* manual. If the message you are interested in has a 2-character alphabetic component ID after the “DFH” prefix, and it has **either** XMEOUT global user exit parameters or a destination of “Terminal User”, you can use it to construct a system dump code to add to the dump table.

You cannot enable dumping for messages that do not have these characteristics. For example, some messages that are issued early during initialization cannot be used to cause CICS to take a system dump, because the mechanisms that control dumping might not be initialized at that time. Also, you cannot enable dumping for the message domain's own messages (they are prefixed by “DFHME”) where they do not normally cause CICS to take a system dump.

System dump actions associated with messages DFHAP0001 and DFHSR0001

In the event of a program check or MVS abend in the AP domain or in a user application program, CICS may issue either message DFHAP0001 or DFHSR0001. Message DFHSR0001 is issued by CICS only when storage protection is active; that is, the system initialization parameter STGPROT=YES is specified and the hardware and software environment supports the storage protection facility. CICS

determines which of these messages to issue depending on whether or not the program check or MVS abend occurs in code running in user key.

If the code had been running in user key at the time of the program check or MVS abend, CICS issues message DFHSR0001 and takes a system dump with dump code SR0001. Only application programs defined with EXECKEY(USER) run in user key.

If the code had not been running in user key at the time of the program check or MVS abend, CICS issues message DFHAP0001 and takes a system dump with dump code AP0001.

So, if CICS storage protection is active, this mechanism enables you to suppress the system dumps caused by errors in application programs, while still allowing dumps caused by errors in CICS code to be taken. To achieve this, use either a CEMT SET SYDUMPCODE or an EXEC CICS SET SYSDUMPCODE command to suppress system dumps for system dumpcode SR0001:

```
CEMT SET SYDUMPCODE(SR0001)  ADD  NOSYSDUMP
```

If storage protection is not active, the dumps may be suppressed via a suppression of dumpcode AP0001. Note, however, that this suppresses dumps for errors in both application **and** CICS code.

You cannot control AP0001 and SR0001 system dumps by using the DUMP parameter of the TRANSACTION resource definition. The DUMP parameter of the TRANSACTION resource definition controls only transaction dumps.

Usually, program checks, or MVS abends caused by an application program, are also followed by an ASRA, ASRB or ASRD transaction abend and a transaction dump. If, in some instances, you want the SDUMP for one of these transaction abends but not the other, specify the one you want by using either a CEMT TRDUMPCODE or an EXEC CICS TRANDUMPCODE command. For example, specifying:

```
CEMT SET TRDUMPCODE(ASRB)  ADD  SYSDUMP
```

adds an entry to the dump table and ensures that SDUMPs are taken for ASRB abends. However, note that the SDUMP in this instance is taken at a later point than the SDUMP normally taken for system dump code AP0001 or SR0001.

The dump code options you can specify

You can specify what dump action is to be taken by CICS for each individual dump code, either by using a CEMT transaction or by using a system programming command. The options you can specify differ slightly, depending on whether you are defining the action for a transaction dump code or for a system dump code.

- For a **transaction dump code**, you can specify:
  - Whether a transaction dump is to be taken.
  - Whether a system dump is to be taken, with or without a transaction dump.
  - Whether a system dump is to be taken on every CICS region in the sysplex related to the CICS region on which the transaction dump is taken. A related CICS region is one on which the unit of work identifiers, in the form of APPC tokens, of one or more tasks match those in the CICS region that takes the transaction dump.

  - Whether CICS is to be terminated.
  - The maximum number of times the transaction dump code action can be taken during the current run of CICS, or before the count is reset.
- For a **system dump code**, you can specify:
  - Whether a system dump is to be taken.
  - Whether a system dump is to be taken on every CICS region in the sysplex related to the CICS region on which the system dump is taken. A related CICS region is one on which the unit of work identifiers, in the form of APPC tokens, of one or more tasks match those in the CICS region that takes the system dump.
  - Whether CICS is to be terminated.
  - The maximum number of times the system dump code action can be taken during the current run of CICS, or before the count is reset.
  - Whether the system dump is eligible for suppression by DAE.

Notes:

1. Only a *transaction* dump code can cause both a transaction dump and a system dump to be taken.
2. If a severe error is detected, the system can terminate CICS even if you specify that CICS is not to be terminated.
3. Values of 0–998 for “maximum times dump code action can be taken” are literal, but a value of 999 (the default) means there is no limit.
4. You cannot suppress CICS kernel domain dumps.

All the options you specify are recorded in the appropriate dump table, and written in the CICS global catalog. Any dump table entries you have changed or created during a CICS run are preserved when CICS is subsequently shut down. They are available during the next run unless CICS is cold started, when they are deleted from the global catalog.

The only circumstances in which dump table additions and changes are lost are:

- When CICS is cold started.
- When the CICS global catalog is redefined, although this is likely to be done only in exceptional circumstances.
- When CICS creates a temporary dump table entry for you, because you have asked for a dump for which there is no dump code in the dump table.

Dump table statistics

CICS maintains the following statistics for each dump table entry:

- The number of times the dump code action has been taken. This count is referred to as the “dumps taken”. Its value is incremented every time the dump code action is taken, irrespective of what action has been specified in the dump table entry. The current count can be reset to zero by a CEMT transaction, by a system programming command, or by statistical interval expiry.

  If system dumping is globally suppressed:

  - The dumps-taken count for a system dump code is not incremented by dump requests using that dump code, but the dumps-suppressed count is incremented.
  - The dumps-taken count for a transaction dump code specifying a system dump is incremented by dump requests using that dump code. This is so even if the dump code specifies that *only* a system dump is to be taken.

- For a transaction dump code, the number of transaction dumps that have been taken. The number is incremented every time a transaction dump is taken for this dump code. However, it is not incremented if transaction dumping has been suppressed in this table entry.
- For a transaction dump code, the number of transaction dumps that have been suppressed.
- The number of system dumps that have been taken. The number is incremented every time a system dump is taken for this dump code. It is not incremented if system dumping has been suppressed, either explicitly in the dump table entry or because system dumping has been suppressed globally.
- The number of system dumps that have been suppressed, either explicitly for this dump code or because system dumping has been suppressed globally.

Notes:

1. Dump code statistics are all reset when CICS is shut down—unlike dump code attributes, which are not reset.
2. The following dump code statistics are reset at the end of every statistics collecting interval:
   - Number of transaction dumps taken
   - Number of transaction dumps suppressed
   - Number of system dumps taken
   - Number of system dumps suppressed.

The transaction dump table

Table 32 shows some examples of the sort of information that might be maintained in the transaction dump table for different transaction dump codes.

Table 32 (Page 1 of 2). Examples of transaction dump table entries

Type of information	**Example 1**	**Example 2**	**Example 3**
Transaction dump code	MYAB	ASRA	AKC3
Take a transaction dump?	YES	NO	NO
Take a system dump?	YES	YES	NO
# Take system dumps on related systems?	NO	YES	NO
Shut down CICS?	NO	NO	NO
Maximum times dump code action can be taken	50	30	999
Times dump code action already taken (current count)	0	30	37
Transaction dumps taken	0	0	0
Transaction dumps suppressed	0	30	37

Table 32 (Page 2 of 2). Examples of transaction dump table entries			
Type of information	**Example 1**	**Example 2**	**Example 3**
System dumps taken	0	30	0
System dumps suppressed	0	0	37

- **Example 1** shows a transaction dump table entry for transaction dump code MYAB. This is a user-supplied dump code, specified either on an EXEC CICS DUMP TRANSACTION command, or as a transaction abend code on an EXEC CICS ABEND command.

  The table entry shows that when this dump code is invoked, both a transaction dump and a system dump are to be taken, and CICS is not to be terminated. System dumps on related systems are not to be taken. The dump code action can be taken a maximum of 50 times, but the action for this dump code has not been taken since CICS was started or since the current count ("times dump action taken") was reset.

- **Example 2** shows a transaction dump table entry for transaction dump code ASRA. This is a CICS transaction abend code, and this dump table entry is referred to every time a transaction abends ASRA. The entry shows that a system dump only is to be taken for an ASRA abend, and that CICS is not to be terminated. System dumps on related systems are to be taken. It also shows that the action for this abend code has already been taken the maximum number of times, so no action is taken when another ASRA abend occur. However, the current count could be reset to 0 dynamically using either a CEMT transaction or a system programming command (SET TRANDUMPCODE or SET SYSDUMPCODE). More system dumps would then be taken for subsequent ASRA abends.

- **Example 3** shows a transaction dump table entry for transaction dump code AKC3. This is a CICS transaction abend, and this dump table entry is referenced every time a transaction abends AKC3—that is, whenever the master terminal operator purges a task.

  The entry shows that no action at all is to be taken in the event of such an abend. System dumps on related systems are not to be taken. The maximum number of times the dump code action can be taken is given as 999, meaning that there is no limit to the number of times the specified action is taken. The dump code action has been taken 37 times, but each time both the transaction dump and the system dump were suppressed.

Table 33 shows how the transaction dump table entry for transaction dump code MYAB would be updated with and without global suppression of system dumping. Only the updated fields are shown.

Table 33 (Page 1 of 2). Effect of global suppression of system dumping on transaction dump table update			
Type of information	**Before update**	**System dumping enabled**	**System dumping suppressed**
Transaction dump code	MYAB		
Take a transaction dump?	YES		

Table 33 (Page 2 of 2). Effect of global suppression of system dumping on transaction dump table update

Type of information	Before update	System dumping enabled	System dumping suppressed
Take a system dump?	YES		
Take system dumps on related systems?	NO		
Shut down CICS?	NO		
Maximum times action can be taken	50		
Times action already taken	0	1	1
Transaction dumps taken	0	1	1
Transaction dumps suppressed	0	0	0
System dumps taken	0	1	0
System dumps suppressed	0	0	1

The statistics show that a system dump was taken when system dumping was enabled, but not when system dumping was suppressed.

There is a further effect. CICS maintains a record of the **current dump ID**—the number of the most recent dump to be taken. This is printed at the start of the dump, together with the appropriate dump code. It is concatenated with the CICS run number—that is, the number of times that CICS has been brought up since the global catalog was created—to provide a unique ID for the dump.

Note: This does not apply to SDUMPs taken by the kernel; these always have a dump ID of 0/0000.

For example, for the ninth dump to be taken during the eleventh run of CICS, if the dump code were TD01, this is what you would see:

```
CODE=TD01 ID=11/0009
```

If system dumping is enabled for the dump code, the same dump ID is given to both the transaction dump and the system dump.

The system dump table

Table 34 shows two examples of the sort of information that might be maintained in the system dump table for different system dump codes.

Table 34 (Page 1 of 2). Examples of system dump table entries

Type of information	Example 1	Example 2
System dump code	SYDMP001	MT0001
Take a system dump?	YES	YES
Take system dumps on related systems?	YES	NO
Is the system dump eligible for DAE?	YES	NO
Shut down CICS?	YES	NO

Table 34 (Page 2 of 2). Examples of system dump table entries		
Type of information	**Example 1**	**Example 2**
Maximum times action can be taken	(default)	999
Times action already taken	0	79
System dumps taken	0	79
System dumps suppressed	0	0

The sort of information kept in the system dump table is similar to that kept in the transaction dump table (see Table 32 on page 263).

- **Example 1** shows a system dump table entry for system dump code SYDMP001, a user-supplied system dump code, specified using EXEC CICS PERFORM DUMP. System dumps on related systems are to be taken. Dumps duplicate of this one are to be suppressed by DAE. The table entry shows that no dumps have yet been taken. However, if one were taken, CICS would be shut down. If global suppression of system dumping was in effect, no dump would be taken but CICS would be shut down if this dump code were referenced.
- **Example 2** shows the system dump table entry for system dump code MT0001, the CICS-supplied dump code for system dumps requested from the master terminal, with CEMT PERFORM DUMP or CEMT PERFORM SNAP. CICS is not shut down when a dump is taken for this dump code. Also, the value of 999 for "maximum times action can be taken" shows that an unlimited number of dumps can be taken for this dump code. The current count ("times action already taken") shows that to date, 79 dumps have been requested using CEMT.

What happens to a dump request if there is no dump table entry?

If a dump is requested, either by CICS or the user, using a dump code that is not in the dump table, CICS makes a temporary dump table entry using default values for the attributes. However, the entry is not written to the CICS global catalog, and it is lost when CICS is shut down.

The **default** value used for the DAEOPTION attribute (for all new system dump codes) is set by means of the DAE= system initialization parameter. The **default** value for the maximum number of times that the dump action can be taken is set by the TRDUMAX system initialization parameter (for new or added transaction dump codes) and the SYDUMAX system initialization parameter (for new or added system dump codes).

You can modify the default values for a transaction dump table entry using the following commands:

- CEMT SET TRDUMPCODE
- EXEC CICS SET TRANDUMPCODE
- EXEC CICS SET TRANSACTION DUMPING (to modify the TRANDUMPING attribute only).

The following table shows the default values for transaction dump table entries and the attributes you can specify to modify them:

Table 35. Default values for transaction dump table entries			
Action	**Default**	**Attribute**	**Permitted value**
Take a transaction dump?	YES	TRANDUMPING	TRANDUMP or NOTRANDUMP
Take a system dump?	NO	SYSDUMPING	SYSDUMP or NOSYSDUMP
Take system dumps on related systems?	NO	DUMPSCOPE	LOCAL or RELATED
Shut down CICS?	NO	SHUTOPTION	SHUTDOWN or NOSHUTDOWN
Maximum times dump code action can be taken	999	MAXIMUM	0 through 999

You can modify the default values for a system dump table entry using the following commands:

- CEMT SET SYDUMPCODE
- EXEC CICS SET SYSDUMPCODE
- EXEC CICS SET SYSTEM DUMPING (to modify the SYSDUMPING attribute only).

The following table shows the default values for system dump table entries and the attributes you can specify to modify them:

Table 36. Default values for system dump table entries			
Action	**Default**	**Attribute**	**Permitted value**
Take a system dump?	YES	SYSDUMPING	SYSDUMP or NOSYSDUMP
Take system dumps on related systems?	NO	DUMPSCOPE	LOCAL or RELATED
Shut down CICS?	NO	SHUTOPTION	SHUTDOWN or NOSHUTDOWN
Is dump eligible for DAE?	NO	DAEOPTION	DAE or NODAE
Maximum times dump code action can be taken	999	MAXIMUM	0 through 999

For example, if you issue a command requesting a dump, using the previously undefined dump code SYDMPX01:

```
EXEC CICS PERFORM DUMP DUMPCODE('SYDMPX01')
```

CICS makes a temporary dump table entry for dump code SYDMPX01, and you can browse it, and see that it has the default attributes for a system dump code. You can also see that the current count has been set to 1, as a dump has been taken.

Attempting to add the dump code to the dump table after CICS has made the entry causes the exception response 'DUPREC' to be returned. If you want to make a

change to the CICS-generated default table entry, and have that entry preserved across CICS runs, you must delete it and then add a new entry with the options you require.

Specifying the areas you want written to a transaction dump

When you use the EXEC CICS DUMP TRANSACTION command to get a transaction dump, you can specify which areas of storage are to be dumped. You cannot specify in the dump table which areas are to be written to the transaction dump for particular transaction dump codes. You always get a complete transaction dump whenever a transaction abend occurs, if the dump code requires a transaction dump to be taken.

The CSFE ZCQTRACE facility

The CSFE ZCQTRACE facility is used to gather information during the build process of terminal or connection definition. The syntax is as follows:

```
CSFE {ZCQTRACE=termid|ZCQTRACE,AUTOINSTALL|ZCQTRACE,OFF}
```

When CSFE ZCQTRACE is enabled, a dump of the builder parameter set and the appropriate TCTTE is written to the transaction dump data set at specific points in the processing. Table 37 shows the circumstances in which dumps are invoked, the modules that invoke them, and the corresponding dump codes.

Table 37. ZCQTRACE dump codes

Module	Dump code	When invoked
DFHTRZCP	AZCQ	Installing terminal when termid = terminal ID
DFHTRZZP	AZQZ	Merging terminal with TYPETERM when termid = terminal ID
DFHTRZXP	AZQX	Installing connection when termid = connection ID
DFHTRZIP	AZQI	Installing sessions when termid = connection ID
DFHTRZPP	AZQP	When termid = pool terminal ID
DFHZCQIQ	AZQQ	Inquiring on resource when termid = resource ID (resource = terminal or connection)
DFHZCQIS	AZQS	Installing a resource when termid = resource ID (resource = terminal or connection), or when ZCQTRACE,AUTOINSTALL is specified.

If a terminal definition is shipped from a terminal owning region (TOR) to an application owning region (AOR) and ZCQTRACE is enabled for that terminal in the TOR, then DFHZCQIQ invokes a dump in the TOR, and DFHZCQIS invokes a dump in the AOR.

Where dumps are written

Transaction dumps and system dumps are written to different destinations.

- **Transaction dumps** go to a pair of CICS BSAM data sets, with DD names DFHDMPA and DFHDMPB. Some of the attributes of these data sets can be set by system initialization parameters, some can be set dynamically using CEMT SET DUMPDS or EXEC CICS SET DUMPDS, and all can be inquired on by using CEMT INQ DUMPDS or EXEC CICS INQUIRE DUMPDS. DFHDMPA and DFHDMPB have the following attributes:
  - CURRENTDDS status. This tells you the data set that is currently active, which is the one where transaction dumps are currently written.

    You can use the system initialization parameter DUMPDS to specify the transaction dump data set that is to be opened during initialization. You can use the CEMT SET DUMPDS transaction or an EXEC CICS SET command to switch the dump data sets.
  - One of the statuses OPEN or CLOSED. A transaction dump data set must be OPEN if it is to be written to.
  - The current data set has one of the statuses AUTOSWITCH or NOAUTOSWITCH. You can set the status during system initialization using the DUMPSW system initialization parameter, and you can set it dynamically using the CEMT transaction or an EXEC CICS SET command.

    If the status is AUTOSWITCH, a switch is made automatically to the other dump data set when the current one becomes full, and subsequent transaction dumps are written to the new dump data set. The overflowing transaction dump is written in its entirety to the new dump data set.

    The dump data set being switched to does not inherit the AUTOSWITCH status, to prevent data in the first dump data set from being overwritten by another switch. You need to reset the AUTOSWITCH status explicitly, if you want it.

    If the status is NOAUTOSWITCH, a switch is not made when the current dump data set becomes full, so no more transaction dumps can be written.
- **CICS system dumps** are written to MVS SYS1.DUMPxx data sets, where “xx” has a value of 00 through 99, by an invocation of the MVS SDUMP macro. CICS can write only one system dump to any individual SYS1.DUMPxx data set.

  If another address space is already taking an SDUMP when CICS issues the SDUMP macro, the request fails but is automatically retried. You can use the DURETRY system initialization parameter to define the total time that CICS is to continue trying to take an SDUMP.

  If CICS tries to take an SDUMP when all the MVS SYS1.DUMPxx data sets are full, the dump is lost. Because of this, it is advisable to monitor the number of full data sets, and to take copies of dumps before processing them, so that you can free the SYS1.DUMPxx data sets for reuse.

Looking at dumps

CICS system dumps and transaction dumps are written unformatted to the appropriate dump data set. In other words, they are memory dumps of all or part of the CICS address space.

Unformatted dumps are not easy to interpret, and you are recommended not to use them for debugging. CICS provides utilities for formatting transaction dumps and CICS system dumps, and you should always use them before you attempt to read any dump. You can quickly locate areas of storage that interest you in a formatted dump, either by browsing it online, or by printing it and looking at the hard copy.

The formatting options that are available for transaction dumps and system dumps are described in “Formatting transaction dumps” and “Formatting system dumps” on page 277, respectively.

Formatting transaction dumps

You can format transaction dumps offline using the CICS dump utility program, DFHDU410. Individual transaction dumps must be formatted in their entirety, but you can control which dumps are formatted from the dump data set. You can select dumps to be formatted as follows:

- Those taken in a specific period of time
- Those associated with a specific transaction identifier
- Those taken for a specific transaction dump code
- Those having specific dump IDs—that is, those for specific CICS runs and dump count values.

You can also use the SCAN option with the dump utility program, to get a list of the transaction dumps recorded on the specified dump data set.

For information about using DFHDU410 to format transaction dumps, see the *CICS/ESA Operations and Utilities Guide*.

Interpreting transaction dumps

This section describes the contents of a transaction dump, and gives guidance on locating information that is useful for debugging transactions. The different parts of the transaction dump are dealt with in the order in which they appear, but be aware that only those parts that users should be using for problem determination are described. Some control blocks which do appear in the transaction dump are intended for the problem determination purposes of IBM Service and, as such, are not described in this section.

Job log

The job log for the dump utility program, DFHDU410, is sometimes shown at the start of the transaction dump, depending on how the job was invoked. You can ignore it, because it doesn’t contain any information that can be used for debugging.

Symptom string

The symptom string tells you something about the circumstances of the transaction dump. It might show, for example, that the dump was taken because the transaction abended with abend code ASRA.

If you refer the problem that caused the dump to be taken to the IBM Support Center, they can use the symptom string to search the RETAIN database for problems that resemble it.

CICS/ESA level

The CICS/ESA level shows you what level of CICS/ESA was being executed when the transaction dump was taken.

Transaction environment summary

The transaction environment summary is a formatted summary of the transaction environment at the time the dump is taken.

Remote abend indicator

If the transaction abended remotely (the abend originally occurred in a remote distributed program link (DPL) server program), and the abend is being re-issued on the local system, a message indicates this. The message contains the SYSID of the system that passed the abend to the local system. This information is taken from the transaction abend control block (see “Transaction storage” on page 273).

PSW

If the transaction dump was taken in response to a local abend with abend code AICA, ASRA, ASRB, or ASRD, a PSW is formatted from the dump data set. It belongs to the program that was being executed when the abend occurred. It is taken from the transaction abend control block (see “Transaction storage” on page 273).

Registers at the time of interrupt

If the transaction abended locally with abend code AICA, ASRA, ASRB, or ASRD, you see a set of registers that belong to the program that was executing when the error was detected. They are taken from the transaction abend control block (see “Transaction storage” on page 273).

Execution key

If the transaction abended locally with abend code ASRA or ASRB, the execution key that is in force at the time of the abend is formatted. It is taken from the transaction abend control block (see “Transaction storage” on page 273).

Space

If the transaction abended locally with an ASRA or ASRB abend code, CICS formats the space, basespace or subspace, in which the program was executing. If transaction isolation is not enabled, the program always executes in the basespace.

Registers at last EXEC command

If the transaction has issued any EXEC commands, then a set of registers is displayed. These are the registers from the last EXEC command that was issued.

Task control area (TCA)

The next thing in the transaction dump is the entire TCA. This contains information about the transaction to which the dump relates. Note that the user area precedes the system area.

Task control area, system area

The system area of the task control area is formatted separately as it can be addressed separately by CICS. It contains system information relating to the task and can often be a valuable source of debugging information.

Transaction work area (TWA)

Any transaction work area relating to the transaction is formatted, if present.

EXEC interface structure (EIS)

The EIS contains information about the transaction and program specific to the execution interface component of CICS.

System EXEC interface block (SYSEIB)

This is used solely by programs using the SYSEIB option. See the *CICS/ESA Application Programming Guide* for more details.

EXEC interface user structure (EIUS)

The EIUS contains execution interface component information that must reside in user key storage.

EXEC interface block (DFHEIB)

DFHEIB contains information relating to the passing of EXEC requests from the program to CICS, and the passing of data between the program and CICS. Field EIBFN is of particular interest, because it shows the type of the last EXEC command to be issued by the program. For programming information about the values that EIBFN can contain, see the *CICS/ESA Application Programming Reference* manual or the *CICS/ESA User's Handbook*.

Kernel stack entries

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.

Common system area (CSA)

The CSA is one of the main control areas used by CICS. It contains information relating to the system as a whole, and to the task that was running when the transaction dump was invoked. It can be very useful for debugging both application problems and system problems. You cannot access fields in the CSA in your programs. Attempting to do so causes your transaction to terminate abnormally with abend code ASRD.

Common system area optional features list (CSAOPFL)

The common system area optional features list, an extension of the CSA, contains the addresses of CICS optional features.

Trace table (abbreviated format)

The abbreviated-format trace table is formatted by default. You can suppress it by
specifying the NOABBREV parameter in the DFHDU410 job. A “one entry per line” summary of the trace entries, copied from the internal trace table, is formatted.

Provided that you had EI level-1 and PC level-1 tracing selected for your task, you can identify the last CICS command issued by your task quite easily. The procedure is outlined in “Locating the last command or statement” on page 274.

Trace table (extended format)

Following the abbreviated-format trace table is the corresponding extended-format
trace table. This is formatted by default. You can suppress it by specifying the
NOFULL parameter in the DFHDU410 job. It contains more detail, but you can probably get all the information you need using the abbreviated trace.

Common work area (CWA)

The CWA is the installation-defined work area for use by all programs and is formatted if it exists.

Transaction storage

“Transaction storage” is storage that might have been obtained by CICS to store information about a transaction, or it might have been explicitly GETMAINed by the transaction for its own purposes. You are likely to find several such areas in the dump, each introduced by a header describing it as transaction storage of a particular class, for example:

USER24
USER31
CICS24
CICS31

Transaction storage class CICS31 contains, among other things, the transaction abend control block (TACB). To find it, look for the eye-catcher **DFHTACB**. DFHTACB contains valuable information relating to the abend. It contains:

- The PSW and general purpose registers of the program executing at the time of the abend (for local AICA, ASRA,ASRB and ASRD abends only. However, for some AICA abends, only the “next sequential instruction” part of the PSW
and the registers are present.) Registers 12 and 13 contain the addresses of
the TCA and CSA respectively in all abends except for ASRA and ASRB
abends, when these registers contain data as at the time of the abend.
- The name of the failing program.
- The offset within the failing program at which the abend occurred (for local ASRA, ASRB and ASRD abends only).
- The execution key at the time of the abend (for local ASRA and ASRB abends only).
- Whether the abend was caused by an attempt to overwrite a protected CICS DSA (local ASRA abends only).
- Whether the program is executing in a subspace or the basespace.

- The subspace STOKEN.
- The subspace's associated ALET.

Note that if the abend originally occurred in a remote DPL server program, an eye-catcher ***REMOTE*** is present. If this is the case, the registers and PSW are not present.

Terminal control table terminal entry (TCTTE)

The TCTTE contains information about the terminal that is the principal facility of the transaction. You usually find one TCTTE for the transaction if the transaction is terminal oriented. For "daisy chained" transactions, you may find more than one.

To look at the TIOA for the task, find the address in field TCTTEDA of the TCTTE.

Program information for the current transaction

This shows summary information for all linked programs for the transaction that have not yet returned, including load point, entry point, and length. This is followed by the program storage for each of these programs. This is where you can find the instructions addressed by register 14 and by the PSW, and hence the point of failure in your program. For details of how you do this, see "Locating the last command or statement."

Other program manager control blocks are shown too, including PPT entries for active programs, and load list elements and program storage for any programs loaded by this transaction but not yet released.

Module index

The final item that you find in the transaction dump is the module index. This shows you all the modules that were in storage when the error was detected, and their addresses.

Locating the last command or statement

The easiest way of locating the last command issued by your application before the abend is to use the internal trace table. You must have had internal trace running, and you need to have captured entries from the EI level-1 and PC level-1 trace points for your task. The way you can find the last command using trace is described in "Last command identification."

If you did not have trace running, you need to rely on values in registers belonging to your programs, and try to relate these to your source statements. That might lead you to an EXEC CICS command, or to another type of statement in your program. The procedure is outlined in "Last statement identification" on page 275.

Last command identification

This process for identifying the last command applies to system dumps, in which it is necessary to identify the abending task. Item 2 on page 275 is relevant to transaction dumps.

1. If the abend was a local AICA, ASRA, ASRB, or ASRD, find the last entry in the table and work back until you find an entry for trace point ID AP 1942. If you cannot find AP 1942, then search for any one of the following: AP 0790, AP 0791, or AP 0792. The trace entry for AP 1942 is made on entry to APLI's

recovery routine. The entries for AP 0790, AP 0791, and AP 0792 are made by DFHSRP, the AP domain recovery routine that deals with program checks, operating system abends, and runaway task conditions. The task number for your task is shown in the entry.

If the abend was none of those mentioned above, find the last entry in the table and work back until you find an entry for trace point ID AP 00F2 (PCP abend) that references the abend code. The task number of your task is shown in the entry.

2. Now go back until you find the last trace entry showing your task number that was made from trace point ID AP 00E1. The trace entry is located in the EXEC interface program, DFHEIP. The data in the trace entry includes the value of EIBFN, which tells you the specific command your program issued before the abend. For programming information about the possible values that EIBFN can take, and their meanings, see the *CICS/ESA Application Programming Reference* manual, or the *CICS/ESA User's Handbook*.
3. You might now be able to identify the program that was being run when the abend occurred, from knowing the structure of the application. If not, you can identify the program by using the information in the "program information for the current transaction" section of the dump. The failing program is the one most recently linked to (the first program printed in this section). The summary information includes the name of the program, and its load point, entry point, and length.
4. You should by now have found the program containing the last EXEC command that was issued before the abend. You next need to locate the EXEC command in that program. If you cannot do it by inspection, use the techniques described in the next section.

Last statement identification

1. Locate the "transaction storage –CICS31" areas of the transaction dump. These areas are maintained by CICS, and they relate to the transaction that was running when the abend occurred. You should be able to see the eye-catcher **DFHTACB** in at least one of the areas. This signifies the start of the transaction abend control block, and it contains the registers and PSW belonging to the program being executed when the abend occurred. If there is more than one area containing this eye-catcher, it means that two or more successive abends occurred. You need to find the first occurrence, because that relates to the abend that started the sequence.
2. Locate the PSW for the program in the TACB, and make a note of the next sequential instruction address. The PSW for the program is present if the abend is AICA, ASRA, ASRB or ASRD. Alternatively, obtain the offset of the abend within the failing program load module from the TACB. The offset is present if the abend is ASRA, ASRB or ASRD and is valid if not `X'FFFFFFFF'`. Note down the value of register 14, too.
3. Use the "program information for the current transaction" section of the dump to obtain the name and entry point of the failing program. Alternatively, obtain the name of the failing program from the TACB.
4. The offset or PSW should point to an instruction in the program. If it doesn't, register 14 should show a return address into your program. Either way, you need to correlate the address with a statement in the source code for the program.

If the source language is **assembler**, the instruction where the abend occurred is apparent from the program storage in the dump. If the source language is **COBOL**, **PL/I**, or **C/370**, you need to refer to a compiler output mapping source statements onto the object code.

Note: In system dumps formatted by DFHPD410, a program's load point is formatted in the kernel error data section as an asterisk (*). Because the EXEC interface stub is linked at the start of the program code, you need to subtract X'28' bytes from this load point in order to calculate the actual offset of the failing instruction.

Locating program data

You might want to look at the data that your application program has in its storage areas. CICS maintains a pointer to the chain of dynamic storage that the program uses, in field TCAPCDSA of the system area of the TCA.

- For **PL/I programs**, TCAPCDSA addresses the chain of PL/I DSAs.
- For **COBOL programs**, TCAPCDSA addresses the task global table (TGT) and working storage.
- For **assembler programs**, TCAPCDSA addresses the DFHEISTG storage.

You need to look in the appropriate programming language manual for details of the structure of the program's acquired storage.

Dumps for C/370 programs

If DUMP(YES) is coded on the transaction definition, CICS system and transaction dumps are produced for failing C/370 programs. The use of the relevant C/370 registers is as follows:

Register	Use
3	In most circumstances, is the base register
12	Holds the address of the CICS TCA for the C/370 program
13	Holds the address of the register save area

Location of COBOL dumped areas

The dumped COBOL program can be found in the "program information for the current transaction" section of the dump, and is addressed by the LOAD_POINT parameter on the appropriate LDLD ACQUIRE_PROGRAM exit trace entry. The register save area INIT1+X'48' (covering registers 0 through 14) should have register 12 pointing to the program global table (PGT), register 13 pointing to the task global table (TGT), and some others to locations in the data area and compiled code of the program storage. If not, a CICS error is indicated.

For each invocation of the COBOL program, CICS copies the **static** TGT from program storage into CICS **dynamic** storage (the COBOL area) and uses the dynamic copy instead of the static one. CICS also copies working storage from program storage to the COBOL area, above the TGT. Task-related COBOL areas thus enable the single copy of a COBOL program to multithread as if it were truly reentrant.

The dumped COBOL area

The COBOL area is addressed by TCAPCDSA (alias TCAPCCA) in the system part of the TCA (and forms part of the transaction storage chain). The COBOL area contains:

- The COBOL working storage for the task
- A copy of the TGT.

The TGT is addressed by TCAPCHS in the system part of the TCA.

The TGT is used to hold intermediate values set at various stages during program execution. The first 18 words of the TGT constitute a standard save area, in which the program's current registers are stored on any request for CICS service.

The TGT also holds the base locator for linkage (BLL) cells, which contain the addresses of all areas defined in the linkage section of the COBOL program. You can use the memory map from the COBOL compiler listing for your application to find the offset of the BLL cells in the TGT.

BLL cells in a CICS application program

To investigate a problem connected with BLL cells, you should examine the COBOL compiler listing for your application to determine the use you have made of the BLL cells.

The **first** BLL cell that is used points to the EXEC interface block (an area that follows the literal string 'DFHEIB').

The **second** BLL cell that is used points to the first (or only) 4096-byte block of CICS COMMAREA as supplied with an EXEC CICS LINK, XCTL, or RETURN with a TRANSID. If the COMMAREA is greater than 4096 bytes, the **third** BLL cell that is used points to the next 4096-byte block of COMMAREA; further 4096-byte COMMAREAs are pointed to by succeeding BLL cells.

The next BLL cell always points to itself; the remaining BLL cells are set by the application program. Thus, for example, if the COMMAREA size is 10KB, it is the **fifth** BLL cell used that points to itself.

Note that an ASRD abend will occur if an attempt is made to access storage via the BLL cell immediately after the one that points to itself, if it has not yet been set by the application program.

Formatting system dumps

You can process system dumps from the SYS1.DUMPxx data set using an invocation of the interactive problem control system (IPCS).

In releases prior to CICS/ESA 4.1, the CICS formatting routine for use under the MVS interactive problem control system (IPCS) is supplied as DFHPDX. This standard name is not suitable for those users running more than one release of CICS, because the dump formatting process in each version of DFHPDX is release-specific, and you must use the correct version for the system dump you are formatting. The module is named with the release identifier as part of the name – DFHPD410 is the formatting routine you must define to IPCS when formatting CICS/ESA 4.1 system dumps.

The DFHIPCSP CICS exit control data

From MVS/ESA 3.1.3, IPCS provides an exit control table with imbed statements to enable other products to supply exit control information. The IPCS default table, BLSCECT, normally in SYS1.PARMLIB, has the following entry for CICS:

```
# IMBED MEMBER(DFHIPCSP)  ENVIRONMENT(ALL) /*CICS                */
```

Ensure that the CICS-supplied DFHIPCSP member can be found by your IPCS job. You can either copy DFHIPCSP into SYS1.PARMLIB (so that it is in the same default library as BLSCECT) or provide an IPCSPARM DD statement to specify the library containing the IPCS control tables. For example:

```
//IPCSPARM  DD  DSN=SYS1.PARMLIB,DISP=SHR         For BLSCECT
//          DD  DSN=CICS410.SDFHPARM,DISP=SHR     For DFHIPCSP
```

Figure 45 shows the release-specific entries specified in DFHIPCSP.

```
/* ============================================================= */
   EXIT EP(DFHPD212) VERB(CICS212) ABSTRACT(+
     'CICS Version 2 Release 1.2 analysis')
   EXIT EP(DFHPD321) VERB(CICS321) ABSTRACT(+
     'CICS Version 3 Release 2.1 analysis')
   EXIT EP(DFHPD330) VERB(CICS410) ABSTRACT(+
     'CICS Version 3 Release 3 analysis')
   EXIT EP(DFHPD410) VERB(CICS410) ABSTRACT(+
     'CICS Version 4 Release 1 analysis')
/* ============================================================= */
```

Figure 45. Release-specific entries in DFHIPCSP for DFHPDnnn routines

To use the DFHIPCSP member as it is, rename the CICS-supplied version of DFHPDX for earlier releases to the names shown in the table.

- The IPCS dump analysis subcommands enable you to format and analyze CICS-produced SDUMPs, which you can either view at a terminal or print. You can:
  - Examine the data in a dump
  - Locate and verify control blocks associated with certain functions or system components
  - Trace and verify chains of control blocks
  - Perform contention analysis on key MVS resources
  - Locate modules and unit control blocks
  - Execute user-written exits for certain control blocks
  - Keep a list of names and locations of control blocks and areas of the dump that you consider important.
- The CICSnnn dump exit (where nnn is the CICS release identifier) enables you to format a dump selectively by specifying one or more CICS component identifiers as parameters to the exit. Thus, the CICS410 dump exit enables you to process a CICS/ESA 4.1 dump selectively. You can:
  - Specify which job in the dump data set is to be formatted (**JOB** parameter).

- Specify which component storage areas are to be formatted, and at what level of detail, by using formatting keywords and level numbers (**keyword** parameter). You do this using the IPCS command: `VERBEXIT CICS410 'keyword,...'`.
- Specify the default level of detail for components that are not explicitly identified by keyword (**DEF** parameter).
- Specify whether the output is to be printed in uppercase characters (UPPERCASE parameter).

The use of formatting keywords enables you to format those parts of the dump that interest you at any particular time, at specific levels of detail. You have the option of formatting other parts later for further investigation by you or by the IBM service organizations. It is advisable to copy your dumps so that you can save the copy and free the SYS1.DUMPxx data set for subsequent use.

For guidance about using the formatting keywords and levels, see the *CICS/ESA Operations and Utilities Guide*.

The default SDUMP formatting levels

The **DEF** dump exit parameter specifies the default level of formatting you want for data from the dump data set. Note that the DEF parameter is only effective for components that are not included in a list of component keywords.

The levels that you can specify are as follows:

Level Meaning

0 For those components not included in a specified list of keywords, suppress all component formatting. Note that if you specify DEF=0, but do not specify any component keywords, you still get the dump summary and, if appropriate, the error message index.

1 For those components not included in a specified list of keywords, and where applicable, format the summary information only. (A summary is not available for all components; see the level numbers available for the individual keywords for those for which a summary of dump information is available.)

2 For those components not included in a specified list of keywords, format the control block information only.

3 For those components not included in a specified list of keywords, format the control block information and also (where applicable) the summary information.

Effects of omitting the DEF parameter:

- If you omit the DEF parameter and *do not* specify any component keywords, the result is as if you specified DEF=3.
- If you omit the DEF parameter *and* specify one or more component keywords, the result is as if you specified DEF=0.

Exceptions to the scope of the DEF parameter: There are two parts of a CICS system dump that are not governed by component keywords, and are therefore outside the scope of the DEF parameter. These are:

1. The dump summary

2. The error message index.

These parts of a CICS system dump are always formatted, even if you specify DEF=0 and no component keywords.

Storage freeze

Certain classes of CICS storage that are normally freed during the processing of a transaction can, optionally, be kept intact and freed only at the end of the transaction. Then, in the event of an abend, the dump contains a record of the storage that would otherwise have been lost, including the storage used by CICS service modules. The classes of storage that can be frozen in this way are those in the teleprocessing and task subpools, and in terminal-related storage (TIOAs).

The storage freeze function is invoked by the CSFE transaction. For information about using CSFE, see the *CICS/ESA CICS-Supplied Transactions* manual.

Using FEPI dump

For information about using dumps to solve FEPI problems, refer to the *CICS/ESA Front End Programming Interface User's Guide*.

Chapter 16. The global trap/trace exit

The global trap/trace exit (DFHTRAP) is intended to be used only under the guidance of IBM Service personnel. It is designed so that a detailed diagnosis of a problem can be made without having to stop and then restart CICS.

Typically, the global trap/trace exit is used to detect errors that cannot be diagnosed by other methods. These might cause intermittent problems that are difficult to reproduce, the error perhaps occurring some time before the effects are noticed. For example, a field might be changed to a bad value, or some structure in storage might be overlaid at a specific offset.

DFHTRAP is an assembler language program that can be invoked whenever the trace (TR) domain is called to make a trace entry. The trap must be activated before it is used, either dynamically or at CICS initialization.

A skeleton version of DFHTRAP is supplied in both source and load-module forms. The source of DFHTRAP is cataloged in the CICS410.SDFHSRC library. The source of the skeleton DFHTRAP contains comments explaining its use of registers and DSECTs, and the coding you need to do if you need to use the exit.

The code in DFHTRAP must **not** make use of any CICS services, cause the current task to lose control, or change the status of the CICS system.

Establishing the exit

DFHTRAP can be installed as part of the RDO group DFHFE. You can install it either with the GRPLIST system initialization parameter, or dynamically using the CEDA transaction.

When CICS is running, activation and deactivation of the trap exit routine can be requested using the TRAP operand of the CSFE DEBUG command:

```
CSFE DEBUG,TRAP={ON|OFF}
```

The trap exit can also be activated at CICS initialization by specifying TRAP=ON, either in DFHSIT or as a startup override. If you want to **replace** a trap exit routine while CICS is running, use the CSFE DEBUG commands in conjunction with the CEMT NEWCOPY command. The following sequence of commands causes the currently active version of DFHTRAP to be refreshed:

```
CSFE DEBUG,TRAP=OFF
CEMT SET PROGRAM(DFHTRAP) NEWCOPY
CSFE DEBUG,TRAP=ON
```

Information passed to the exit

To assist in the diagnosis of faults in a CICS system, the trace domain passes information to the exit in a parameter list addressed by register 1. A DSECT (DFHTRADS) is supplied for this list, which contains the addresses of:

- The return-action flag byte
- The trace entry that has just been added to the table
- Up to three areas to be included in a further trace entry
- An 80-byte work area for the sole use of the trap exit
- The CSA (which may be zero during early initialization)
- The TCA, if there is one
- A register save area.

The DSECT also contains EQU statements for use in setting the return-action flag byte.

The exit can look at data from the current trace entry to determine whether or not the problem under investigation has appeared. It can also look at the TCA of the current task, and the CSA. The DSECTs for these areas are included in the skeleton source.

The CSA address is zero for invocations of DFHTRAP early in initialization, before the CSA is acquired.

Actions the exit can take

The return-action flag byte can be set by the exit to tell the trace domain what action is required on return from the exit. The following list gives the possible choices:

- Do nothing.
- Make a further trace entry.
- Take a CICS system dump using dump code TR1003.
- Terminate CICS without a dump after message DFHTR1000. (If you require a dump, the system dump return-action flag must also be set.)
- Disable the trap exit, so that it is not invoked again until reactivated by the CSFE transaction.

Any combination of these actions can be chosen and all actions are honored on return to the trace domain.

To reactivate the trap exit when it has been disabled, use CSFE DEBUG,TRAP=ON, unless the exit routine is to be replaced. In this case the sequence of commands given above applies.

The skeleton program shows how to make a further trace entry. When DFHTRAP detects a TS GET request, it asks for a further trace entry to be made by entering the data required in the area supplied for this purpose, and by setting the appropriate bit in the return-action flag byte.

The trace domain then makes a trace entry with trace point ID TR 0103, incorporating the information supplied by the exit.

Trace entries created in this way are written to any currently active trace destination. This could be the internal trace table, the auxiliary trace data set, or the GTF trace data set.

The skeleton DFHTRAP also shows how to detect the trace entry made by the storage manager (SM) domain for a GETMAIN request for a particular subpool. This is provided as an example of how to look at the data fields within the entry.

Program check handling

The occurrence of a program check in the trap exit is detected by the recovery routine in DFHTRPT. This then:

1. Marks the exit as unusable
2. Issues the message DFHTR1001 to the system console
3. Takes a CICS system dump with dump code TR1001, showing the PSW and registers at the time of the interrupt
4. Continues (ignoring the exit on future invocations of the trace domain).

To recover from this situation, execute the commands given above for replacing the current version of the exit routine.

Coding the exit

When using the trap exit, note the following points:

- A skeleton version of DFHTRAP is supplied in both source and load-module forms. Make sure that the library search sequence in the CICS startup JCL finds the correct version of the load module.
- DFHTRAP must save and restore the trace domain's registers. The supplied skeleton version contains the code necessary to do this. You are strongly advised not to change this code.
- The 80-byte work area provided for the sole use of the exit is in working storage acquired by the trace domain using an MVS GETMAIN. It is acquired and initialized to binary zeros when the trap is activated. It then exists until the trap is deactivated (CSFE DEBUG,TRAP=OFF). In a CICS transaction dump, the DFHTRAP working storage can be found soon after the CSA optional features list. The 80-byte work area is at the end of the DFHTRAP working storage and is immediately preceded by a 16-byte eye-catcher (**DFHTRAP_WORKAREA**), so that the work area can be located even if it has not been formatted. In a CICS system dump, the DFHTRAP working storage is in the trace domain (TR) section. See "Formatting system dumps" on page 277 for details of how to use the TR keyword to format the trace domain information in the dump.
- DFHTRAP must always run with AMODE(31) and RMODE(ANY) specified. In particular, it must always return control to the trace domain in 31-bit mode.

Part 4. Working with IBM to solve your problem

Part 4 contains:

Chapter 17. IBM program support

The IBM Customer Engineering Program Support structure exists to help you resolve problems with IBM products, and to ensure that you can make the best use of your IBM computing systems. Program support is available to all licensed users of IBM licensed programs, and you can get assistance by telephoning your local Support Center.

This chapter helps you decide when to contact the Support Center, and what information you need to collect before contacting the Center. The chapter also gives you an understanding of the way in which IBM Program Support works.

When to contact the Support Center

Before contacting the Support Center, try to ensure that the problem belongs with the Center. Don't worry if you can't be sure that the problem is due to CICS itself. How sure you are depends on the complexity of your installation, the experience and skill levels of your systems staff, and the symptoms that you have been getting.

In practice, many errors reported to Program Support turn out to be user errors, or they cannot be reproduced, or they need to be dealt with by other parts of IBM Service such as Hardware CE or Systems Engineering. This indicates just how difficult it can be to determine the precise cause of a problem. User errors are mainly caused by faults in application programs and errors in setting up systems. TCT parameters, in particular, have been found to cause difficulty in this respect.

Dealing with the Support Center

Your first contact with the Support Center is the call receipt operator, who takes initial details and routes your call to the correct support group.

The Support Center needs to know as much as possible about your problem, and you should have the information ready before making your first call. It is a good idea to put the information down on a problem reporting sheet, such as the one shown in Figure 46 on page 288.

PROBLEM REPORTING SHEET

Date	Severity	Problem No.
		Incident No.

Problem/Enquiry

Abend/Prog CK	Incorrout	MVS Rel
Wait	Module	MVS Lvl
Loop	Message	CICS Rel
Performance	Other	CICS Lvl

Documentation available

Abend	System dump	Program output
Message	Transaction dump	Other
Trace	Translator output	
Symptom string	Compiler output	

Actions

Date	Name	Activity

Resolution

APAR	PTF	Other

Figure 46. Sample problem reporting sheet

There are two advantages of using a problem reporting sheet:

1. You will be communicating with the IBM Support Center by telephone. So, with all your findings before you on a sheet of paper, you are prepared to respond to the questions that you may be asked.
2. You should maintain your own in-house tracking system to record all problems. This information can then be used for planning, organizing, communicating, and establishing priorities for controlling and solving these problems.

What the Support Center needs to know

When you contact the Support Center, you need to give the operator the name of your organization and your access code. Your access code is a unique code authorizing you to use IBM Software Services, and you provide it every time you contact the Center. Using this information, the operator accesses your customer profile, which contains details of your address, relevant contact names, telephone numbers, and details of the IBM products at your installation.

The Support Center operator asks you if this is a new problem, or a further call on an existing one. If it is new, you are assigned a unique incident number. A problem management record (PMR) is opened on the RETAIN system, where all activity associated with your problem is recorded. The problem remains "open" until it is solved.

Make a note of the incident number on your own problem reporting sheet. The Center expects you to quote the incident number in all future calls connected with this problem.

If the problem is new to you, the operator asks you for the source of the problem within your system software—that is, the program that seems to be the cause of the problem. As you are reading this book, it is likely that you have already identified CICS as the problem source. You also need to give the version and release number, for example Version 4 Release 1.

You need to give a severity level for the problem. Severity levels can be 1, 2, or 3. They have the following meanings:

- Severity level 1 indicates that you are unable to use a program, resulting in a critical condition that needs immediate attention.
- Severity level 2 indicates that you are able to use the program, but that operation is severely restricted.
- Severity level 3 indicates that you are able to use the program, with limited functions, but the problem is not critical to your overall operation.

When deciding the severity of the problem, take care neither to understate it nor to overstate it. The Support Center procedures depend on the severity level so that the most appropriate use can be made of the Center's skills and resources. Your problem is normally dealt with immediately if it is severity level 1.

Finally, the call receipt operator offers you a selection of specific component areas within CICS (for example, terminal control, file control) and asks you to choose the area where your problem appears to lie. Based on this selection, the operator can route your call to a specialist in the chosen area.

The keywords are subsequently used as search arguments on the RETAIN database, to see if your problem is a known one that has already been the subject of an authorized program analysis report (APAR).

You are not asked for any more information at this stage. However, you need to keep all the information relevant to the problem, and any available documentation such as dumps, traces, and translator, compiler, and program output.

How your problem is subsequently progressed depends on its nature. The representative who handles the problem gives you guidance about what is required from you. The possibilities are described in the next section.

What happens next

Details of your call are passed using the RETAIN problem management system to the appropriate support group. Your problem, assuming it is one associated with CICS, is put on the CICS queue. The problems are dealt with in order of severity level.

At first, a support center representative uses the keywords that you have provided to search the RETAIN database. If your problem is found to be one already known to IBM, and a fix has been devised for it, a Program Temporary Fix (PTF) can quickly be dispatched to you.

Let the representative know if any of the following events occurred before the problem appeared:

- Changes in level of MVS or licensed programs
- Regeneration of any product
- PTFs applied
- Additional features used
- Application programs changed
- Unusual operator action.

You might be asked to give values from a formatted dump or trace table. You might also be asked to carry out some special activity, for example to set a trap, or to use trace with a certain type of selectivity, and then to report on the results.

It might be necessary to have several follow-up telephone calls, depending on the complexity of the symptoms and your system environment. In every case, the actions taken by you and the Support Center are entered in the PMR. The representative can then be acquainted with the full history of the problem before any follow-up call.

The result of the investigations determines whether your problem is a new one, or one that is already known. If it is already known, and a fix has been developed, the fix is sent to you.

If the problem is new, an APAR may be submitted. This is dealt with by the CICS change team. See Chapter 18, “APARs, fixes, and PTFs” on page 293.

IBM Program Support structure

Figure 47 on page 291 attempts to show the relationships between the customer, the staff at the IBM Support Center, and the change team. The role of each member of the IBM support staff is outlined in the preceding parts of this chapter.

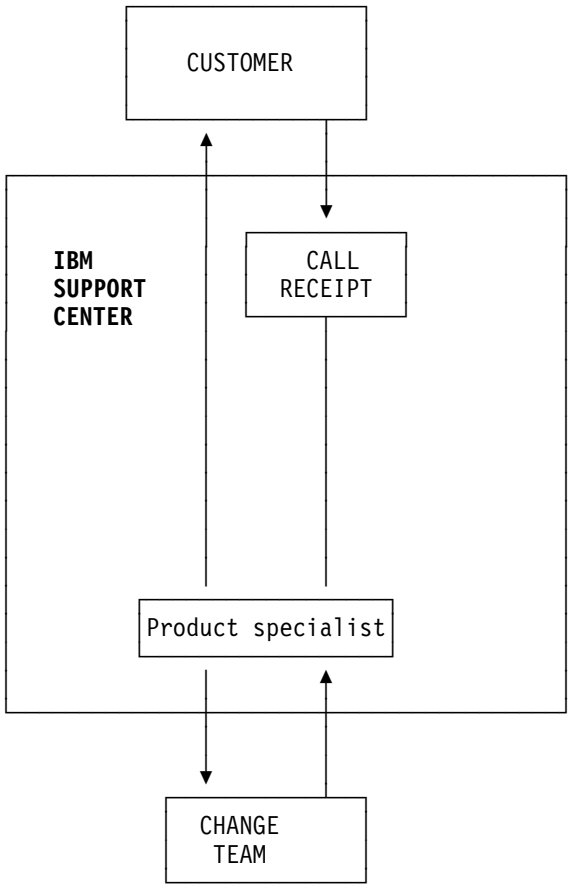


Figure 47. Structure of IBM Program Support

Reporting a FEPI problem to IBM

For information specifically about reporting a FEPI problem, refer to the *CICS/ESA Front End Programming Interface User's Guide*.

Chapter 18. APARs, fixes, and PTFs

An APAR is an “authorized program analysis report”. An APAR is your means of informing the appropriate change team of a problem you have found with an IBM program.

When the change team solves the problem, they produce a fix enabling you to get your system running properly again. Finally, a PTF is produced to replace the module in error, and the APAR is closed.

The APAR process

The first step in the APAR process is that a support center representative enters your APAR into the RETAIN system. The APAR text contains a description of your problem. If you have found a means of getting round the problem, details of this are entered as well. Your name is also entered, so that the Support Center knows who to contact if the change team need to ask anything further about the APAR documentation.

When the APAR is entered, you are given an APAR number. You must write this number on all the documentation you submit to the change team. This number is always associated with the APAR and its resolution and, if a code change is required, it is associated with the fix as well.

The next stage in the APAR process, getting relevant documentation to the change team, is up to you.

The following is a summary of the things you need to do:

1. You must collect all of the documentation that is required for the APAR. You are given guidance by the support center on precisely what you need to send. The documentation that is required varies, depending on the problem area, but “Collecting the documentation for the APAR” gives you an idea of the material that you should supply.
2. You need to package all the documentation and send it to the change team. The procedure for this is given in “Sending the documentation to the change team” on page 294.
3. Lastly, you need to apply the PTF resulting from the APAR, possibly after testing the fix on your system. This is described in “Applying the fix” on page 295.

Collecting the documentation for the APAR

As a general rule, the documentation you need to submit for an APAR includes all the material you need yourself to perform problem determination. Some of the documentation is common to all CICS problems, and some is specific to particular types of problem.

Make sure the problem you have described can be seen in the documentation you send. If the problem has ambiguous symptoms, you need to reveal the sequence of events leading up to the failure. Tracing is valuable in this respect, but you

might be able to provide details that trace cannot give. You are encouraged to annotate your documentation, if your annotation is legible and if it does not cover up vital information. You can highlight data in any hard copy you send, using transparent highlighting markers. You can also write notes in the margins, preferably using a red pen so that the notes are not overlooked.

Finally, note that if you send too little documentation, or if it is unreadable, the change team will return the APAR marked “insufficient documentation”. It is, therefore, worthwhile preparing your documentation carefully and sending everything relevant to the problem.

The general documentation is described in “General documentation needed for all problems with CICS.” However, these are only guidelines—you must find out from the support center precisely what documentation you need to send for your specific problem.

General documentation needed for all problems with CICS

The following is a list of the general documentation you might be asked to submit for an APAR:

- Any hard or soft copy illustrating the symptoms of the problem.
- A system dump of the CICS address space. Format the whole system dump if you plan to submit hard copy. Otherwise, you can just send the system dump data set on tape.
- A CICS trace. Auxiliary trace and GTF trace are best, but internal trace will do if you don’t have either of these. If internal trace has been running, it is in the system dump in any case.
- Listings of CICS management modules, such as DFHKCP, that are not supplied as object code only. **These should be unnecessary if preassembled modules are used**.
- Relevant CICS tables. Again, these should be unnecessary if preassembled versions are used.
- Listings of relevant application programs.
- Console logs.
- CICS logs (for example, the CSMT log) wherever possible. These contain information that is often overlooked. They are particularly useful when VTAM is in use.
- JCL listings. These may appear on dumps, and need not be sent twice.
- A list of PTFs and APARs applied. The System Modification Program (SMP) CICS control data set (CDS) provides this information and should be sent.
- Details of any user modifications.

Sending the documentation to the change team

Follow the directions in APAR II02709 when gathering documentation to send to the change team. If you do not have access to the RETAIN database, ask your IBM representative to obtain a copy of APAR II02709 for you. The documentation you submit for the problem is best shipped in an APAR box, which you can obtain

from your local IBM branch. APAR boxes are clearly marked as such, and they have a panel where tracking information such as the APAR number can be written.

Packing and mailing the APAR box

Place all your documentation and notes in one or more APAR boxes, making sure that the boxes are marked, for example, “1 of 2”, and so on, if you need to use more than one.

If you include any magnetic tapes, write this clearly on the outside of the box. This lessens the chance of their being stored in magnetic fields strong enough to damage the data.

To make sure the documentation reaches the correct destination, that is, the CICS change team, write the following on the box:

```
SHIP TO CODE 5U6
```

You also need a mailing label with the address of the CICS change team on it.

When the change team receives the package, this is noted in your APAR record on the RETAIN system. The team then investigates the problem. Occasionally, they need to ask the Support Center to contact you for more documentation, perhaps specifying some trap you must apply before getting it.

When the problem is solved, a code is entered on RETAIN to close the APAR, and you are provided with a fix.

You can enquire any time at your Support Center on how your APAR is progressing, particularly if it is a problem of high severity.

Applying the fix

When the change team have found a fix for your problem, they might want you to test it on your system. If they do ask you to test the fix, you are normally given two weeks to do it and to provide them with the results. However, you can ask for an extension if you are unable to complete the testing in that time.

When the team is confident that the fix is satisfactory, the APAR is certified by the CICS development team and the APAR is closed. You receive notification when this happens.

The APAR becomes a PTF

If the solution involves a change to code in a CICS module that you can assemble, you are sent the code change right away. The change is later distributed as a PTF.

If you cannot assemble the module yourself, because it involves a part of CICS that is object serviced, you might be supplied with a ZAP or a TOTEST PTF.

If you want a PTF to resolve a specific problem, you can order it explicitly by its PTF number through the IBM Support Center. Otherwise, you can wait for the PTF to be sent out on the standard distribution tape.

Part 5. Appendixes

Appendix A. SDUMP contents and IPCS CICS VERBEXIT keywords

The following two tables provide a cross-reference between the CICS control blocks contained in an SDUMP and their associated IPCS CICS VERBEXIT keyword.

The first table provides a list of IPCS CICS VERBEXIT keywords and the CICS control blocks that they display.

The second table provides a list of all CICS control blocks in an SDUMP, alphabetically, with their associated IPCS CICS VERBEXIT keyword.

Finding the control blocks from the keywords

AI keyword

- AITM (static storage)
- AITMTE (autoinstall terminal models)

AP keyword

- CICS24 (task storage – below 16MB, CICS key)
- CICS31 (task storage – above 16MB, CICS key)
- DBL (journal control dynamic log buffer)
- DWE (user DWE storage)
- EIB (EXEC interface block)
- EIS (EXEC interface structure)
- EIUS (EXEC interface user structure)
- FILE (user file storage)
- JCA (journal control area)
- MAPCOPY (user BMS MAP storage)
- QCA (queue control area)
- QEA (queue element area)
- SYSEIB (system EXEC interface block)
- SYS_TCA (task control area – system area only)
- TD (user transient data)
- TS (user temporary storage)
- TCA (task control area – user)
- USER24 (task storage – below 16MB, user key)
- USER31 (task storage – above 16MB, user key)

CC keyword

- Local (anchor block)
  - CCBUFFER (local catalog buffers – one each thread)
  - CC_ACB (local catalog ACB)
  - CC_RPL (local catalog RPLs – one each thread)
- Global (anchor block)
  - GCBUFFER (global catalog buffers – one each thread)
  - GC_ACB (global catalog ACB)
  - GC_RPL (global catalog RPLs – one each thread)

CP keyword

- CPSTATIC (common program interface storage)

CSA keyword

- CSA (common system area)
- CSAOPFL (CSA optional features list)
- CWA (common work area)

DD keyword

-

DLI keyword

- CWE (CICS-DBCTL control work element)
- DFHDLP (CICS-DLI interface parameter list)
- DGB (CICS-DBCTL global block)
- DGBCTA (DBCTL transaction area)
- DSB (CICS-DBCTL scheduling block)

- DXPS (CICS-DL/I-XRF anchor block)
- ISB (interface scheduling block)
- PAPL (DL/I-DRA architected parameter list)
- PST (partition specification table)
- RSB (remote scheduling block)
- SCD (system contents directory)
- SYS_TCA (TCA system area only)
- TCA (task control area - user)

DM keyword

- DMANCHOR (domain manager anchor block)
- WQP (domain wait queue)

DS keyword

- DSANC (dispatcher anchor block)
- DS_TCB (TCB block)
- DTA (dispatcher task area)
- SUSPAREA (unformatted SUSPEND_AREAS/TOKENS)
- TASK (unformatted DTAs)

DU keyword

- DUA (dump domain anchor block)
- DUBUFFER (transaction dump data set buffer)
- OPENBLOK (transaction dump open block)
- SDTE (system dump table elements)
- TDTE (transaction dump table elements)

FCP keyword

- ACB (VSAM ACBs)
- AFCTE (application file control table element)
- DCB (data control block)
- DSNB (data set name block)
- DFHDTTABLE (data table base area)
- DFHDTFILE (data table path area)
- DFHDTHEADER (data table global area)
- DTRGLOBL (data table remote global area)
- FBWA (data table browse area)
- FCSTATIC (FCP static storage – anchor block)
- FCTE (file control table element)
- SHRCTL (shared LSRPOOLs)
- VSWA (VSAM work area)

ICP keyword

- ICE (interval control elements/AIDs)

JCP keyword

- BUFFER (journal buffer)
- DCB (data control block)
- DECB (data extent control block)
- DFHJCOCL (journal open/close parameter list)
- DFHJCT (journal control table header)
- DFHJCTTE (journal table entry)
- DFHSERIS (series definition table)
- JAB (journal archive anchor block)

- JACD (journal archive control data set ACB)
- JACD (journal archive control data set RPL)
- JAW (journal archive work element)
- LECBS (logical ECBs)
- VOLINDEX (cyclic index of volume identifiers)

KE keyword

- AFCB (CICS AFCB)
- AFCS (CICS AFCS)
- AFT (CICS AFT)
- AUTOSTCK (automatic storage stack entry)
- KCB (kernel anchor block)
- KERNSTCK (kernel linkage storage stack entry)
- KERRD (kernel error data)
- KTCB (KTCB table entry)
- DOH (domain table header)
- DOM (domain table entry)
- TAH (task table header)
- TAS (task table entry – TASENTRY)
- TCH (KTCB table header)

LD keyword

- APE (active program element)
- CPE (current program element)
- CSECTL (program CSECT List)
- LD_GLBL (loader domain global storage – anchor block)
- LDBE (loader domain browse element)
- LDWE (loader domain wait element)
- LLA (load list area)

LM keyword

- FREECHAI (LM domain freechain 1)
- FREECHAI (LM domain freechain 2)
- FREECHAI (LM domain freechain 3)
- LMANCHOR (lock manager domain anchor block)
- LMQUICK1 (LM domain quickcell 1)
- LMQUICK2 (LM domain quickcell 2)
- LMQUICK3 (LM domain quickcell 3)
- LOCK_ELE (LM domain lock element)

ME keyword

- MEA (message domain anchor block)

MN keyword

- MCT (monitoring control table)
- MNA (monitoring domain global storage – anchor block)
- MNAFB (monitor authorization facility parameter block)
- MNCONNS (monitor field connectors)
- MNDICT (monitor dictionary)
- MNEVE (SYSEVENT record buffer)
- MNEXC (exception record buffer)
- MNEXLIST (user EMP address list)
- MNFLDMAP (excluded/included CICS field map)
- MNPER (performance data buffer)

- MNSMF (SMF record buffer)
- MNTMA (transaction monitoring area)
- MNWLMPB (MVS WLM performance blocks)

MRO keyword

- CCB (connection control block)
- CRB (CICS region block)
- CSB (connection status block)
- DWE (deferred work element)
- LACB (logon address control block)
- LCB (logon control block)
- LXA (LX array)
- SCACB (subsystem connection address control block)
- SCCB (subsystem connection control block)
- SCTE (subsystem control table extension)
- SLCB (subsystem logon control block)
- SUDB (subsystem user definition block)
- UCA (use count array)
- URD (unit of recovery descriptor)

PA keyword

- DFHSIT (system initialization table)
- OVERSTOR (override parameter temporary work area)
- PAA (parameter manager domain anchor block)
- PARMSAVE (SIT override parameters)
- PRVMODS (SIT PRVMOD list)
- SITDLI (SIT DL/I extension)

PCP keyword

- PPTTE (program processing table entries)

PCT keyword

- PCTTE (program control table entries)

PG keyword

- PGA (program manager anchor)
- LLE (load list element)–can be system LLE or task LLE
- PGWE (program manager wait element)
- PPTE (program processing table element)
- PTA (program transaction area)
- PLCB (program manager program level control block)
- HTB (handle table)

For an explanation of PG summary data in a level-1 dump, see Appendix B, “Summary data for PG and US keywords” on page 313.

PR keyword

- PRSTATIC (partner resource static area)
- PRTE (partner resource table entries)

RM keyword

- RMCBS (recovery manager control blocks)

SM keyword

- CTN (cartesian tree node)
- DXE (DSA extent list element)
- DXG (DSA extent getmain descriptor)
- DXH (DSA extent list header)
- MCA (SM macro-compatibility control area)
- PAM (page allocation map)
- PPA (page pool control area)
- PPX (page pool extent control area)
- QPF (quickcell page free element)
- QPH (quickcell page header)
- SAE (storage access table entry)
- SAT (storage access table)
- SCA (subpool control area)
- SCE (storage element descriptor)
- SCF (free storage descriptor)
- SMA (storage manager domain anchor block)
- SMSVCTRT (DFHSMSVC trace table)
- SMX (transaction storage area)
- SQE (suspend queue element)
- STAB (storage manager statistics buffer)
- SUA (subspace area)

SSA keyword

- SSAL (static storage address list)
- SSA (static storage areas)

ST keyword

- STANCHOR (statistics domain anchor block)
- STSAFPB (statistics authorization facility parameter block)
- STSMF (statistics SMF record)
- STSTATS (statistics domain statistics record)

SZ keyword

- SZSDS (FEPI static area)

TCP keyword

- ACB (VTAM access method control block)
- AID (automatic initiation descriptor)
- AWE (autoinstall work element)
- BIND (bind image)
- DCB (BSAM data control block)
- DIB (data interchange block)
- EXLST (VTAM ACB exit list)
- ISORM (indirect system object resolution map)
- LOGDS (extracted logon or CLSDST pass data)
- NIB (node initialization block)
- PWE (postponed work element)
- RACE (receive-any control elements)
- RAIA (VTAM receive-any input area)
- RPL (VTAM request parameter list – receive-any RPLs)
- SNTTE (SNT terminal entry)
- TACLE (terminal abnormal condition line entry)
- TCTENIB (NIB descriptor)

- TCTESBA (APPC send/receive buffer)
- TCTFX (TCT prefix)
- TCTLE (TCT line entries)
- TCTTELUC (TCTTE APPC extension)
- TCTME (TCT mode entry)
- TCTSE (TCT system entries)
- TCTSK (TCT skeleton entries)
- TCTTE (TCT terminal entries)
- TCTTETTE (TCTTE extension)
- TCTTEUA (TCTTE user area)
- TIOA (terminal I/O area)
- WAITLST (wait list)
- ZEPD (TC module entry list)

TDP keyword

- ACB (TD VSAM ACB)
- BUFFER (TD I/O buffer)
- DCTE (destination control table entries)
- MBCA (TD buffer control area)
- MBCB (TD buffer control block)
- MQCB (TD queue control block)
- MRCA (TD string control area)
- MRCB (TD string control block)
- MRSD (TD CI state map – segment descriptor)
- MWCB (TD wait control block)
- RPL (TD VSAM RPL)
- SDSCI (TD SCSCI)
- TDST (TD static storage)
- VEMA (TD VSAM error message area)

TI keyword

- TIA (timer domain anchor block)
- TRE (timer request elements)

TMP keyword

- TMSTATIC (table manager static storage)
- SKT (scatter tables)
- DIRSEG (directory segments)
- TM_LOCKS (read lock blocks)

TR keyword

- TRA (trace domain anchor block)
- TRDCB (auxiliary trace dataset DCB)
- TRDECB (auxiliary trace dataset DECB)

TSP keyword

- TSACA (TS auxiliary control area)
- TSBMAP (TS byte map)
- TSBCA (TS buffer control area)
- TSBUFFER (TS I/O buffer)
- TSCOM (TS common area)
- TSGID (TS group identifier)
- TSRE (TS request element)
- TST (TS table header)

- TSTTE (TS table entry)
- TSUT (TS unit table header)
- TSUTE (TS unit table entry)
- TSVCA (TS VSWA control area)

UEH keyword

- DWE (deferred work element)
- EPB (exit program blocks)
- GWA (EPB global work area)
- TIE (task interface element)
- UET (user exit table)
- URD (unit of recovery descriptor)

US keyword

- USA (user domain anchor block)
- USXD (user domain transaction data)
- USUD (user domain user data), one or more of the following:
  - Principal
  - Session
  - EDF

For an explanation of US summary data in a level-1 dump, see Appendix B, “Summary data for PG and US keywords” on page 313.

XM keyword

- XMA (XM domain anchor block)
- TXN (XM domain transaction))
- TCL (XM domain tclass)
- XM_XB (XM domain browse element)
- MXT (XM domain MXT tclass)

XRF keyword

- XRPSTAT (XRP static storage)
- XRP_ACTS (XRP active status area)
- XRP_ALTS (XRP alternate status area)
- XRP_HLTH (XRP health area)
- XRP_XRSA (XRP anchor area)
- CAVM_STA (CAVM static storage)

XS keyword

- XSA (security domain anchor block)
- XSSS (security supervisor storage)

Finding the keywords from the control blocks

CICS Control Block		VERBEXIT Keyword
ACB	VSAM ACBs	FCP
ACB	VTAM access method control block	TCP
ACB	TD VSAM ACB	TDP
AFCB	CICS AFCB	KE
AFCS	CICS AFCS	KE
AFCTE	application file control table element	FCP

CICS Control Block		VERBEXIT Keyword
AFT	CICS AFT	KE
AID	automatic initiation descriptor	TCP
AITM	static storage	AI
AITMTE	autoinstall terminal models	AI
APE	active program element	LD
AUTOSTCK	automatic storage stack entry	KE
AWE	autoinstall work element	TCP
BIND	bind image	TCP
BUFFER	journal buffer	JCP
BUFFER	TD I/O buffer	TDP
CAVM_STA	CAVM static storage	XRF
CC_ACB	local catalog ACB	CC
CC_RPL	local catalog RPLs, one each thread	CC
CCB	connection control block	MRO
CCBUFFER	local catalog buffers, one each thread	CC
CICS24	task storage - below 16MB, CICS key	AP
CICS31	task storage - above 16MB, CICS key	AP
CPE	current program element	LD
CPSTATIC	common program interface storage	CP
CRB	CICS region block	MRO
CSA	common system area	CSA
CTN	cartesian tree node	SM
CSAOPFL	CSA optional features list	CSA
CSB	connection status block	MRO
CSECTL	program CSECT list	LD
CWA	common work area	CSA
CWE	CICS/DBCTL control work element	DLI
DBL	journal control dynamic log buffer	AP
DCB	data control block	FCP
DCB	data control block	JCP
DCB	BSAM data control block	TCP
DCTE	destination control table entries	TDP
DECB	data extent control block	JCP
DFHDLP	CICS/DLI interface parameter list	DLI
DFHJCOCL	journal open/close parameter list	JCP
DFHJCT	journal control table header	JCP
DFHJCTTE	journal table entry	JCP
DFHSERIS	series definition table	JCP
DFHSIT	system initialization table	PA
DGB	CICS/DBCTL global block	DLI
DGBCTA	DBCTL transaction area	DLI
DIB	data interchange block	TCP
DIRSEG	directory segments	TMP
DMANCHOR	domain manager anchor block	DM
DOH	domain table header	KE
DOM	domain table entry	KE
DS_TCB	TCB block	DS
DSANC	dispatcher anchor block	DS
DSB	CICS/DBCTL scheduling block	DLI
DSNB	dataset name block	FCP
DFHDTTABLE	data table base area	FCP
DFHDTFILE	data table path area	FCP
DTA	dispatcher task area	DS

CICS Control Block		VERBEXIT Keyword
DFHDTHEADER	data table global area	FCP
DTRGLOBL	data table remote global area	FCP
DUA	dump domain anchor block	DU
# DUBUFFER	transaction dump data set buffer	DU
DWE	deferred work element	MRO
DWE	deferred work element	UEH
DWE	user DWE storage	AP
DXE	DSA extent list element	SM
DXG	DSA extent getmain descriptor	SM
DXH	DSA extent list header	SM
DXPS	CICS-DL/I-XRF anchor block	DLI
EIB	EXEC interface block	AP
EIS	EXEC interface structure	AP
EIUS	EXEC interface user structure	AP
EPB	exit program blocks	UEH
EXLST	VTAM ACB exit list	TCP
FBWA	data table browse area	FCP
FCSTATIC	FCP static storage - anchor block	FCP
FCTE	file control table element	FCP
FILE	user file storage	AP
FREECHAI	LM domain freechain 1	LM
FREECHAI	LM domain freechain 2	LM
FREECHAI	LM domain freechain 3	LM
GC_ACB	global catalog ACB	CC
GC_RPL	global catalog RPLs, one each thread	CC
GCBUFFER	global catalog buffers, one each thread	CC
GWA	EPB global work area	UEH
HTB	handle table	PG
ICE	interval control elements/AIDs	ICP
ISB	interface scheduling block	DLI
ISORM	indirect system object resolution map	TCP
JAB	journal archive anchor block	JCP
JACD ACB	journal archive control dataset ACB	JCP
JACD RPL	journal archive control dataset RPL	JCP
JAW	journal archive work element	JCP
JCA	journal control area	AP
KCB	kernel anchor block	KE
KERNSTCK	kernel linkage storage stack entry	KE
KERRD	kernel error data	KE
KTCB	KTCB table entry)	KE
LACB	logon address control block)	MRO
LCB	logon control block)	MRO
LD_GLBL	loader domain global storage - anchor block	LD
LDBE	loader domain browse element	LD
LDWE	loader domain wait element	LD
LECBS	logical ECBs	JCP
# LLA	load list area	LD
# LLE	load list element - can be system LLE or task LLE	PG
LMANCHOR	lock manager domain anchor block	LM

	CICS Control Block		VERBEXIT Keyword
	LMQUICK1	LM domain quickcell 1	LM
	LMQUICK2	LM domain quickcell 2	LM
	LMQUICK3	LM domain quickcell 3	LM
	LOCK_ELE	LM domain lock element	LM
	LOGDS	extracted logon or CLSDST pass data	TCP
	LXA	LX array	MRO
	MAPCOPY	user BMS MAP storage	AP
	MBCA	TD buffer control area	TDP
	MBCB	TD buffer control block	TDP
	MCA	SM macro-compatibility control area	SM
	MCT	monitoring control table	MN
	MEA	message domain anchor block	ME
	MNA	monitor domain global storage - anchor block	MN
	MNAFB	monitor authorization facility parameter block	MN
	MNCONNS	monitor field connectors	MN
	MNDICT	monitor dictionary	MN
	MNEVE	SYSEVENT record buffer	MN
	MNEXC	exception record buffer	MN
	MNEXLIST	user EMP address list	MN
	MNFLDMAP	excluded/included CICS field map	MN
	MNPER	performance data buffer	MN
	MNSMF	SMF record buffer	MN
	MNTMA	transaction monitoring area	MN
	MNWLMPB	MVS WLM performance blocks	MN
	MQCB	TD queue control block	TDP
	MRCA	TD string control area	TDP
	MRCB	TD string control block	TDP
	MRSD	TD CI state map - segment descriptor	TDP
	MWCB	TD wait control block	TDP
|	MXT	XM domain MXT tclass	XM
	NIB	node initialization block	TCP
	OPENBLOK	transaction dump open block	DU
	OVERSTOR	override parameter temporary work area	PA
	PAA	parameter manager domain anchor block	PA
	PAM	page allocation map	SM
#	PAPL	DL/I-DRA architected parameter list	DLI
	PARMSAVE	SIT override parameters	PA
	PCTTE	program control table entries	PCT
	PGA	program management anchor	PG
	PGWE	program management wait element	PG
#	PLCB	program manager program level control block	PG
#	PPA	page pool control area	SM
	PPTE	program processing table element	PG
	PPTTE	program processing table entries	PCP
#	PPX	page pool extent control area	SM
	PRSTATIC	partner resource static area	PR
	PRTE	partner resource table entries	PR
	PRVMODS	SIT PRVMOD list	PA
	PST	partition specification table	DLI
	PTA	program transaction area	PG
	PWE	postponed work element	TCP

	CICS Control Block		VERBEXIT Keyword
	QCA	queue control area	AP
	QEA	queue element area	AP
	QPH	quickcell page header	SM
#	QPF	quickcell page free element	SM
	RACE	receive-any control elements	TCP
	RAIA	VTAM receive-any input area	TCP
	RMCBS	recovery manager control blocks	RM
	RPL	VTAM request parameter list - receive-any RPLs	TCP
	RPL	TD VSAM RPL	TDP
	RSB	remote scheduling block	DLI
	SAE	storage access table entry	SM
	SAT	storage access table	SM
	SCA	subpool control areas	SM
	SCACB	subsystem connection address control block	MRO
	SCCB	subsystem connection control block	MRO
	SCD	system contents directory	DLI
	SCE	storage element descriptor	SM
	SCF	free storage descriptor	SM
	SCTE	subsystem control table extension	MRO
	SDSCI	TD SCSCI	TDP
	SDTE	system dump table elements	DU
	SHRCTL	shared LSRPOOLs	FCP
	SITDLI	SIT DL/I extension	PA
	SKT	scatter tables	TMP
	SLCB	subsystem logon control block	MRO
	SMA	storage manager domain anchor block	SM
#	SMSVCTRT	DFHSMSVC trace table	SM
	SMX	transaction storage area	SM
	SNTTE	SNT terminal entry	TCP
	SQE	suspend queue element	SM
	SSA	static storage areas	SSA
	SSAL	static storage address list	SSA
#	STAB	storage manager statistics buffer	SM
	STANCHOR	statistics domain anchor block	ST
	STSAFPB	statistics authorization facility parameter block	ST
	STSMF	statistics SMF record	ST
	STSTATS	statistics domain statistics record	ST
	SUA	subspace area	SM
	SUDB	subsystem user definition block	MRO
	SUSPAREA	unformatted SUSPEND_AREAS/TOKENS	DS
#	SYSEIB	system EXEC interface block	AP
#	SYS_TCA	TCA - task control area - system area only	DLI,AP
	SZSDS	FEPI static area	SZ
	TACLE	terminal abnormal condition line entry	TCP
	TAH	task table header	KE
	TAS	task table entry - TASENTRY	KE
	TASK	unformatted DTAs	DS
#	TCA	task control area - user	DLI,AP
	TCH	KTCB table header	KE
|	TCL	XM domain tclass	XM
	TCTENIB	NIB descriptor	TCP
#	TCTESBA	APPC send/receive buffer	TCP
	TCTFX	TCT prefix	TCP

	CICS Control Block		VERBEXIT Keyword
	TCTLE	TCT line entries	TCP
	TCTME	TCT mode entry	TCP
	TCTSE	TCT system entries	TCP
	TCTSK	TCT skeleton entries	TCP
	TCTTE	TCT terminal entries	TCP
#	TCTTELUC	TCTTE APPC extension	TCP
	TCTTETTE	TCTTE extension	TCP
	TCTTEUA	TCTTE user area	TCP
	TD	user transient data	AP
	TDST	TD static storage	TDP
	TDTE	transaction dump table elements	DU
	TIA	timer domain anchor block	TI
	TIE	task interface element	UEH
	TIOA	terminal I/O area	TCP
	TM_LOCKS	read lock blocks	TMP
	TMSTATIC	table manager static storage	TMP
	TRA	trace domain anchor block	TR
	TRDCB	auxiliary trace dataset DCB	TR
	TRDECB	auxiliary trace dataset DECB	TR
	TRE	timer request elements	TI
	TS	user temporary storage	AP
	TSACA	TS auxiliary control area	TSP
	TSBCA	TS buffer control area	TSP
	TSBMAP	TS byte map	TSP
	TSBUFFER	TS I/O buffer	TSP
	TSCOM	TS common area	TSP
	TSGID	TS group identifier	TSP
	TSRE	TS request element	TSP
	TST	TS table header	TSP
	TSTTE	TS table entry	TSP
	TSUT	TS unit table header	TSP
	TSUTE	TS unit table entry	TSP
	TSVCA	TS VSWA control area	TSP
|	TXN	XM domain transaction	XM
	UCA	use count array	MRO
	UET	user exit table	UEH
	URD	unit of recovery descriptor	MRO
	URD	unit of recovery descriptor	UEH
#	USA	user domain anchor block	US
	USER24	task storage - below 16MB, user key	AP
	USER31	task storage - above 16MB, user key	AP
#	USUD	user domain user data - one or more of:	US
#		Principal, Session, EDF	
#	USXD	user domain transaction data	US
	VEMA	TD VSAM error message area	TDP
	VOLINDEX	cyclic index of volume identifiers	JCP
	VSWA	VSAM work area	FCP

	CICS Control Block		VERBEXIT Keyword
	WAITLST	wait list	TCP
	WQP	domain wait queue	DM
	XMA	XM domain anchor block	XM
	XM_XB	XM domain browse element	XM
	XRP_ACTS	XRP active status area	XRF
	XRP_ALTS	XRP alternate status area	XRF
	XRP_HLTH	XRP health area	XRF
	XRP_XRSA	XRP anchor area	XRF
	XRPSTAT	XRP static storage	XRF
#	XSA	security domain anchor block	XS
#	XSSS	security supervisor storage	XS
	ZEPD	TC module entry list	TCP

Appendix B. Summary data for PG and US keywords

This chapter lists the elements of the control block summaries in a level-1 IPCS dump for the PG and US keywords.

PG keyword

The summaries appear below in the sequence in which they appear in a dump. This is broadly the sequence in which the control blocks are listed in Appendix A, “SDUMP contents and IPCS CICS VERBEXIT keywords” on page 299 form=numonly, but note:

- The system LLE summary, if present, follows the PGA summary, but the task LLE summary, if present, follows the PTA summary.
- The HTB does not appear in a summary.

PGA (program manager anchor)

PG Domain Status
One of the following:

- Initializing
- Initialized
- Quiescing
- Quiesced
- Terminating
- Terminated.

Autoinstall status
Either active or inactive.

Autoinstall catlg status
Autoinstall catalog status, one of the following:

- All
- Modify
- None.

Autoinstall exit name
Autoinstall exit name.

Attempted autoinstalls
Number of attempted autoinstalls in decimal.

Failed autoinstalls
Number of failed autoinstalls in decimal.

Rejected autoinstalls
Number of rejected autoinstalls in decimal.

XRSINDI active
Status of user exit, either Y or N.

Exec calls allowed
Either Y or N.

System LLE chain head
Address of system LLE chain head, zero if no chain exists.

PGWE chain head
Address of PGWE chain head, or zero if no chain exists.

Stats last - 1st word
Statistics last reset time using GMT (on two lines).

Reset time - 2nd word
Second part of last reset time.

SM access token
SM access token value.

SM isolation token
SM isolation token value.

Storage protect
Either Y or N.

Cold start
Either Y or N.

Recovery complete
Either Y or N.

System LLE Summary

LLE-ADDR
LLE address.

PROGRAM
Program name.

PPTE-ADD
PPTE address.

PGWE Summary

PGWE-ADD
Address of suspended program.

PROGRAM
Name of suspended program.

SUS-TOKN
Suspend token.

PPTE-ADD
Program PPTE address.

PPTE Summary

PPTE ADDRESS
Address of PPTE block.

PROGRAM NAME
The tables are indexed using the program name.

MOD TYPE
Module type, one of the following:

- PG - Program
- MP - Mapset
- PT - Partitionset.

LANG DEF

Language defined, one of the following:

- NDF - Not defined
- ASS - Assembler (or Ada)
- C - C370
- COB - Cobol
- CO2 - Cobol 2
- LE3 - Le370
- PLI - PL/I.

LANG DED

Language deduced, one of the following:

- NDD - Not deduced
- ASS - Assembler (or Ada)
- C - C370
- COB - Cobol
- CO2 - Cobol 2
- LE3 - Le370
- PLI - PL/I.

INST TYPE

PPTE installation type, one of the following:

- R - Built from RDO
- C - Built from catalog constant
- G - Built from grouplist
- A - Autoinstall
- S - System autoinstall
- M - Manual.

CEDF STAT

CEDF status, either CED(CEDF allowed) or NOC(CEDF not allowed).

AVAL STAT

Program availability status, either E(enabled) or DI(disabled).

DATA LOC

Data location, either A (any location) or B(below 16M).

EXEC KEY

Execution key, either C (CICS) or U (user).

DPL SUBS

DPL subset, either DP (DPLsubset) or F (full API).

RE LOAD

Indicator of whether this is a reload program, either Y or N.

LOAD STAT

Load status, one of the following:

- L - Loaded
- NL - Not loadable
- ND - Not loaded.

HOLD STAT

CICS hold status, either C (loaded for CICS lifetime) or T (task lifetime).

USE COUNT
Use count in decimal, blank if 0.

LOCK OWNER
Transaction number of locking program.

PGWE CHAIN
Indicator of presence of any PGWEs, either Y or N.

REMOTE PRGID
Remote program name.

REMOTE SYSID
Remote system name.

REMOTE TRNID
Remote transaction name.

PTA Summary

TRAN NUM
Transaction number.

PTA ADDRESS
Address of PTA.

LOG-LVL
Logical level count in decimal.

SYS-LVL
System level count in decimal.

TASK-LLE
Address of task LLE head, zero if no task LLE exists.

PLCB
Address of PLCB head, or zero if no PLCB exists.

Task LLE Summary

LLE-ADDR
LLE address.

PROGRAM
Program name.

PPTE-ADD
PPTE address.

Task PLCB Summary

PLCB-ADD
PLCB address.

PROGRAM
Program name.

LOG-LVL
Logical level of program.

LOAD
Program load point.

ENTRY
Program entry point.

LENGTH
Program length.

CA-CURR
Current commarea address.

CLEN
Current commarea length.

INVK-PRG
Name of invoking program.

STG
Commarea storage class. Can be one of five:

- Blank - No commarea for this level.
- C - CICS
- C24 - CICS 24 bit
- U - User
- U24 - User 24 bit.

EXIT-NME
Exit name derived from user exit number, if applicable.

ENV
Environment type, one of the following:

- EXEC - Command level application
- GLUE - Global user exit
- PLT - PLT program
- SYS - CICS system program
- TRUE - Task-related user exit
- URM - User-replaceable program.

PPTE-ADD
Program PPTE address.

US keyword

A level-1 dump summarizes only the user domain data (USUD). The fields displayed are the same for each type of USUD (principal, session, or EDF).

USXD summary

TRAN NUM
Transaction number.

PRINCIPAL TOKEN
Principal token, if any.

SESSION TOKEN
Session token, if any.

EDF TOKEN
EDF token, if any.

USUD summary

TOKEN
User token.

USERID
User identifier.

GROUPID
Group identifier.

ADDCOUNT
Adduser use count.

TRNCOUNT
Transaction use count.

OPID
Operator identifier.

CLASSES
A bitmap expressing the operator classes in order 24 to 1.

PRTY
Operator priority.

TIMEOUT
Timeout interval in hours and minutes (hh:mm).

ACEE
Address of ACEE.

XRFSOFF
XRF user signon. Can be NOFORCE or FORCE.

USERNAME
User name.

Index

A

B

C

D

E

F

G

H

I

M

N

O

P

Q

R

S

T

U

V

W

X

Z

Sending your comments to IBM

CICS/ESA

Problem Determination Guide

SC33-1176-01

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
  - From outside the U.K., after your international access code use 44 962 870229 (after 16 April 1995, use 44 1962 870229)
  - From within the U.K., use 0962 870229 (after 16 April 1995, use 01962 870229)
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: WINVMJ(IDRCF)
  - Internet: idrcf@winvmj.vnet.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

CICS/ESA

Problem Determination Guide

SC33-1176-01

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate without incurring any obligation to you.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

1 Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____

Company or Organization _____

Address _____

_____

EMAIL _____

Telephone _____

1 Cut along this line

4 Fasten here with adhesive tape

Program Number: 5655-018

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1176-01