IBM

CICS

CICS

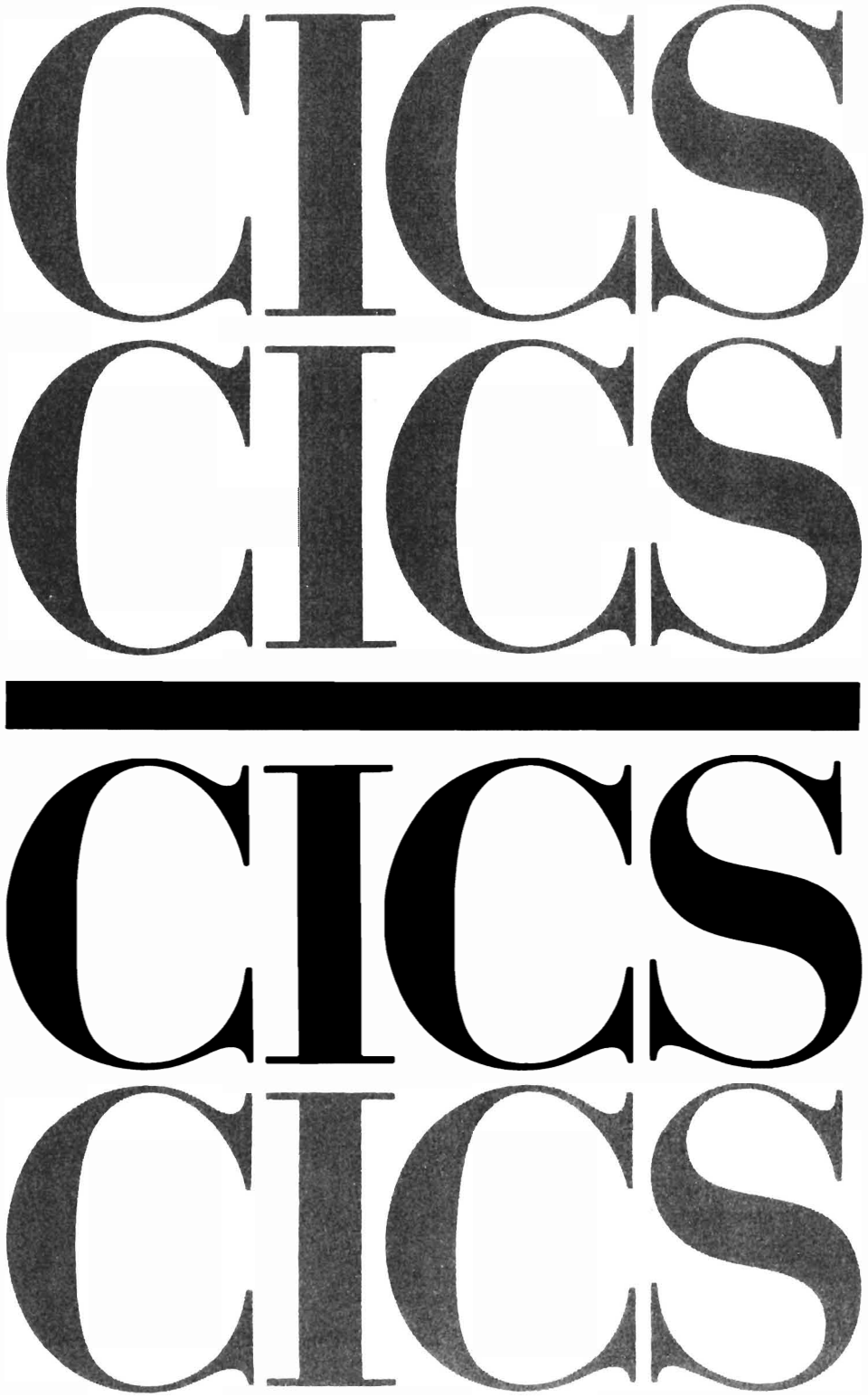
CICS

CICS

CICS

CICS/CMS User's Guide

IBM



Customer
Information
Control System
CICS/CMS

Licensed Program
Release 1

Program Number
5668-795

CICS/CMS User's Guide

SC33-0285-0

First Edition (June 1986)

This edition applies to Release 1 of the IBM program product Customer Information Control System/Conversational Monitor System (CICS/CMS), program number 5668-795.

Changes are periodically made to the information herein; before using this publication in connection with the operation of IBM systems or equipment, refer to the latest *IBM System/370, 30XX, and 4300 Processors Bibliography*, GC20-0001, for the editions that are applicable and current.

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 program product in this publication is not intended to state or imply that only IBM's program product may be used. Any functionally equivalent program may be used instead.

Publications are not stocked at the addresses given below. Requests for IBM publications should be made to your IBM representative or to the IBM branch office serving your locality.

A form for reader's comments is provided at the back of this publication. If the form has been removed, comments may be addressed either to:

International Business Machines Corporation, Department 6R1H,
180 Kost Road, Mechanicsburg, PA 17055, USA.

or to:

IBM United Kingdom Laboratories Limited, Information Development,
Mail Point 95, Hursley Park, Winchester, Hampshire, England, SO21 2JN

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

Preface

What this book is about

This book tells you how to use CICS/CMS to develop CICS applications. As its name implies, CICS/CMS runs under the Conversational Monitor System (CMS), and can be used by both CICS/OS/VS and CICS/DOS/VS application programmers.

Who this book is for

This book is for anyone who wants to develop CICS applications using CICS/CMS, and for those responsible for administering application developers' use of CICS/CMS.

What you need to know to understand this book

This book is primarily intended for application programmers with some experience of both CICS and CMS. However, if you have just a basic knowledge of CMS, you should be able to use CICS/CMS, with this book and the *CICS/VS Application Programming Primer*, to learn about CICS application programming.

Those parts of the book written for system administrators, that is, those responsible for installing and maintaining the CICS/CMS system, assume a much more detailed knowledge of VM and CMS.

How to use this book

Part One of this book is designed to be read sequentially by those using CICS/CMS for the first time. It provides enough information for you to start using CICS/CMS without having to learn about its functions in detail. The rest of the book is intended largely for reference. However, we've arranged the chapters so that they describe CICS/CMS facilities in the order you're most likely to need to learn them, so that you can read the whole book sequentially if you want.

Notes on terminology

CICS/VS	is used to mean both CICS/OS/VS and CICS/DOS/VS.
PC	is used to mean a PC AT/370 or PC XT/370.
COBOL	is used for the OS/VS COBOL Compiler, and the language it supports.
COBOL II	is used for the VS COBOL II Compiler, and the language it supports.

PL/I is used for the OS/VS PL/I Optimizing Compiler, and the language it supports.

EFH is a prefix, used to denote elements that belong to CICS/CMS (EXECs, utilities, and so on). It's the CICS/CMS equivalent of the DFH prefix, that denotes CICS/VS elements.

Whenever we use the term "CICS", on its own, without appending "/VS" or "/CMS", we are either talking about the part of CICS/CMS used in application testing, or about CICS in general terms, as in "CICS application programmer."

Bibliography

This bibliography gives the titles and order numbers of other books that you will find useful when using CICS/CMS.

We've listed the books in categories, as follows:

- CICS/CMS manuals
- CICS/VS manuals
- VM/SP manuals
- VM/PC manuals
- Other manuals.

CICS/CMS manuals

CICS/CMS Application Programmer's Reference Summary, SX33-6051

Provides a quick reference to the subset of the command level application programming interface (API) that CICS/CMS supports.

CICS/CMS Messages and Codes, SC33-0409

Lists all the messages that CICS/CMS issues that are unique to CICS/CMS. This may be of use to application programmers, but the majority of CICS/CMS messages that you're likely to see while developing applications are self-explanatory, so you shouldn't need it for constant reference. System administrators, however, will need it as part of their basic set of CICS/CMS documents to deal with problems referred to them by application programmers.

CICS/VS manuals

CICS/VS Application Programming Primer, SC33-0139

The best book for learning CICS application programming. You'll need it to understand fully the sample application that comes with CICS/CMS.

CICS/VS Application Programmer's Reference Manual (Command Level), SC33-0241

The definitive description of CICS application programming. You'll need to use this with the *CICS/CMS Application Programmer's Reference Summary*, SX33-6051 if you want to use only features that CICS/CMS supports.

CICS/VS Application Programmer's Reference Summary, SX33-6012

The CICS/VS equivalent of the CICS/CMS summary.

CICS/OS/VS CICS-Supplied Transactions, SC33-0240

Contains descriptions of all the CICS-supplied transactions, including the field engineering transaction (CSFE), and the operator commands for BMS terminal paging (CSPG transaction).

Note: IBM introduced this manual with CICS/OS/VS 1.7. Before then, the same information was in the *CICS/VS Operator's Guide*, SC33-0080.

CICS/VS Remote Server Diagnosis, LC33-0438

Presents an outline logic flow of the CEHS transaction (the remote server) that is supplied as part of CICS/OS/VS V1 R7, and that is also available on both CICS/OS/VS V1 R6.1 and CICS/DOS/VS V1 R6.

This book is primarily intended for system programmers interested in problems with the CICS/CMS remote server at either end of the link.

CICS/VS Resource Definition Guide, SC33-0149

Includes descriptions of the TCT entries that relate to terminal control parameters in the EFHSETP and EFHPROF EXECs.

CICS/VS Problem Determination Guide, SC33-0089

Gives detailed information on debugging CICS/VS programs.

CICS/VS Data Areas, LY33-6035

Describes the CICS/VS data areas. You may need to refer to this manual occasionally for help in understanding CICS/CMS messages and trace entries.

VM/SP manuals

VM/SP CMS Primer, SC24-5236

A useful starting point for anyone who is completely new to VM.

VM/SP CMS User's Guide, SC19-6210

The basic reference book for using VM/SP.

VM/SP CMS Command and Macro Reference, SC19-6209

Essential for system administrators to help diagnose problems.

VM/SP CP Command Reference for General Users, SC19-6211

Includes information on the CP debugging tools available with CICS/CMS.

VM/SP System Programmer's Guide, SC19-6203

Provides detailed information on CP and CMS, and tells you how to debug VM/SP. This book is an essential part of a system administrator's set of documentation.

VM/SP System Messages and Codes, SC19-6204

Lists all the CMS messages, many of which might appear while you're developing applications using CICS/CMS.

VM/SP Installation Guide, SC24-5237

The system administrator's main source of information for CMS aspects of installing and servicing CICS/CMS on VM/SP.

VM/SP Terminal Reference, GC19-6206

Includes information on the 3270 terminals that VM/SP supports.

VM/PC manuals

VM/PC User's Guide, SC26-0601

Contains basic instructions and reference information on how to use VM/PC.

VM/PC System/370 Language Supplement, SC26-4120

Contains information specific to programming under VM/PC.

Other manuals

CMS User's Guide for COBOL, SC28-6469

Contains information specific to writing OS/VS COBOL programs under CMS.

VS COBOL II Application Programming: Supplement for CMS User's, SC26-4214

Contains information specific to writing VS COBOL II programs under CMS.

OS/PLI Optimizing Compiler: CMS User's Guide, SC33-0037

Contains information specific to writing PL/I programs under CMS.

Contents

Figures

Questionnaire CICS/CMS User's Guide

(CICS/CMS Release 1)

To help us produce books that meet your needs, please fill in this questionnaire. It would help us if you provide your name and address in case we need to clarify any of the points you raise. Please understand that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you.

1. Please rate the book on the points shown below

   The book is:

accurate	1	2	3	4	5	inaccurate
readable	1	2	3	4	5	unreadable
well laid out	1	2	3	4	5	badly laid out
well organized	1	2	3	4	5	badly organized
easy to understand	1	2	3	4	5	incomprehensible
adequately illustrated	1	2	3	4	5	inadequately illustrated
has enough examples	1	2	3	4	5	has too few examples

   And the book as a whole?

excellent	1	2	3	4	5	poor

2. When using this book, did you find what you were looking for? _____

   What were you looking for? _____

   What led you to this book? _____

   Did you come straight to this book? _____

3. Which topics does the book handle well?

   _____

   _____

   _____

   _____

   _____

4. And which does it handle badly?

   _____

   _____

   _____

   _____

   _____

5. How could the book be improved? _____

   _____

   _____

   _____

   _____

6. How often do you use this book?

   Less than once a month? ☐ Monthly? ☐ Weekly? ☐ Daily? ☐

7. What sort of work do you use CICS for? _____

   _____

8. How long have **you** been using CICS? ____ years/months

9. Have you any other comments to make?

   _____

   _____

   _____

Thank you for your time and effort. No postage stamp necessary if mailed in the USA. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to either address in the Edition Notice on the back of the title page.)

Note: Staples can cause problems with automated mail-sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.

SC33-0285-0

Questionnaire

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Name .....

Job Title ..... Company .....

Address .....

..... Zip .....

Part one—First steps

This Part of the book gives you:

- A general introduction to CICS/CMS
- An opportunity to see CICS/CMS in action.

Chapter 1. Introduction to CICS/CMS

Until now, developing CICS application programs has involved scheduling access to a CICS/VS test system, preparing updates to tables and, sometimes, switching between different systems to edit, translate, compile, and test CICS programs and maps. Some of this work has also involved wrapping job control language (JCL) statements round programs, and submitting them to batch queues.

CICS/CMS changes all that, by providing a CICS system specifically designed to let you develop CICS applications interactively. In a single environment, you can create, edit, translate, compile, and execute your programs, and use the diagnostic facilities of both CICS and CMS to check them out. CICS/CMS will run on a wide range of IBM machines, *including* the PC/370.

Who is this book for?

The main aim of this *User's Guide* is to tell application programmers all about CICS/CMS. The ***you*** in the guide is therefore almost always an application programmer. However, the guide also contains information needed by the person responsible for administering a CICS/CMS system (that is, installing it, tailoring it to local requirements, and providing assistance when the application programmers need it). This person may be a system programmer, or a senior application programmer. Throughout this guide, we call him or her the ***system administrator***. We make sure you know when we're talking to the system administrator, rather than to the application programmers.

We should also say something about the level of experience you'll need to use this guide.

As its name implies, CICS/CMS runs under the Conversational Monitor System (CMS) within VM. To understand this guide, and use CICS/CMS, you'll need to know something about CMS and, in particular, about the way it handles files.

System administrators will need to know a lot more than just "something." They'll need to be experienced VM users, who are also familiar with CICS and, in particular, with CICS application programming.

If you're using CICS/CMS on a PC/370, you'll need to know something about the general operation of the PC and about its VM system, VM/PC.

You don't, however, have to be a CICS expert to understand this guide as an application programmer. It tries to give an overview of every topic it raises and, where appropriate, refers less experienced users to introductory information. You couldn't learn to become a CICS programmer using just CICS/CMS and this guide. But we think you'll find the guide useful if you're using CICS/CMS as part of your CICS/VS education.

If you are the system administrator, you will need to have quite a bit of experience of both CICS and CMS to use effectively those parts of the guide that are written for you.

What can you use CICS/CMS for?

You can use CICS/CMS to develop CICS application programs easily. The rest of this section will add a few details to that simple statement, but it's the most accurate answer to the question above. Two aspects of CICS/CMS, in particular, make for easy program development:

1. CICS/CMS gives you all the facilities you need in a single-user environment.

2. The CICS/CMS environment has been designed for one task: developing CICS applications.

If we look briefly at the way that you go about developing applications using CICS/CMS, you'll see how it makes your work easier.

Creating and editing programs

As soon as you've got CMS going, you can start CICS/CMS with a single command. The first thing CICS/CMS does is display a menu on your screen. One of the options on the menu lets you list all the files associated with your current application development. The application files (programs, maps, data files, and so on) are held on CMS disks; the disks you need can be set up in your profile when you start CMS.

Using a standard CMS editor, such as XEDIT, you can then go on to create a new program, or edit an existing one.

Translating and compiling programs

CICS/CMS lets you assemble your maps, and translate and compile your programs, without adding any JCL to your source, and without leaving CICS/CMS.

CICS/CMS writes the listings from the assemblies, translations, and compilations to easily-identifiable files, and stores these on a temporary CMS disk. You can check immediately for errors in any stage, make any necessary corrections, and try again.

Testing a program

With CICS/CMS, you can go straight from a successful compilation to testing under CICS, with ***no preparation at all***. You don't have to link-edit your programs. You don't have to move your programs, maps, or data to a CICS/VS system; you've already got one. And you don't have to ask anyone to update tables for you. If you need to define resources to CICS/CMS, you can do so yourself but, as you'll see, you don't often need to do so. When you do, you just edit a CMS file.

You execute a CICS program by asking for the CICS/CMS execution panel, entering the program name, and pressing the ENTER key. The word "program" is important. You don't have to put together a complete application before you start

testing. If (as is likely) your application consists of many different programs, you can test each one individually before you put them all together and test the complete application.

If the program asks for data that isn't there, or links to another program that you haven't yet written, CICS/CMS will complain. In a preliminary test of this kind, however, you probably want to test only the logic of the program and the correctness of EXEC CICS commands. By using the execution diagnostic facility (EDF), you can simulate the program's behavior in a working environment. You can invoke EDF by pressing a single PF key before (or during) the execution of your program.

At some stage, you will want to test your application in more realistic conditions. At that point, you will have to do a few things before you execute the application. Let's look at the preparation you'll need to do.

How to use data files

Most CICS programs use some form of data, usually VSAM files or DL/I data bases. You can use either from CICS/CMS.

To make it as easy as possible for you to create and maintain ***local*** data files, CICS/CMS uses special CMS files that simulate VSAM files. You don't have to create these files from scratch; CICS/CMS provides utilities to convert existing files to the form it requires. One of the utilities converts existing local CMS files into CICS/CMS pseudo-VSAM files; the other is a more general utility for copying and converting files. Its primary purpose is to convert VSAM files, stored on a remote CICS/VS system, into CICS/CMS pseudo-VSAM files, and store those files on a CMS disk of your choice. However, you can also use it the other way round; to send data files, that you've created using CICS/CMS, to a remote CICS/VS system as VSAM files.

Once you've set up these pseudo-VSAM files on a CMS disk, there's nothing else you have to do before using them from your program. You don't need to create or change a file control table to use local files.

CICS/CMS will also let you use data files stored on a remote CICS/VS system. If you want to use DL/I data bases, you can't use local versions, because CMS does not support DL/I. CICS/CMS uses a feature called the ***remote server*** to let you test programs containing EXEC DL/I calls. The same feature lets you access VSAM files on a remote CICS/VS system directly, rather than using local pseudo-VSAM copies.

If your program requests a resource that you have defined as being remote, the remote server passes that request to your defined remote CICS/VS system. There, the remote server transaction, CEHS, executes the request and ensures that any data or messages get back to the program that made the request.

Before you try to use DL/I data bases or remote VSAM files, you have to ensure that your program can get at them. You can:

1. Define the resource as being remote, in a CICS/CMS table.

   Whenever you need to define a resource in CICS/CMS, you change a CMS file that contains simple definitions of the resource you want. We refer to these files as tables, since they serve the same purpose as CICS/VS control tables. As you'll see in Chapter 5, "How CICS/CMS handles CICS resources," CICS/CMS

table files are easy to understand and easy to use. Most importantly, they let you control the resources your programs need, without consulting anyone else.

2. Establish the link to the remote CICS/VS system, by connecting to that system and running the remote server transaction, CEHS.

As well as DL/I data bases and VSAM files, you can also access remote temporary storage and transient data queues, and use interval control remotely.

Defining programs to CICS/CMS

We mentioned earlier that you can test a single program without having to change a program control table (PCT), or anything like it. However, to test a complete application, consisting of several programs, perhaps associated with more than one transaction ID, you will have to do a little preparation. CICS/CMS has to be told what the associations between the programs and transactions are. You don't have to write any macros or use resource definition online to do this, or ask anyone else to do it for you. You just edit a CICS/CMS table file, containing program definitions.

Help with your testing

CICS/CMS provides the usual aids for testing CICS programs, and introduces some interesting new features of its own. The CICS/VS aids it provides are:

- The execution diagnostic facility (CEDF transaction) to step through the program, command by command
- The command level interpreter (CECI transaction) to check CICS statement syntax and/or execute CICS statements one-by-one conversationally
- The temporary storage browse facility (CEBR transaction) to check the contents of temporary storage or transient data queues.

You can also use CP or CMS debugging aids, such as:

- CMS DEBUG to examine the contents of storage, registers, or the program status word (PSW)
- CP PER to monitor events in your virtual machine as your program executes
- CP ADSTOP to suspend a program's execution at specified points
- CP STORE to change the contents of registers or storage locations.

Among the new facilities that CICS/CMS provides are:

- The EFHUMAP utility to display the addresses of CICS/CMS modules and CICS control blocks
- The EFHUSTG utility to trace the use of storage by a CICS application program.

You'll find information on these, and other CICS/CMS testing and debugging aids, in Chapter 15, "Debugging tools."

When things go wrong, you'll find that CICS/CMS presents error messages in a way that is in keeping with its interactive approach. Each error message is shown on a separate panel. The panel identifies the source of the error and gives a description of the nature and, where possible, the likely cause of the error. Usually, you should be able to diagnose errors without referring to a manual.

Perhaps the most useful testing feature of all, however, exploits the fact that you've got both CICS and CMS available to you at the same time. While you've got CICS running, you can bring up an *escape panel*, at any time that CICS is waiting for you to enter something at your terminal.

The escape panel offers a variety of services, including access to the CICS/VS testing aids, and access to the CMS subset. You can suspend a program in the middle of a test, and do a variety of useful things. You can use CECI to check the validity of a CICS statement, use CEBR to check that the program is writing to temporary storage correctly, use commands in the CMS subset to inspect or even edit files, and so on.

When you have finished doing what you want from the escape panel, you can press a single key to return to the exact point from which you escaped.

What can't you use CICS/CMS for?

You can't use CICS/CMS to write or test applications written using the CICS/VS macro level application programming interface (API). CICS/CMS only supports programs written using the command level API.

Apart from that, you can use CICS/CMS to test most of the features of CICS that you could test on a CICS/VS system. There are, however, a few limitations on what CICS/CMS can do. Most of these are imposed by the nature of CICS/CMS; by the fact that it is a single-user system.

For example, CICS/CMS supports only one interactive terminal. This means that it cannot execute an EXEC CICS command such as `ROUTE`, which sends a message to an identified terminal or terminals.

What does CICS/CMS do with such commands when it finds them in your programs?

Usually, it displays a message telling you that you have used an unsupported feature, assumes that the command has executed correctly, and passes on to the next statement in the program. You can check the correctness of all such commands using CEDF or CECI. However, you won't know for certain that they are going to do what you expect until you move your application from CICS/CMS to your CICS/VS test system.

These limitations are few and, generally, you can be pretty certain that a program that works on CICS/CMS will work on CICS/VS. But we must emphasize that CICS/CMS is a useful addition to, not a replacement for, your existing application development setup.

A list of the main differences between the CICS that CICS/CMS supports, and the CICS that CICS/VS supports, is given in Appendix B, "Differences Between CICS/CMS and CICS/VS."

The other main restriction that CICS/CMS imposes is on the terminals you can use. CICS/CMS only supports PC/370s, and the 3270 terminals that VM/SP supports. You'll find a list of these in the *VM/SP Terminal Reference* manual.

Note also that, if you're using a PC, and you need to use the remote server to gain access to resources on a remote CICS/VS system, your PC must have the 3278/3279 emulation adapter, not the 3277 emulation adapter.

Chapter 2. Getting to know CICS/CMS

Before we start describing CICS/CMS in detail, we'll give you a chance to see it in action, using the sample application from the *CICS/VS Application Programming Primer*.

We'll start by telling you how to get CICS/CMS going, then run you through the steps of entering, translating, compiling, and testing a CICS program. We won't give too much explanation. One of the first things you'll notice is that CICS/CMS doesn't need much; it's very easy to use.

The main aim of this chapter is to give you a broad outline of what CICS/CMS can do, before we start to fill in the details in "Part two—Before you start developing applications."

Notes

1. The sample application is written in OS/VS COBOL. If you don't have access to the OS/VS COBOL compiler or its libraries, you won't be able to try most of the functions described in this chapter. It's still worth reading it, however, to get a general idea of the flow of application development using CICS/CMS.

   In "Checking for correct installation" on page 206, we describe a set of EXEC CICS commands for the system administrator to use to check for correct installation of CICS/CMS. You'll probably find it instructive to go through that set.

2. CICS/CMS lets you use program function (PF) keys for most of its operations. The CICS/CMS panels only show definitions for PF keys 1 to 12. If you are using a terminal with 24 PF keys, you can use PF keys 13 to 24 for the operations defined for keys 1 to 12. For example, to get help on any panel, you can press either PF1 or PF13.

3. Throughout this chapter (indeed, throughout most of this guide), we assume the following:

   - That you're using CICS/CMS with the IBM-supplied settings for PF keys. For example, on the CICS/CMS panels where you can invoke a CMS editor using a PF key, we always assume that the key is PF6.

   - That you're using the IBM-supplied filemode (Z) for the CICS/CMS temporary disk.

   If you notice any difference between what we describe in this chapter and what happens when you try it, ask your system administrator whether he or she has modified CICS/CMS.

Getting ready to use CICS/CMS

Before you can start using CICS/CMS from either VM/SP or VM/PC, your system administrator has to install it on the host VM/SP system. We tell the system administrator how to do this in "Installing CICS/CMS on a VM system" on page 204.

As soon as CICS/CMS is installed on the host VM system, host-connected terminal users can start using it, as explained below.

PC users who want to use CICS/CMS locally, however, will need to copy the master version, stored on the host system, to their hard disks. Chapter 3, "Installing CICS/CMS on a PC" on page 37 tells you how to do this. Once you've installed CICS/CMS on your PC, you can continue with this chapter, starting at "Getting ready to use CICS/CMS from a PC" on page 11.

As explained there, however, this process can take a long time. If you want to get on with this chapter immediately, and you're using a PC, you might like to connect to your master version of CICS/CMS directly, using your PC in 3278/3279 emulation mode.

Getting ready to use CICS/CMS from a terminal

There are six steps in getting ready to use CICS/CMS, as follows:

1. Switch your terminal on.
2. Log on to VM.
3. IPL CMS.
4. Link to the system disk where your system administrator has installed CICS/CMS.
5. Link to the system disk containing the compilers and libraries for the languages you want to use. You can leave out this step if they're on the same disk as CICS/CMS.
6. Issue `GLOBAL` commands for the libraries that you need.

It's more than likely that your systems people have set up your logon process so that steps 3, 4, 5, and 6 are done automatically. If not, you type:

```
IPL CMS
```

for the third step; appropriate CMS `LINK` and `ACCESS` commands for the fourth and fifth steps; and `GLOBAL` commands for the sixth step. We can't tell you here what the exact form of those commands will be, but someone at your site will tell you.

Having taken those six steps, you're ready to get CICS/CMS going, as described in "Starting CICS/CMS" on page 11.

Getting ready to use CICS/CMS from a PC

We're assuming here that you've already installed VM/PC, CICS/CMS, and the compilers and libraries you need, on your hard disk. If you haven't, read Chapter 3, "Installing CICS/CMS on a PC" on page 37, before continuing.

Getting ready to use CICS/CMS couldn't be simpler. Just follow these few steps:

1. Switch on your PC.
2. Once it has loaded DOS, type the date and time.
3. Type `VMPC` to start the version of VM that runs on PCs.
4. Select the local session (option 2) on the VM/PC session selection menu.
5. Log on to VM/PC, unless your VM/PC profile is set up to do so automatically.
6. Type:

   ```
   IPL CMS
   ```

   to start CMS. In fact, you might be able to leave this step out, if your VM profile is set up to start CMS automatically.

You are now ready to start CICS/CMS, as explained in the next section.

Starting CICS/CMS

To get CICS/CMS running, on a PC or a host-connected terminal, type:

```
CICSCMS
```

CICS/CMS will display its initial panel on your screen, as shown in Figure 1.

```
EFH1                CICS/CMS PROGRAM DEVELOPMENT SELECTION

 Select one of the following:

                  1   List Application Files
                  2   Execute Program/Transaction
                  3   List Resource Tables
                  4   Convert CMS file to CICS/CMS file
                  5   Erase Temporary Files
                  6   Release CICS/CMS Nucleus Extensions
                  7   Apply Service

          Selection ===> 1

 For application filelist specify the optional list criteria required:

          Criteria  ===> ACCT* * *


             (c) Copyright IBM Corp 1985

PF1=Help          PF3=End          PA2=Enter CMS Subset
```

Figure 1. CICS/CMS program development selection panel (EFH1)

Notes:

1. Figure 1 shows the options available to the terminal user. If you're a PC user, your option 7 reads:

   ```
   Download from Host
   ```

   which is how you put your own copy of CICS/CMS on your hard disk.

2. As shown in Figure 1, option 7 for host-connected terminal users says `Apply Service`. No application programmer ever needs to use this option. It's there for the system administrator to make changes to the master copy of CICS/CMS, as explained in "Applying service to CICS/CMS" on page 214.

The sample application

If you select option 1 on panel EFH1 (just press ENTER), you will get a new panel, showing all the files associated by the `Criteria` selected on EFH1. The CICS/CMS system that we supply has the criteria preset to:

filename `ACCT*`

Selects all files with filenames beginning with ACCT.

filetype *

Selects files of any filetype.

filemode *

Selects files on any of the disks to which you currently have access.

Make sure that the `Selection` and `Criteria` lines are the same as shown in Figure 1. Press ENTER, and you will see the display shown in Figure 2.

Note: The filemode of the files in Figure 2 will be the letter associated with whatever disk your system administrator has set your CICS/CMS system disk to be. In all the examples in this chapter, we assume that the CICS/CMS system disk is the B-disk, but it may well be different in your installation.

```
 EFH11                FILELIST:ACCT* * *                      Line 1 of 10

Cmd    Filename Filetype Fm Format Lrecl     Records     Blocks   Date     Time
       ACCT03   COBOL    B2 F          80          70          2  3/01/86 15:43:14
       ACCT04   COBOL    B2 F          80         144          3  3/01/86 15:43:14
       ACCT02   COBOL    B2 F          80         378          8  3/01/86 15:43:13
       ACCT01   COBOL    B2 F          80         371          8  3/01/86 15:43:12
       ACCTSET  ASSEMBLE B2 F          80         171          4  3/01/86 15:43:11
       ACCT00   COBOL    B2 F          80          17          1  3/01/86 15:43:11
       ACCTFIL  EFHVDATA B2 F         383           2          1  3/01/86 15:43:09
       ACCTIX   EFHVDATA B2 F          63           2          1  3/01/86 15:43:09
       ACCTIX   EFHVINDX B2 F         504           2          1  3/01/86 15:43:09
       ACCTFIL  EFHVINDX B2 F         504           2          1  3/01/86 15:43:08




PF1=Help      2=Refresh 3=End       4=All this Name 5=Translate/Compile 6=Edit
PF7=Backward 8=Forward 9=Install  10=MAP-TEXT&ADS 11=Execute          12=

====>
```

Figure 2. CICS/CMS application objects panel (EFH11)

Those of you who have studied the *CICS/VS Application Programming Primer* will recognize the filenames in Figure 2 immediately. They are the program, map, and data files for the sample application in that book. If you don't know the *Primer*, you might like to take a bit of time out to see what the application does. But you don't need to. You'll catch on to the purpose of the application when you run it.

Looking at the panel, it's possible to work out what each file contains, from its filetype. You've got five OS/VS COBOL programs (ACCT00 through ACCT04), a set of BMS maps (ACCTSET), and two pairs of data files (ACCTFIL and ACCTIX).

There are three things worth noting about these application objects:

1. The program and map files are "raw", that is, we haven't translated and compiled or assembled them. We'll ask you to do that in just a minute but, before you move on to the next section, you might like to have a look at some of the files using your editor. Just move the cursor to the file you want, and press PF6.

   Note: "Standard" CICS/CMS, as supplied by IBM, uses the CMS editor, XEDIT. It's possible that your system administrator has changed this so that you can use your preferred CMS editor.

2. The data files are CICS/CMS versions of VSAM key sequential data set (KSDS) files. We'll have more to say about them in "Setting up your data files" on page 24.

3. Those of you who are familiar with the *Primer* application will notice that two copybook files, ACCTREC and ACIXREC, appear to be missing. Don't worry, you don't have to enter them yourself. They're kept in a special macro library file, EFHCSTD, and those sample application programs that need them will find them without your doing anything.

   We'll describe EFHCSTD, and other macro library files, in "CICS/CMS macro libraries" on page 99, but you don't need to know anything more about them to carry on to the next section.

Assembling a map

The first thing you need to do to test the sample application is to assemble the maps.

Move the cursor alongside the file containing the maps (ACCTSET), and press PF10. This runs an EXEC, EFHMAPCR, which assembles the maps and produces:

- Your physical map, on the file ACCTSET TEXT, on your A-disk.
- Your symbolic map description (DSECT), as a member of the special macro library file:

  EFHCUSER MACLIB A

  CICS/CMS creates this file the first time you assemble a map, as you'll see when you assemble ACCTSET. It then adds all future DSECTs to that macro library file.

  As we'll see in "CICS/CMS macro libraries" on page 99, CICS/CMS uses different macro libraries for the different programming languages it supports.

If you now move the cursor alongside file ACCTSET, and press PF10, you'll see the display shown in Figure 3.

```
EFH9175I Formatting 5 cylinders of temporary 3350 disk (BLKSIZE 1024) on 199 ...


EFH9182I Map ACCTSET being assembled for Application Data Structure....

ASSEMBLER (XF) DONE
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
EFH9183I DSECT assembly rc = 0
EFH9184I ACCTSET added to EFHCUSER MACLIB A. Return code =  0 .


EFH9186I Map ACCTSET being assembled to produce Object Deck....

ASSEMBLER (XF) DONE
NO STATEMENTS FLAGGED IN THIS ASSEMBLY
EFH9187I MAP assembly rc = 0.
EFH9193I ACCTSET TEXT A now created.




                                                              MORE...
```

Figure 3. Screen output from a successful BMS map assembly

You will usually see the first line of the display only when you first assemble, translate, or compile in any CICS/CMS session. It's the result of CICS/CMS setting up disk space on the CMS Z-disk to hold listings. If you're using a PC, and you've set up your Z-disk as described in "Organizing your PC storage for CICS/CMS," you won't see the message at all.

Note also that the exact form of the message will depend on the type of disks your installation uses. The example above came from using our installation setup; yours may well be different.

The messages beginning EFH tell you about the progress of the assembly. In this case, the assembly has worked, so all the messages end in "I"; they are for information only. Messages EFH9183I and EFH9184I confirm that the DSECT has been created successfully, and added to the macro library file for the COBOL maps. Because this is the first assembly you've done, CICS/CMS will create that file.

Messages EFH9187I and EFH9193I confirm the successful creation of the map.

If there were any problems with any part of the assembly, CICS/CMS would tell you in this display. You'll find more about this in "Assembling maps" on page 94.

Clear the screen, and CICS/CMS will redisplay panel EFH11, with the cursor still alongside the map file. If you press PF4, CICS/CMS will display all files having that name, as shown in Figure 4.

```
 EFH11              FILELIST:ACCTSET * *                    Line 1 of 4

Cmd   Filename Filetype Fm Format Lrecl     Records     Blocks   Date     Time
      ACCTSET  TEXT     A1 F         80          41          1  3/03/86  8:59:58
      ACCTSET  ASMLIST2 Z1 F        121        1521        180  3/03/86  8:59:57
      ACCTSET  ASMLIST1 Z1 F        121         601         72  3/03/86  8:59:27
      ACCTSET  ASSEMBLE B2 F         80         171          4  3/01/86 15:43:11

PF1=Help     2=Refresh 3=End       4=All this Name 5=Translate/Compile 6=Edit
PF7=Backward 8=Forward 9=Install  10=MAP-TEXT&ADS 11=Execute          12=

====>
```

Figure 4. File output from a successful BMS map assembly

The display is a version of panel EFH11, showing only those files with the name you selected with the cursor. In Figure 4 you can see four such files: the assembled TEXT file, containing the physical map, stored on your A-disk; the listing files from the two stages of the assembly, stored on your temporary Z-disk; and finally, the source BMS map file, stored on the CICS/CMS system disk.

You won't see your DSECT in this display; CICS/CMS has put it in the macro library file, which isn't called ACCTSET. If you want to check the contents of ACCTSET, move the cursor on EFH11 to the command line (the line at the bottom beginning ====>). Type:

```
FILELIST EFHCUSER * A
```

and press ENTER. You'll then get a CMS file list showing only your EFHCUSER macro library file. You can check your DSECT in that file, using your editor. However, all you can do is look at the DSECT with a CMS editor; you can't change it. We'll discuss how you change MACLIB type files in "Changing your user macro library" on page 99.

Translating and compiling a CICS program

Function key PF5 on panel EFH11 is set to `Translate/Compile`. To translate ***and*** compile or assemble a CICS source program, move the cursor alongside the filename of the file containing the program, and press PF5. This executes the CICS/CMS EXEC, EFHTC.

To try it out, you first need to get back to the `ACCT*` form of EFH11. Assuming you are currently looking at the macro library file EFHCUSER with your editor, you need to press PF3 once (to leave the editor), again (to return to the `ACCTSET` form of EFH11), then again (to return to the EFH11 you want).

Once you've reached the correct panel, move the cursor to ACCT00 COBOL, the first of the programs in the sample application, and press PF5. As soon as EFHTC begins the translation, it starts to write messages to your terminal, as shown in Figure 5.

```
EFH9166I Translating ACCT00 COBOL B2 ...
EFHCTRAN (NUM
EFH9167I No translator messages.


EFH9094I Invoking COBOL compiler to process translated O/P ....
COBOL TRANOUT (BATCH APOST LIB NOTRUNC


REL2.4 OS/VS COBOL IN PROGRESS
EFH9144I Compilation complete, return code= 0 . ACCT00 TEXT on A DISK.









                                                          MORE...
```

Figure 5. Screen output from a successful translate/compile

As you can see from the EFH messages, the translator has no problems to report, and the compilation return code is 0. This tells you that both translation and compilation have been successful.

Clear the screen, and you'll return to the EFH11 panel from which you started the translation/compilation. Press PF4, and you will get a list of all the files with the name ACCT00, as shown in Figure 6.

```
 EFH11              FILELIST:ACCT00 * *                    Line 1 of 4

Cmd   Filename Filetype Fm Format Lrecl     Records     Blocks   Date     Time
      ACCT00   TEXT     A1 F         80          46          1  3/03/86  9:01:56
      ACCT00   COBLIST  Z1 F        133         123         16  3/03/86  9:01:55
      ACCT00   TRANLIST Z1 F        121          40          5  3/03/86  9:01:49
      ACCT00   COBOL    B2 F         80          17          1  3/01/86 15:43:11









PF1=Help     2=Refresh 3=End       4=All this Name 5=Translate/Compile 6=Edit
PF7=Backward 8=Forward 9=Install  10=MAP-TEXT&ADS 11=Execute          12=

====>
```

Figure 6. File output from a successful translate/compile

There are three new files. The ones with the filetypes TRANLIST and COBLIST, on your Z-disk, contain your translator and compiler listings respectively. The one with the filetype TEXT, on your A-disk, contains the translated and compiled program, which is ready to run.

The important phrase is "ready to run". You've created a program TEXT file, and assembled the set of maps that it needs, and that's all you need to do to get this particular program ready for testing on CICS/CMS. You don't have to move to another system; you can run CICS here. And you don't have to do anything to your CICS system to get the program running. Since all the resources it uses are local, and since you're going to run it as a program, rather than as a transaction, you don't have to update any tables to define the resources it uses.

Testing a CICS program

So far, you've assembled the set of maps for the sample application, and translated and compiled the first of the programs that make up the application. In a later section, we'll ask you to get the rest ready and test the complete application. There's no reason, however, why you shouldn't just test the program you've prepared. If there's something wrong with the program that drives the rest of the application, it would be a good idea to fix it before doing anything else.

Ensure that you are looking at an EFH11 panel, with the cursor beside the file ACCT00 TEXT (your translated and compiled program). Press PF11 (`Execute`), and CICS/CMS will display its execution panel, EFH12, as shown in Figure 7 on page 19.

```
EFH12                          EXECUTE PROGRAM/TRANSACTION

 Type the name of the transaction or program and any optional parameters:

      Transaction   ===>
      Program       ===> ACCT00

 Status:

    EDF is OFF









PF1=Help         2=CEBR          3=End        4=CECI       5=          6=Start-Clear
PF7=Set-parms    8=VM-Session    9=EDF ON     10=          11=         12=Terminate
PA2=CMS Subset

CP/CMS Command ===>
```

Figure 7. CICS/CMS execution panel (EFH12)

We won't look at the full range of facilities available from this panel now; we'll get round to them later. For the moment, you just want to run the program you've compiled.

As you can see, in moving from panel EFH11 to panel EFH12, CICS/CMS has carried along the name of your TEXT file, and placed it in the `Program` field on panel EFH12. To run the program, you just press ENTER.

It is at this point that the CICS component of CICS/CMS is initialized: when you run a transaction or program from panel EFH12. Here, and throughout this guide, we refer to this part of CICS/CMS as the ***CICS environment***. We call the time you spend in that environment, that is, the time from starting a transaction or program from panel EFH12, to returning to panel EFH12, a ***CICS test session***. This helps to distinguish the testing part of application development from the ***CICS/CMS session***, which is everything that happens between entering the `CICSCMS` command, and returning to CMS.

As you'll see in "The CICS environment" on page 124, there are some differences between what you can do in a CICS/CMS session and what you can do in the CICS environment within that session.

As ACCT00 starts, CICS/CMS clears the execution panel (EFH12), and displays a screen showing the program name in the top left-hand corner. When this clears, you'll see the application's initial selection menu, as shown in Figure 8 on page 20.

```
ACCOUNT FILE: MENU

   TO SEARCH BY NAME, ENTER:                                    ONLY SURNAME
                                                                REQUIRED. EITHER
      SURNAME:                FIRST NAME:                       MAY BE PARTIAL.

   FOR INDIVIDUAL RECORDS, ENTER:
                                                                PRINTER REQUIRED
      REQUEST TYPE:     ACCOUNT:         PRINTER:              ONLY FOR INT
                                                                REQUESTS.
      REQUEST TYPES:  D = DISPLAY    A = ADD     X = DELETE
                      P = PRINT      M = MODIFY

   THEN PRESS  ENTER              -OR-   PRESS "CLEAR" TO EXIT
```

Figure 8. Sample application—initial selection menu

Give it something to do. Try entering D (for Display) for the REQUEST TYPE, and any five-figure number for the ACCOUNT.

Note: If you are running CICS/CMS as supplied by IBM, it will not convert anything you enter in the CICS environment from lowercase to uppercase. The sample application accepts requests in uppercase only, so that is how you must enter them. We tell you how to ask CICS/CMS to translate all your entries into uppercase in "Terminal control parameters" on page 244.

When you've entered your request on the sample application menu, press ENTER. You'll get a new screen, containing nothing but the message:

```
THE FOLLOWING NAMES ARE UNDEFINED:
 ACCT01
```

What has happened is this:

- The first program, ACCT00, has finished, setting a return transaction ID of AC01.
- The program table supplied with CICS/CMS defines ACCT01 as being the program for AC01, as shown in Figure 12 on page 23.
- You haven't translated or compiled ACCT01 yet, so, when CICS asks CMS for a file called ACCT01 TEXT, CMS can't find it, and displays the message above.

The CMS screen has MORE... in the bottom right-hand corner, so you need to press CLEAR to get the next display, which is a CICS/CMS error handler panel giving more information about the problem. This display is shown in Figure 9 on page 21.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 09:05:20

Program Name: Unknown                                Line Number: Unknown

Message EFH0450E has been generated

After this screen the CICS environment will CONTINUE

Module ACCT01   not successfully loaded.
Return code from CMS for INCLUDE command = 4
Perhaps ACCT01   TEXT does not exist or
   there are unresolved external references.

Refer to LOAD MAP A.



Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 9. Error handler report of a missing program

We won't look at the form of the error handler display here, or describe the facilities available from its PF keys. We deal with it in "The CICS/CMS error handler display" on page 161.

It's worth noting, however, how much help the error handler gives you. It suggests two possible reasons for the failure, one of which is:

```
ACCT01   TEXT * does not exist
```

If you now press PF12, CICS/CMS will take you back to the execution panel (EFH12).

Testing the complete application

Now you've seen how easy it is to test a CICS program using CICS/CMS, you can test the complete application. Before doing so, however, you need to translate and compile the four remaining programs (ACCT01 to ACCT04).

Press PF3 on panel EFH12. This will return you to the ACCT00 form of panel EFH11. Press PF3 again, and you will return to the ACCT* form of EFH11.

You then need to press PF5 against each application program name. You'll notice that CICS/CMS does not display a new list of files after each translation/compilation. If you want the file list to show the TEXT and listing files, you'll need to press PF2 (Refresh) after each operation.

Once you've translated and compiled all the source programs, you're ready to test the complete sample application. Before you do so, however, let's look at what you'd have to do to prepare for a test if this were your own application.

Updating tables

If you were developing your own application, it is at this point that you would need to update a table. You would need to add entries to the program table to associate programs with the transaction ID(s). However, we have supplied a predefined program table for the sample application, so, here, you don't have to do anything. It's worth looking at the table, however, to see how easy it is to use and to change.

If you've been translating and compiling the sample programs, as suggested above, you'll currently have some form of EFH11 panel on your screen. Press PF3 to return to panel EFH1. Now select option 3. CICS/CMS will display a new panel, as shown in Figure 10.

```
EFH13                CICS/CMS APPLICATION RESOURCE DEFINITION


 Select one of the following:

                  1   List Program/Transaction tables
                  2   List File Tables
                  3   List Transient Data Queue tables
                  4   List Remote Temporary Storage tables
                  5   List PSB Directories


           Selection ===> 1









PF1=Help          PF3=End          PA2=Enter CMS Subset
```

Figure 10. CICS/CMS application resource definition panel (EFH13)

Option 1 (the default) selects any program tables you have. Press ENTER, and you'll get a new display as shown in Figure 11 on page 23.

```
 EFH131               FILELIST:* EFHTPROG *                    Line 1 of 1

Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      CICSCMS  EFHTPROG B2 F         80         24          1  3/01/86 17:00:18









PF1=Help      2=Refresh  3=End       4=Sort(date)    5=Sort(size)      6=Edit
PF7=Backward  8=Forward  9=Install  10=Cursor       11=               12=

====>
```

Figure 11. CICS/CMS program table file list (EFH131)

CICS/CMS tables are CMS files that you have defined as tables. Your system administrator will set up default definitions, but you can set up your own, as explained in Chapter 4, "Setting up your CICS/CMS environment" on page 47. Here, we are using the IBM-supplied defaults for a program table: a file with the filename CICSCMS and the filetype EFHTPROG.

If you now press PF6, you can look at the CICS/CMS sample program table. It contains the table entries for the application programs, as shown in Figure 12.

```
* ************************************************************************
* EXAMPLE OF A CICS/CMS PROGRAM TABLE                                    *
* ************************************************************************
* For full details please refer to CICS/CMS User Guide.
* Comment lines commence asterisk space.
* Program name is the name of a TEXT file.
* Entry name is name of PROGRAM-ID(COBOL), external procedure (PL/I) or
*   CSECT (Assembler).  This field may be omitted if same as program
*   name.
* Language is ASSEMBLE, PLI, COBOL or COBOL2.
* Load method is INCLUDE (default) or NUCXLOAD.
* Data is column dependent as indicated by the following numbers.
*         11        21        31        41
* program  entry   language   load      transact
* name     name               method    or /PFnn
ACCT00    ACCT00    COBOL     INCLUDE   ACCT
ACCT01    ACCT01    COBOL     INCLUDE   AC01
ACCT02    ACCT02    COBOL     INCLUDE   AC02
ACCT03    ACCT03    COBOL     INCLUDE   AC03
* ACLG and AC05 are 3 alternative transaction-ids for ACCT03
                                       ACLG
                                       AC05
ACCT04    ACCT04    COBOL     INCLUDE   AC04
```

Figure 12. Sample CICS/CMS program table

Each program has a separate entry in the table. The fields in each entry must start in defined columns, as shown in Figure 12. They define, in turn:

- The program name (starting in column 1).
- The program's entry point (starting in column 11). Here, the entry point is the same as the program name.
- The language in which the program is written (starting in column 21). This is COBOL for the sample application programs.
- The way that the program is loaded (starting in column 31). This is INCLUDE for all the sample application programs.
- The transaction ID associated with the program (starting in column 41).

  Note how we have associated ACCT03 with its three transaction IDs, by listing those IDs in the `transact` column.

If you need to change a particular entry, or add a new one, you just use your editor.

Setting up your data files

If you were developing an application that needed data files, you'd need to make the files available before testing it.

We've made it easy for you here by providing the data files the sample application needs: ACCTFIL and ACCTIX. You may remember from Figure 2 on page 13 that these files in fact come as four files: two with the filetype EFHVDATA, and two with the filetype EFHVINDX. This is the way that CICS/CMS simulates key sequential data set (KSDS) VSAM files. It lets you use the EXEC CICS commands associated with CICS/VS file control on CMS files, without making any changes to those commands.

How do you create such files? You'll be pleased to hear that you don't have to create them yourself, from scratch, using your editor. CICS/CMS provides two file utilities: one converts CMS files to CICS/CMS ***pseudo-VSAM*** files; the other, among other things, copies VSAM files from a remote CICS/VS system, converts them into pseudo-VSAM files, and saves the converted files on one of your CMS disks.

If this were your own application, you'd almost certainly have to use one of the utilities at this point to prepare your data. Since we've done it for you, however, we needn't say anything more about data files at the moment. We describe the form of pseudo-VSAM files in detail in "How CICS/CMS supports VSAM files" on page 60, and the conversion utilities in "Converting local CMS files to CICS/CMS keyed files" on page 110 and "General file conversion" on page 113.

There is one more thing that you might think you have to do at this stage; update the file control table (FCT). In fact, as we'll explain fully in "File tables," you only have to define files in a table if:

- They are new files that you're going to create
- They are files that you want to access on a remote CICS/VS system, without using the `SYSID` option on your EXEC CICS commands
- You want to write information to an existing, variable-length file, and the information is longer than the file's current maximum record length.

Because the files that this application uses are existing, fixed-length, local files, you don't have to define them in a table. There is one piece of file preparation you will have to do, just this once, before you test the complete sample application. At the moment, the sample data files are on your CICS/CMS system disk. While they are there, you can only read them; you can't write anything to them. Since we'll be asking you to add information to the files in some of the sections that follow, you'll need to copy them (using CMS's COPYFILE command) from the system disk to your A-disk before going on to the next section.

To do this, you need to go through the following steps:

1. Return to the ACCT* form of EFH11 panel. Assuming you are still looking at the sample program table, you need to press PF3 once (to return to panel EFH131), again (to go to panel EFH13), then again (to go to panel EFH1). From there, you can press ENTER to display panel EFH11.

2. On panel EFH11, you need to copy four files: the two EFHVDATA type files, and the two EFHVINDX type files, with the names ACCTFIL and ACCTIX. Move the cursor down the column headed Cmd, until it is alongside the first of these files. Type the CMS command:

   ```
   COPYFILE / = = A
   ```

   This means "make a copy of the file indicated by the cursor, giving it the same filename and filetype, but a filemode of A". Don't worry about the command overwriting the rest of the information in the line; it doesn't matter. Before you press ENTER, use your "carriage return" key to move the cursor alongside each of the other three files, and type an "equals" (=) beside each one. This tells CMS to repeat the COPYFILE command for each file. Press ENTER, and, when CMS indicates that the command is complete by displaying an asterisk (*) beside each filename, press PF2. This will refresh the display, and you'll see that you have copies of the four data files on your A-disk.

Testing a CICS transaction

Because the CICS/CMS program table for the sample application is predefined, and the data files prepared, you can test it as soon as you've assembled all the maps and translated/compiled the programs.

To test the application, you need to display the execution panel (EFH12). Assuming you have just copied the data files, as described at the end of the previous section, you need to press PF3 to return to panel EFH1. On EFH1, select option 2 to display panel EFH12. On the line that says:

```
Transaction     ===>
```

type the transaction ID for the application, ACCT. Press ENTER, and the application will run.

If you're not familiar with the *CICS/VS Application Programming Primer*, you will probably never have seen the sample application in action before. It manages customer accounts for an imaginary store. Through a series of menus, its users can display, add, remove, and modify customer records. You might like to try out some of these functions before moving on to the next section.

Notes:

1. The sample data file we supply only contains a single record: account number 11111. There's nothing to stop you adding a few yourself, however.

2. When you've finished trying out the sample application, you'll need to get back to panel EFH12 before going on to the next section. There are three ways of doing this:

   a. If you press CLEAR or ENTER on one of the sample application's panels, you'll get a blank screen. To get back to EFH12, you enter:

      ```
      CCMS QUIT
      ```

      This is a CICS/CMS command that lets you go back to the execution panel at any time that CICS is waiting for a transaction ID.

   b. If your screen is displaying a CICS/CMS error handler panel (EFH125), you can press PF12 to get back to panel EFH12.

   c. If you press PA2 on any panel where CICS is waiting for you to enter something, you will display the CICS/CMS escape panel, EFH122. Pressing PF12 on that panel will return you to panel EFH12.

Testing tools

To help you test your applications, we've provided some of the interactive CICS/VS tools with CICS/CMS, and provided a feature, unique to CICS/CMS, that lets you take full advantage of having both CICS and CMS available at the same time. This section introduces some of the testing tools.

The execution diagnostic facility

The execution diagnostic facility (EDF) lets you monitor the execution of the EXEC CICS commands in your transactions, by running them under the control of another transaction, CEDF. As a transaction runs, CEDF interrupts its execution at defined points, and displays one of its panels on the terminal. The points at which you get this display include:

- The start of the transaction
- Before and after execution of each EXEC CICS command.

To see how useful EDF can be when you use it with CICS/CMS, try running the sample application with EDF turned on, and simulate an error to see the results.

Press PF9 on panel EFH12. You'll see that, in the panel's `Status` area, EDF changes from `OFF` to `ON`. At the same time, the definition of PF9 changes from `EDF ON` to `EDF OFF`. You can tell from this that PF9 is a switch. Whatever the current status of EDF is when you press PF9, it switches to the opposite.

Now run the sample application by entering `ACCT` in the `Transaction` field, and pressing ENTER. Continue to press ENTER to step through the EDF displays until the initial transaction ends. You can then continue to the next transaction, under EDF's control, by changing `NO` to `YES` in the bottom right-hand corner of

the final EDF display for the first transaction, and pressing ENTER. The AC01 transaction will then start.

Continue to press ENTER until you see the application menu. Enter D (for display) for the REQUEST TYPE, and some random 5-figure number (say, 79999) for the ACCOUNT. Carry on pressing ENTER until you get the EDF panel indicating that it is about to execute an EXEC CICS READ on the application file, as shown in Figure 13.

```
TRANSACTION: AC01   PROGRAM: ACCT01     TASK NUMBER: 0000008   DISPLAY:  00
STATUS:  ABOUT TO EXECUTE COMMAND
EXEC CICS READ
 DATASET ('ACCTFIL ')
 INTO ('.....................................................'...)
 LENGTH (383)
 RIDFLD ('79999           ')




OFFSET:X'002064'    LINE:  237          EIBFN=X'0602'
RESPONSE:
                                                              REPLY:
ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : UNDEFINED
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: UNDEFINED           PF12: ABEND USER
```

Figure 13. EDF display before executing a READ command

Before you let the program continue, you've got a perfect opportunity to see what happens for a particular (and obvious) error: a missing file. You can see on the EDF display that the READ is to the file ACCTFIL. What would happen if that file didn't exist? One way of finding out is to change the name of the file on the EDF display to some unlikely name (try UNLIKELY). All you have to do is move the cursor to the file (DATASET) name and overtype it.

Press ENTER twice and you will get a CICS/CMS error handler panel, warning you that the application is in trouble, as shown in Figure 14.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 09:10:03

Program Name: ACCT01                                 Line Number:    241

Message EFH0946E has been generated

After this screen the CICS environment will CONTINUE

A request has been received for file UNLIKELY which does not exist
and which has not been defined as a new file.
You must insert an entry in your file resource table for this file.
The request is rejected with DSIDERR.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 14. Error handler message for a nonexistent file

The message clearly states what is wrong, and what the results will be. If you press ENTER to resume the application, you will see how accurate its analysis is.

The first thing you will see is the EDF panel reporting the results of executing the READ, as shown in Figure 15.

```
 TRANSACTION: AC01   PROGRAM: ACCT01     TASK NUMBER: 0000008   DISPLAY:  00
 STATUS:  COMMAND EXECUTION COMPLETE
 EXEC CICS READ
  DATASET ('UNLIKELY')
  INTO ('.....................................................'...)
  LENGTH (383)
  RIDFLD ('79999           ')




 OFFSET:X'002064'    LINE:  237          EIBFN=X'0602'
 RESPONSE: DSIDERR                       EIBRESP=12
                                                          REPLY:
ENTER:  CONTINUE
PF1 : UNDEFINED           PF2 : SWITCH HEX/CHAR     PF3 : END EDF SESSION
PF4 : SUPPRESS DISPLAYS   PF5 : WORKING STORAGE     PF6 : USER DISPLAY
PF7 : SCROLL BACK         PF8 : SCROLL FORWARD      PF9 : STOP CONDITIONS
PF10: PREVIOUS DISPLAY    PF11: UNDEFINED           PF12: ABEND USER
```

Figure 15. EDF display after executing a READ command

As the EFH125 panel warned, the command has failed with a DSIDERR error condition. If you continue to press ENTER, the application will link to ACCT04, its error message program, and you will eventually see an error panel as shown in Figure 16 on page 29.

```
ACCOUNT FILE: ERROR REPORT

TRANSACTION  AC01  HAS FAILED IN PROGRAM  ACCT01 BECAUSE OF

A PROGRAM OR FCT TABLE ERROR (INVALID FILE NAME).

THE FILE IS:

PLEASE ASK YOUR SUPERVISOR TO CONVEY THIS INFORMATION TO THE
OPERATIONS STAFF.

THEN PRESS "CLEAR".  THIS TERMINAL IS NO LONGER UNDER CONTROL OF
THE "ACCT" APPLICATION.
```

Figure 16. Sample application error report

If you continue to press ENTER, the application will eventually abend, and you will see the error handler panel shown in Figure 17.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 09:11:27

Program Name: ACCT04                                 Line Number:   139

Message EFH8698S has been generated

After this screen the CICS environment will TERMINATE

ABEND EACC - User abend. Please refer to application program.
Request to abend task from DFHEIP.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 17. Error handler display for a user abend

From here, you can press PF12 or ENTER to return to panel EFH12. As the message says, the problem is now so severe that your CICS test environment will have to terminate, returning you to panel EFH12, where you can sort out the problem before starting a new test.

Using EDF, you have demonstrated that the execution path you defined to handle data set errors is working properly. If you were testing one of your own applications, you could use this technique to try out every possible execution path in your program. You could use EDF to simulate all the most likely error

conditions to ensure that any HANDLE CONDITION commands in your program did what you wanted them to do.

You can also use EDF the other way round, that is, to simulate correct execution when there are, in fact, errors. When you're testing a single program that forms part of a larger application, you will probably get errors because it tries to link to a program that you haven't yet written, or read a data file that you haven't yet created. If you run the program with EDF, you can change any error conditions, on the EDF panel, to NORMAL, testing the logic of the program in its working environment.

The command level interpreter

The command level interpreter is a transaction (CECI). It lets you execute individual EXEC CICS commands. You can use it for online help, to check the syntax of a command, or you can use it to change or inspect application objects without running a complete transaction. We'll show you an example of this in the next section.

You can use CECI from the execution panel (EFH12), by pressing PF4. If you do that now, you will see the panel shown in Figure 18.

```
 STATUS:  ENTER ONE OF THE FOLLOWING

ABend        ENDbr        POSt         SPOOLClose
ADdress      ENQ          PURge        SPOOLOpen
ALlocate     ENTer        PUSh         SPOOLRead
ASKtime      EXtract      READ         SPOOLWrite
ASSign       FOrmattime   READNext     START
BIf          FREE         READPrev     STARTBr
BUild        FREEMain     READQ        SUspend
CAncel       Getmain      RECeive      SYncpoint
CONNect      Handle       RELease      Trace
CONVerse     IGnore       RESEtbr      Unlock
DELAy        INquire      RESYnc       WAit
DELETE       ISsue        RETRieve     WRITE
DELETEQ      Journal      RETUrn       WRITEQ
DEQ          LInk         REWrite      Xctl
DIsable      LOad         ROute
DUmp         POInt        SENd
ENAble       POP          SET


PF 1 HELP 2 HEX 3 END 4 EIB 5 VAR 6 USER             9
```

Figure 18. Initial display from CECI

CECI lists all the EXEC CICS commands it knows, and invites you to enter one. You can enter a complete command, in which case, CECI will check the syntax and report any errors. Alternatively, you can enter just the statement keyword, in which case CECI will produce a further display, showing you the possible parameters. Note that, because CICS/CMS doesn't support the whole of the CICS/VS command level API, some of the commands that CECI lists won't work in the way you might expect. For a complete account of CICS/CMS support of the API, see the *CICS/CMS Application Programmer's Reference Summary*.

When you've finished with CECI, you press PF3. You'll then see a screen containing the lines:

```
CECI command
 STATUS:  SESSION ENDED
```

where command is the last EXEC CICS command you entered. You can then backspace the cursor over CECI, and overtype either another transaction name (as shown in "Browsing temporary storage" below), or use the CICS/CMS command:

```
CCMS QUIT
```

to get back to the execution panel.

Browsing temporary storage

Another transaction supplied with CICS/CMS is CEBR, which lets you look at the contents of temporary storage queues. You can run CEBR from the execution panel (EFH12) by pressing PF2. Showing you CEBR in action gives us a good opportunity to show you how versatile CICS/CMS can be, by combining the capabilities of CECI and CEBR.

As we explained in the last section, you can use CECI to execute any EXEC CICS command, including the WRITEQ command to write information to a temporary storage queue. If you now start CECI (press PF4) and execute the command:

```
WRITEQ TS QUEUE('DUMMY') FROM('THIS IS A TRYOUT')
```

CICS/CMS will create a temporary storage queue (DUMMY) and write to it. You can then check that queue with CEBR. Press PF3 to stop CECI, and type:

```
CEBR DUMMY
```

over the CECI command line. You will then see the panel shown in Figure 19.

```
CEBR            TS QUEUE  DUMMY    RECORD     1 OF     1    COL     1 OF    16
ENTER COMMAND ===>
     ************************** TOP OF QUEUE  ******************************
00001 THIS IS A TRYOUT
     ************************** BOTTOM OF QUEUE  ****************************









PF1 : HELP                PF2 : SWITCH HEX/CHAR     PF3 : TERMINATE BROWSE
PF4 : VIEW TOP            PF5 : VIEW BOTTOM         PF6 : REPEAT LAST FIND
PF7 : SCROLL BACK HALF    PF8 : SCROLL FORWARD HALF PF9 : UNDEFINED
PF10: SCROLL BACK FULL    PF11: SCROLL FORWARD FULL PF12: UNDEFINED
```

Figure 19. Sample CEBR display

When you've finished with CEBR, press PF3, and you'll see a screen containing the message:

```
BROWSE HAS ENDED NORMALLY.  READY FOR NEXT TRANSACTION:
```

You can enter another transaction ID over this line, or use `CCMS QUIT` to get back to panel EFH12.

CICS/CMS escape functions

Suppose you want to check that the sample application is writing records correctly. The CICS/CMS escape panel lets you do this, and many other things, ***while an application is running***.

You can get CICS/CMS to display its escape panel (EFH122) at any time that CICS is waiting for you to enter something on your terminal. For example, you can escape from any CECI, CEBR, or EDF display, or from a display put up by your own application. To escape, you press PA2.

Let's see this in action.

First, make sure that EDF is off. If the status line on panel EFH12 says that it's on, press PF9. Then run the sample application by typing `ACCT` and pressing ENTER. When you see the application's first menu, select A (add) and enter a nonexistent account number. Fill in enough details on the Add panel that follows to get a new account number added to the file. When the application has accepted your addition, and returned to its first menu, press PA2. You will see the display shown in Figure 20.

```
EFH122                          ESCAPE FUNCTIONS

  Type the name of the transaction or program and any optional parameters:

      Transaction    ===>
      Program        ===>


  Status:

     EDF is OFF

     Nest level




PF1=Help       2=CEBR        3=End      4=CECI        5=Appln-PA1   6=Appln-PA2
PF7=Set-parms  8=VM-session  9=EDF ON  10=Reset-msgs 11=          12=Terminate
PA2=CMS-subset

CP/CMS Command ===>
```

Figure 20. CICS/CMS escape panel (EFH122)

The escape panel looks very much like the execution panel, and offers much the same services, including:

- Executing a program or transaction
- Starting EDF, CECI, or CEBR
- Executing a single CP or CMS command.

The main differences between panel EFH122 and panel EFH12 are:

- If you enter a command on the `CP/CMS Command` line on panel EFH122, it is executed within the CMS subset. This limits the commands you can enter; you can't, for example, use the CMS `COPYFILE` command. You'll find some examples of what you can do in the subset in "Using the CICS/CMS escape feature" on page 127.

- You can't run the CICS/CMS EXECs EFHMAPCR and EFHTC from panel EFH122.

- Because escaping suspends any transactions that are executing when you escape, anything you then do from panel EFH122 is nested within that transaction. CICS/CMS doesn't limit the number of times you can escape, so it helps you remember where you are by showing the `Nest level` in panel EFH122. It gives you a symbolic representation of the number of tasks and programs actually "in flight" when you escape. We'll explain this in more detail, and show you how to interpret the nest level, in "Using the CICS/CMS escape feature" on page 127.

  Note that, in the example we're looking at here, the nest level is blank. The application has finished adding a record, and is now waiting for your next request. It is between transactions in its pseudoconversation. There are therefore no transactions or programs in flight.

The escape panel feature we want to use here is the `CP/CMS command` line. We want to list the application files, and then browse the data file to make sure that the application is adding records correctly.

Move the cursor to the line that reads:

```
CP/CMS Command ===>
```

Type `FILELIST`, press ENTER, and you will get a list of all your files. Move the cursor alongside the file ACCTFIL EFHVDATA, and press PF6 to look at the file with your editor. You should be able to see, at the bottom of the file, the record that you just added.

To return to your application, and continue its execution where you left off, press PF3 once (to leave the editor), then again (to leave the file list). This will bring you back to the escape panel. Pressing PF3 again will return you to your application.

If you want to stop the application, and leave the CICS environment, press PF3 twice to get back to panel EFH122, then press PF12 (`Terminate`). This returns you to panel EFH12.

What's next?

The next part of the book describes aspects of using CICS/CMS that you need to know before you start using it to develop your applications.

The information comes in three varieties:

- Things you might ***need to do***: installing CICS/CMS on a PC.
- Things you might ***like to do***: tailoring CICS/CMS to suit your own requirements.
- Things you might ***need to know***: the special ways in which CICS/CMS handles some CICS/VS resources, and the ***tables*** it uses to do so.

Part two—Before you start developing applications

This Part of the book tells you:

- How to install CICS/CMS on a PC
- How to tailor CICS/CMS to your requirements
- How CICS/CMS handles CICS resources.

Chapter 3. Installing CICS/CMS on a PC

The word "installation" has two meanings when you're talking about CICS/CMS.

Before any application programmers can use CICS/CMS, the system administrator has to install it on the host VM system. Because this chapter is for application programmers, we don't discuss this form of installation here. If you're a system administrator, trying to find out how to install CICS/CMS on your host system, please read "Installing CICS/CMS on a VM system" on page 204.

The second meaning of "installation" for CICS/CMS is the process that PC users go through to copy CICS/CMS from the host system to their PCs.

If you are a PC user, you will need to read this chapter to find out how to:

- Set up your PC's fixed disk in preparation for installing CICS/CMS
- Copy the master version of CICS/CMS from the host VM system to your fixed disk
- Change your PC copy of CICS/CMS.

Take note

In " Getting ready to use CICS/CMS" on page 10, we suggested that you use your PC to get to know CICS/CMS by emulating a 3270 terminal and using the master version of CICS/CMS on your host VM system. If you have done so, you will have experienced the performance of CICS/CMS on a large, powerful machine. When you install CICS/CMS on your PC, and start to use it locally, you may at first find the performance disappointing in comparison, particularly for compilation. You wouldn't normally expect a PC to be able to rival the performance of a large mainframe.

Organizing your PC storage for CICS/CMS

Before you install CICS/CMS on your PC, you will need to change your VM/PC configuration file to set up the CMS minidisks you'll need to use CICS/CMS. This section tells you how to go about it, using the VM/PC configurator, VMPCCON. We've tried to give enough information for you to use VMPCCON without referring to any other documents. However, if this is the first time you've changed your configuration file, you'll probably need to refer to the *VM/PC User's Guide* for more information. We present the information as a series of steps, and advise that you follow them in the order given.

Notes

1. A full CICS/CMS system, including the compiler(s) you need, needs more than the 10 megabytes of storage available on an XT/370. You can therefore use CICS/CMS on an XT/370 only if you have an expansion unit (a D drive), that brings your PC up to the 20 megabytes available on an AT/370.

   In the descriptions of the steps below, when we refer to C and D drives, we are giving recommendations to the XT/370 user. On an AT/370, all storage for CICS/CMS will usually be on the C drive.

2. Under VM/PC 2.0, you have a PC-DOS session available on the VM/PC Session Selection menu. You must not use the VM/PC configurator in that session.

3. In the configuration steps, we refer frequently to two different types of disk: the physical disk, attached to your PC, and the CMS virtual minidisks that you're defining. To avoid confusion, we always refer to the PC disk as the ***fixed disk***, and to the CMS virtual disks as ***CMS disks***, or just ***disks***.

The steps in organizing PC storage

1. Enter the `VMPCCON` command from DOS. Enter your password and press ENTER.

2. On the Function menu, press PF3. This will display the User Selection menu. Tab to the user ID you want to use for CICS/CMS, and press ENTER. You will get a panel for that user ID. Press PF1 to display the User Environment panel. The `Virtual Machine Storage Size` line lets you define the amount of storage you'll have simulated in your virtual machine when you use VM/PC. We recommend that you set this to 2048K (2 megabytes) for CICS/CMS.

   The `Environment to Auto IPL` line below gives the name of the program to be loaded into your virtual machine when you log on to VM/PC. This should be CMS to ensure that you IPL CMS whenever you log on.

3. Press PF5 to return to the User Selection menu. Then press PF2 to get a list of your CMS disks.

   In the remaining steps, you set up the CMS disks you need to use CICS/CMS. Whenever we recommend a size for a CMS disk, we always state that size in the units used by the PC: 512-byte blocks.

4. Disk 100 is your VM/PC system disk. You shouldn't change it.

5. Disk 101 is your CMS A-disk. You need to make this as large as possible, preferably 2000 blocks. To do this, you will need to change its drive ID to C, since, by default, VM/PC puts it on your A drive (your floppy disk), which allows a maximum of only 720 blocks.

6. You now need to create some new CMS disks. Press PF5 on the disk list panel.

   a. **Creating a temporary disk.**

      First, you need to create a temporary disk, the Z-disk. In fact, CICS/CMS will format the Z-disk for you, but we recommend that you do it yourself, as explained here, for two reasons:

1) If you don't create a Z-disk at this stage, giving it the space it needs, you may run out of storage space later.

2) Unless you already have a Z-disk, CICS/CMS formats a new one, when needed, in each CICS/CMS session. This can take some time.

Press ENTER, to create a new CMS disk. You need to define:

- The disk address, which must be 199, because that's the address CICS/CMS assumes for the Z-disk.
- The drive ID. We recommend you use the C drive.
- The size of the disk. This depends on the size of your programs. We've found 2000 blocks adequate for most purposes.
- The access mode of the disk. This must be W (write).

b. **Creating the CICS/CMS system disk.**

Now you need to set up the CMS disk onto which you'll load CICS/CMS. You need to define:

- The address. This can be what you like (we usually use 200).
- The drive ID. This is going to be a very large disk, so you should put it on the D drive on an XT/370.
- The disk size. This must be at least 5600 blocks.

  The reason we've included the phrase "at least" is to take account of any changes that your system administrator might have made. 5000 blocks is the size we recommend for CICS/CMS as supplied by IBM. Your system administrator, however, may have changed CICS/CMS to suit your local requirements: for example, by adding some local macro libraries.

  Find out exactly how much space your local copy of CICS/CMS needs. Any space you can spare can be used to increase the size of your A-disk or your Z-disk.

- The disk access mode.

  Initially, this must be W (write), so that you can copy CICS/CMS from your host VM system to your PC. Once you've installed CICS/CMS, however, you should make this R (read only), to safeguard your CICS/CMS system. If your system administrator changes the host VM system copy of CICS/CMS at any time, you'll need to copy the changes down to your PC. At that time, you'll need to change the mode, temporarily, to W (write).

c. **Creating a compiler disk.**

Finally, if you're going to compile your programs on your PC, you will need to set up a CMS disk for the compiler(s) you're going to use and their associated libraries. If you're going to compile your programs on the host VM/SP system, and run them using CICS/CMS on your PC, you need a CMS disk for the compiler libraries. For each one you need to define:

- The disk address.

  Note: Although you can use any disk address you like for a compiler under VM/PC, you must access the disk using the same mode as the CMS disk containing the compiler on your host VM system. For example, if you want to use the COBOL compiler, and the host VM system has it on the P-disk, you must access the compiler disk on your VM/PC system as the CMS P-disk.

- The drive. If the compiler or compilers you are going to use take up a lot of space (see below), this will need to be your D drive on the XT/370.

- The disk size. This depends on the compiler you're using. The minimum for the PL/I compiler and its libraries is 4100 blocks; the minimum for COBOL is 2000 blocks; the minimum for COBOL II is 4750 blocks; the minimum for the H Assembler is 400 blocks.

- The disk access mode. This should normally be R (read) to safeguard your compilers.

Figure 21 overleaf shows a typical PC fixed disk configuration including CICS/CMS, for both an AT/370 and an XT/370.

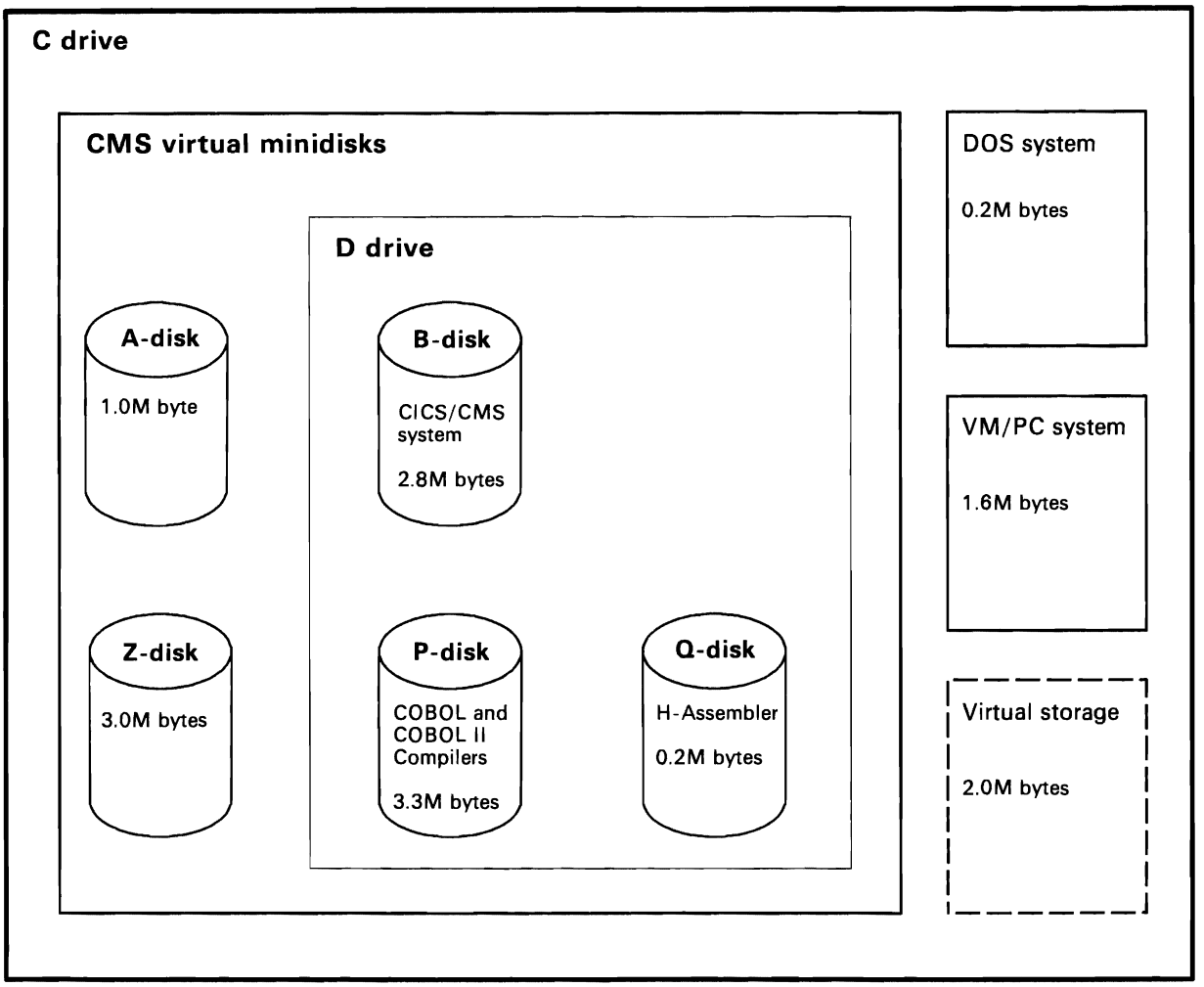


Figure 21. A typical PC storage configuration for CICS/CMS

Notes

1. The total storage defined in the example is 14.1 megabytes, which will all be on the C drive on an AT/370. On an XT/370, 7.8 megabytes is defined on the C drive, and 6.3 megabytes on the D drive.

2. The virtual machine storage size defined is shown in the example as 2.0 megabytes, which is what we recommend as adequate for most CICS/CMS systems. You can, however, use VMPCCON to define as much virtual storage as you have left on the C drive after defining everything else, up to the maximum that VM/PC allows (8 megabytes on an AT/370; 4 megabytes on an XT/370). Whatever you specify must be a multiple of 4K bytes.

As you can see, you'll need to define a lot of storage before you start using CICS/CMS to develop applications, particularly if you're using more than one compiler. You might therefore like to consider tailoring DOS to give yourself some extra space. For example, standard DOS contains several modules for National Language Support (NLS). If you need nothing but your own language, you can save quite a bit of storage by removing the other language modules. Don't, however, start tailoring DOS unless you are very sure which modules you can safely remove. If you have any doubts at all, talk to your local PC expert.

Before you install CICS/CMS on your PC, you need to format the CMS disks you've just created.

Select your local session, and start CMS. If you've just created a new A-disk, you'll get a message, before CMS starts, indicating that the A-disk has an `invalid directory format`. This is just an indication that you need to format the disk, and doesn't stop CMS from starting. You can then use CMS's `FORMAT` command to format each of your new disks. The general form of the command is:

```
FORMAT diskno mode
```

where `diskno` is the address of the disk you want to format, and `mode` is the mode of the disk. For example, to format your Z-disk, you'd enter:

```
FORMAT 199 Z
```

Having configured and formatted your CMS disks, you should add them to your PROFILE EXEC so that they'll be accessed automatically when you IPL CMS.

Copying CICS/CMS to a PC

CICS/CMS is stored on a CMS disk on the host system. The ***first*** time you want to use it locally at a PC, therefore, you have to copy it from that system to the CMS disk you've reserved for it on your PC's fixed disk. Note that we have stressed the word "first". Once you've got CICS/CMS stored on your fixed disk, you can invoke it by starting VM/PC and CMS, and following the process described in "Starting CICS/CMS" on page 11. The only time that you have to go through the process described here is the first time you ever use the product on a PC, or at any time that you need to restore the version on your fixed disk.

Take Note

This process can take a long time. Exactly how long depends mainly on whether the PC is attached to the host machine via a channel, (a ***local*** connection), or via a teleprocessing line (a ***remote*** connection). If you're using a remote connection, the speed at which you install CICS/CMS is further affected by the speed of the communication line.

On a local connection (the fastest), installation should take about 45 minutes. However, if your connection is remote, over a low-speed line, it can take over 2 hours. Before setting out to copy CICS/CMS to your PC, therefore, it's a good idea to find out how your PC is connected and make sure you've got something else to do while you're waiting.

Switch on your PC and, after it has loaded DOS, enter the date and time. You then enter:

```
VMPC
```

to start the VM that runs on PCs. Press ENTER, and you will get a menu on your screen. The exact form of this menu is different for different releases of VM/PC. The example we show in Figure 22 on page 43 is produced by VM/PC 1.1. The options in which you are interested here are the host and local 3278/3277 sessions. On VM/PC 2.0, they are in the reverse order; the local session is option 1, and the host session is option 2.

```
                          VM/PC Session Selection Menu

  Press the number of the session that you want or
  press Ctrl-Break for immediate exit to DOS.


    1 Host 3278/77 Session

    2 Local 3278/77 Session

    3 Host 3101 Session









LA                                                    M Session Selection
```

Figure 22. VM/PC session selection menu

Select the option that gives you a host session (option 1 in Figure 22). Log on to your host system, and type:

```
VMPCSERV
```

to establish the link between host VM and your PC. You now need to get back to your PC (your local session). Press the "hot" key (Sys Req on the AT/370, Scroll Lock on the XT/370). This returns you to the Session Selection menu, and you select the option that gives you a local session (option 2 in Figure 22).

Now start CMS on your PC. You can then link to the host system disk that contains the elements that make up CICS/CMS. Your system administrator will tell you what `LINK` and `ACCESS` commands you need to do this. Having executed them, you can type:

```
CICSCMS
```

You are now using the host system copy of CICS/CMS, using VMPCSERV as the link. This works, but is relatively slow. The sooner you copy CICS/CMS to your fixed disk, the sooner you can start using it the way it was meant to be used.

CICS/CMS will display the menu shown in Figure 23 on your screen.

```
EFH1                CICS/CMS PROGRAM DEVELOPMENT SELECTION

 Select one of the following:

                  1   List Application Files
                  2   Execute Program/Transaction
                  3   List Resource Tables
                  4   Convert CMS file to CICS/CMS file
                  5   Erase Temporary Files
                  6   Release CICS/CMS Nucleus Extensions
                  7   Download from Host

          Selection ===> 1

 For application filelist specify the optional list criteria required:

          Criteria  ===> ACCT* * *


             (c) Copyright IBM Corp 1985

PF1=Help          PF3=End          PA2=Enter CMS Subset
```

Figure 23. CICS/CMS program development selection panel (EFH1) for PC users

Select 7 (`Download from Host`), and press ENTER. You will then get a new panel, as shown in Figure 24.

```
EFH15               DOWNLOAD FROM HOST

 Select one of the following:

                  1   Download Run Time System
                  2   Download Sample programs, tables and maps
                  3   Download COBOL/COBOL2 Program Preparation Files
                  4   Download ASSEMBLE Program Preparation Files
                  5   Download PLI Program Preparation Files
                  6   Download BMS Map Preparation Files
                  7   Download All CICS/CMS Files
                  8   Build CICS/CMS Relocatable Load Modules


          Selection       ===>

          Input Filemode  ===>

          Output Filemode ===> A       Default - A


PF1=Help          PF3=End          PA2=CMS Subset
```

Figure 24. Panel for copying CICS/CMS from the host to a PC/370 (EFH15)

To copy CICS/CMS from the host system to your fixed disk, you need to do three things:

1. Give the output filemode: the letter that identifies the CMS disk on your PC that you have set up for your local CICS/CMS system.

2. Give the input filemode: the letter that identifies the host system CMS disk on which your system administrator has set up the master CICS/CMS system.

3. Select the option(s) you need to copy the CICS/CMS system to your fixed disk.

The first six options in panel EFH15 give you the opportunity to copy different components of CICS/CMS, depending on what you need. Option 7 copies the complete host CICS/CMS system. There are two advantages in copying only those elements of CICS/CMS that you need, as follows:

1. CICS/CMS will take up less space on your fixed disk, leaving you more space to give to, say, your A-disk.

2. You'll speed up the process of copying CICS/CMS from the host system.

What you get from each option is:

Option 1 The parts of CICS/CMS needed just to execute transactions and programs. If you don't use option 7, you must always use at least this option.

Option 2 The files and tables needed by the CICS/CMS sample application.

Option 3 The parts of CICS/CMS that you need to develop COBOL or COBOL II programs.

Option 4 The parts of CICS/CMS that you need to develop assembler programs.

Option 5 The parts of CICS/CMS that you need to develop PL/I programs.

Option 6 The parts of CICS/CMS that you need to develop BMS maps.

After selecting each option you need, and pressing ENTER, you will have to wait while the parts of CICS/CMS you've selected are copied from the host to your fixed disk. Remember from our note earlier in the section: if you select option 7, the process will take at least 45 minutes, and may take much more, depending on how your PC is connected to the host system. For any of the "partial" options, 1 to 6, it will be much quicker.

When it finishes copying each part of CICS/CMS you have selected, CICS/CMS will display the message:

```
EFH9110I Download completed successfully
```

When you've copied all the parts you need, you'll have to stop the link (VMPCSERV) between your PC and the host, and restart CICS/CMS. If you don't, all your CICS/CMS requests will continue to go to the host system. Use the hot key to return to the VM/PC session selection menu, and select the option that gives you a host session. Stop VMPCSERV by pressing PF3. Use the hot key once again, to return to the VM/PC session selection menu, and select the option that gives you a local session. Stop CICS/CMS, by pressing PF3 once (to take you back to panel EFH1), then again (to stop the CICS/CMS session and return you to CMS).

You now have CICS/CMS stored on your fixed disk. If this is the first time you've used CICS/CMS, and you have an OS/VS COBOL compiler, you'll probably want to go through Chapter 2, "Getting to know CICS/CMS." Remember that you'll only be able to do the things that Chapter 2 describes if you've copied the sample files. You can copy them either explicitly (using option 2 on panel EFH15), or implicitly (using option 7).

Note: You're probably wondering what option 8 on panel EFH15 does. It's a special option, which helps the system administrator ensure that changes to the principal CICS/CMS modules can be passed on to PC users. Since it's not needed by application programmers, we don't describe its use until "Rebuilding your CICS/CMS system" on page 216.

Changing CICS/CMS on a PC

As we said earlier, once you've got CICS/CMS on your fixed disk, you should never need to install it again, unless some accident damages or erases that disk. There is one other occasion, however, when you might have to install part or (very rarely) all of CICS/CMS.

From time to time, your system administrator will change the master copy of CICS/CMS to solve problems or include improvements. Anyone using VM/SP will automatically pick up the new copy of CICS/CMS from the system disk. PC users, however, will have to apply the changes to their fixed disk copy.

When your system administrator has put up a new copy of CICS/CMS on the system disk, he or she will alert you that you need to copy those parts of CICS/CMS that have changed onto your PC's fixed disk, replacing the existing copies. You can do this using VMPCSERV. Your system administrator will tell you what files you need to copy. You can then connect to your host system with VMPCSERV, link to the updated CICS/CMS system disk, and copy the files you need to your local CMS A-disk. From there, you can use the files to replace those on your defined local CICS/CMS system disk.

Note: "Incorporating service changes into your master CICS/CMS system" on page 220 tells system administrators what they need to do to prepare changed components of CICS/CMS so that their PC users can copy them.

Chapter 4. Setting up your CICS/CMS environment

This chapter tells you how to tailor CICS/CMS to suit your requirements.

There are three ways that you can define the environment in which CICS/CMS runs:

1. As CICS/CMS starts, it executes an EXEC called EFHSETP. This EXEC contains statements that set defaults for various aspects of CICS/CMS: programming language used, size of data areas, names of tables, and so on.

   This establishes a general environment for application testing for everyone using that copy of CICS/CMS. Your system administrator will probably set it up to suit the most common requirements of your application programming team.

2. You can change the attributes set by EFHSETP by creating a file called EFHPROF EXEC, usually on your CMS A-disk. From then on, whenever you start CICS/CMS, the EFHSETP EXEC uses the values specified in your EFHPROF to replace some or all of its values.

   IBM supplies a sample EFHPROF EXEC with CICS/CMS. The easiest way to create your own is to copy the sample EFHPROF to your A-disk, and change it to suit your requirements.

3. Occasionally, you might want to change your own environment for testing a particular application, or even part of an application. CICS/CMS therefore offers an interactive means of making temporary changes to your environment. This is explained in "Changing your CICS/CMS environment within a session" on page 119.

The next section tells you how to define your own environment, using the EFHPROF EXEC.

Using EFHPROF to create your own environment

EFHPROF must reside as a file of filetype EXEC on a CMS disk to which you currently have access (usually your A-disk). To help you get started, IBM supplies a sample EFHPROF. You should find it on your CICS/CMS system disk.

EFHPROF contains two types of entries:

1. CMS commands that change the CMS environment established by your PROFILE EXEC. Typically, you might include commands to clean up your CMS disks before running CICS/CMS, by erasing unwanted files such as old error or trace logs.

2. Statements that change the definitions in EFHSETP.

These are commands that provide a new value for one of the predefined parameters in the EFHSETP EXEC. The full form of these commands is:

```
GLOBALV SELECT EFHPARMS SETL 'keyword value'
```

where `keyword` identifies the parameter you want to change, and `value` is what you want the parameter to be.

Figure 25 shows the sample EFHPROF EXEC that IBM supplies as part of CICS/CMS. It shows examples of both types of entry, and a special entry that can save you some work.

```
/* Set user parameters for CICSCMS.   Sample only */
/* Following is an example */
SETPARM = 'GLOBALV SELECT EFHPARMS SETL'
SETPARM 'TWASIZE 1024'
SETPARM 'CICSDSA 160000'
SETPARM 'TRACEFM Z'
'GLOBAL TXTLIB PLILIB COBLIBVS EFHXLIB'
'FILEDEF SYSPRINT TERM'
FILEDEF EXTO DISK EXTOFILE TDQUEUE A '(' LRECL 1000 RECFM F
FILEDEF EXTI DISK EXTIFILE TDQUEUE A '(' LRECL 1000 RECFM V
```

Figure 25. Sample EFHPROF EXEC

The first two lines in Figure 25 are comment lines; they begin with `/*`. Although comments in EFHPROF EXECs are generally optional, you must ***always*** include at least one comment as the first line in your EFHPROF. This is a general rule for all EXECs written using REXX, the System Product Interpreter. If you want to know more about the interpreter, look in the *VM/SP System Product Interpreter User's Guide*.

The meaning of each statement in Figure 25 is as follows:

`SETPARM = 'GLOBALV SELECT EFHPARMS SETL'`

This lets you avoid typing `GLOBALV SELECT EFHPARMS SETL` every time you want to change a parameter. It equates the full form of the command with the abbreviated form, `SETPARM`. Once you've established that, you can use `SETPARM` as the command to change parameters.

We ***strongly*** recommend that you always start your EFHPROF EXEC with a command of this form. It will save you some work, and reduce the risk of typing errors.

`SETPARM 'TWASIZE 1024'`

Requests that the transaction work area (TWA) for any task be allocated as 1024 bytes.

`SETPARM 'CICSDSA 160000'`

Requests that the dynamic storage area (DSA) for application tests be 160000 bytes long.

This overrides the IBM-supplied EFHSETP default of 256000 bytes. If you use CICS/CMS with the sample EFHSETP and the sample EFHPROF, you will always start your CICS test sessions with a DSA of 160000. If you want to

return to the EFHSETP default, you'll need to remove this statement from EFHPROF when you set it up on your A-disk.

```
SETPARM 'TRACEFM Z'
```

Specifies that any CICS/CMS trace output that you request will be written on the Z-disk (your temporary disk), rather than on your A-disk, as specified in the IBM-supplied EFHSETP.

Putting trace output on your Z-disk is quite a good idea, because the output can become quite large, and may well fill your A-disk. However, you have to remember that the Z-disk is always a temporary disk on VM/SP. Unless you define it with VMPCCON (as we suggest in "The steps in organizing PC storage" on page 38), it is also temporary on VM/PC. If the Z-disk is temporary, you will lose it, and all its contents, every time you log off. If you want to keep all, or part, of your trace output, you'll have to copy it from the Z-disk to a permanent disk.

You'll find a general description of the Z-disk in "The CICS/CMS temporary disk (Z-disk)" on page 93.

```
'GLOBAL TXTLIB PLILIB COBLIBVS EFHXLIB'
```

This statement identifies the text libraries to be used for CICS/CMS. In this example, we've identified the libraries for COBOL (COBLIBVS), PL/I (PLILIB), and CICS/CMS (EFHXLIB).

You'll find more information on the `GLOBAL TXTLIB` command in "General information on using high-level languages with CICS/CMS" on page 85.

```
'FILEDEF SYSPRINT TERM'
```

The CMS `FILEDEF` command identifies files to CMS, and associates their filenames with CMS devices, or CMS files. Here, we're specifying that all PL/I output destined for SYSPRINT should go to the CMS device TERM (the user terminal).

```
FILEDEF EXTO DISK EXTOFILE TDQUEUE A '(' LRECL 1000 RECFM F
FILEDEF EXTI DISK EXTIFILE TDQUEUE A '(' LRECL 1000 RECFM V
```

These two CMS `FILEDEF` commands associate the extrapartition data sets, EXTO and EXTI, with the CMS files EXTOFILE and EXTIFILE respectively. They also specify the format of information in those files. You have to provide a suitable `FILEDEF` command for every extrapartition data set you want to use, and EFHPROF is the logical place to put these commands.

You'll find a complete description of how to set up extrapartition data sets in "Extrapartition transient data" on page 57.

You can use `SETPARM` commands to change almost any aspect of your CICS/CMS environment. The list of parameters that you can change is, however, a long one, and we don't want to break up the flow of the guide by including a lengthy reference section here. We have therefore put this list out of the way in Appendix D, "CICS/CMS parameters" on page 241.

Chapter 5. How CICS/CMS handles CICS resources

The primary purpose of CICS/CMS is to give you what you need to develop and test CICS/VS applications interactively. To do this, CICS/CMS handles some of the main CICS/VS resources in ways that you may find unusual.

This chapter describes those resources, paying special attention to ways that they differ on CICS/CMS from the same resources on a CICS/VS system. The resources we describe here are:

- Programs
- Terminals
- Interval control
- Transient data
- Temporary storage
- Files

All the descriptions apply to your use of ***local*** CICS/CMS resources. However, when your applications use the remote server to access resources on a ***remote*** CICS/VS system, their use of those resources is governed by the rules of the CICS/VS system. We deal with using remote resources in a separate part of this chapter, "Remote resources" on page 64.

Programs

To create an executable program in CICS/CMS, you need to produce a file of filetype TEXT from your source program. The source program can be a COBOL, COBOL II, PL/I, or assembler program, and you can create the TEXT file using a single PF key, as shown in "Translating and compiling a CICS program" on page 16, and fully explained in "Translating and compiling/assembling programs" on page 95.

You don't need to do anything else to run a single program. If you have written it in your default language, as defined in EFHSETP or EFHPROF, you can execute it immediately from the execution panel (EFH12).

However, if you want to run a program in the normal CICS/VS way, via a transaction ID or PF key, or if you want to run a pseudoconversational transaction, you will need to create or update a CICS/CMS equivalent of the CICS/VS program control table (PCT). The form of this table, and its contents, are described in "Program tables" on page 70.

Terminals

CICS/CMS is a single-user system. It has no need for the complex facilities that CICS/VS uses to control networks of terminals. This has two basic effects on your use of terminals:

1. There is no equivalent of the CICS/VS terminal control table (TCT) in CICS/CMS.

   If you want to take advantage of some special characteristics of your terminal, you can set the applicable parameters in your EFHPROF EXEC. The terminal parameters in EFHPROF are described in "Terminal control parameters" on page 244.

2. There are some limitations on the way that CICS/CMS supports communication with terminals. For example, it supports only the 3270 terminals that VM/SP supports, and it doesn't support batch data interchange.

BMS

CICS/CMS supports full function BMS, including paging, except in the following cases:

1. Where a BMS function is designed specifically for a multiterminal environment; for example, routing.

   When an application requests such a function, CICS/CMS issues an appropriate CICS error condition. To see what will happen when you run the application on a CICS/VS system (that does support routing), you need to run the application with EDF, and set the condition to NORMAL on the appropriate EDF panel. "Testing CICS/VS features that CICS/CMS does not fully support" on page 129 tells you how to use EDF for this purpose.

2. Where a BMS function is designed for anything but a 3270 terminal.

   What happens when an application tries to use a non-3270 function depends on the function. CICS/CMS will display one of its error handler panels, and you can decide what to do from the information in the panel.

You will find full information on those BMS features that CICS/CMS supports in a restricted way in the *CICS/CMS Application Programmer's Reference Summary.*

BMS paging

There is one area of BMS support where CICS/CMS is quite different from CICS/VS. Although CICS/CMS supports most of the features of BMS paging in CICS/VS, it supports them in its own way, as follows:

- Since CICS/CMS has no system initialization table (SIT), you have no way of defining terminal paging commands. CICS/CMS does not therefore support single keystroke retrieval. The terminal paging commands it supports are those used in the examples in the *CICS-Supplied Transactions* manual (for CICS/VS 1.7), or the *CICS/VS Operator's Guide* (for earlier CICS/VS releases), as follows:

  P/ for page retrieval

C/ for page chaining

- When you've accumulated a number of BMS pages to display, you have to tell CICS/CMS explicitly that you want to display them. You do this by executing the CICS/CMS control transaction, CCMS, with the START option. This in turn executes the CICS/VS transaction CSPS, which CICS/CMS has scheduled to display your pages.

  When your application finishes building BMS pages, the CICS/CMS error handler displays a panel, like the example in Figure 26.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 10:23:11

Program Name: PAGEX                                  Line Number: PAG00980

Message EFH8811I has been generated

After this screen the CICS environment will CONTINUE

The Terminal Operator Paging Transaction CSPS has been scheduled ready for
execution by BMS, during processing of a SEND PAGE command.
CCMS START must be issued to invoke CSPS to display the pages following
completion of the current transaction.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 26. CICS/CMS message telling you how to display BMS pages

As the message suggests, you can continue until the application ends. When it does, you can display all the pages that it has built by clearing the screen and issuing the command:

```
CCMS START
```

You can then use appropriate P/ commands to browse through the pages. When you've finished browsing, and want to return to the execution panel (EFH12) to get on with something else, you once again clear the last screen, and enter:

```
CCMS QUIT
```

Warning: Don't return to the EFH12 panel before you display your BMS pages (using CCMS START). Every time you return to panel EFH12, CICS/CMS clears all scheduled transactions, including CSPS.

Terminal control

CICS/CMS supports only the 3270 terminals that VM/SP supports. (The *VM/SP Terminal Reference* manual tells you what these are.) Under CICS/CMS, your terminal acts as the virtual console.

CICS/CMS also lets you test applications designed to print output on 3270 printer terminals. It doesn't support the terminals themselves; it emulates them using CMS files. See Chapter 10, "Testing applications that print" on page 133 for more information.

Batch data interchange

CICS/VS batch data interchange provides `EXEC CICS ISSUE` commands that let your applications communicate with the 6670 logical unit, and with the batch logical units of the 3770 and 3790 subsystems.

CICS/CMS does not support these units, and does not therefore support batch data interchange. If you use any `EXEC CICS ISSUE` command, your program will abend with an AEY9 code, indicating that you have used a function that CICS/CMS does not support.

Interval control

You can use CICS/CMS to test an application that uses all the interval control commands defined in the *CICS/VS Application Programmer's Reference Manual.* However, CICS/CMS doesn't execute some of the commands, and those it does, it treats in its own way, as follows:

- It doesn't execute the EXEC CICS commands, `POST`, `DELAY`, or `WAIT EVENT`. It displays a message telling you that it doesn't support them, and, after you press ENTER, carries on with the next statement in the program.
- It supports the `EXEC CICS CANCEL` command, but not in any of the uses that let you leave out the `REQID` option.
- It supports the `EXEC CICS START` command, but ignores any values you give in the `INTERVAL` or `TIME` option. We describe below how this affects your use of interval control.

The *CICS/CMS Application Programmer's Reference Summary* tells you which options CICS/CMS supports on each of the commands.

When an application issues an `EXEC CICS START` command with an `INTERVAL` or `TIME` option, CICS/CMS ignores any value in the option and assumes the default of 0. In CICS/VS, this means "start the transaction immediately". CICS/CMS, however, flags the transaction named in the `START` command as being "ready for execution" (***scheduled***), and leaves it to you to start it, using the CICS/CMS control transaction, CCMS.

To make this clearer, let's look at an example. Suppose you are using CICS/CMS to test an application that contains a program (called STARTER) that issues the command:

```
EXEC CICS START TRANSID('TONY') INTERVAL (010000)
```

Under CICS/VS, that would start the TONY transaction in one hour's time. Under CICS/CMS, as soon as the command is executed, CICS/CMS displays the panel shown in Figure 27.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 15:25:46

Program Name: STARTER                                Line Number: STA00360

Message EFH8803I has been generated

After this screen the CICS environment will CONTINUE

An unsupported option has been specified on the START command.
It is ignored and INTERVAL(0) is forced.
Unsupported option is : INTERVAL with non-zero value




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 27. CICS/CMS display for an EXEC CICS START command

This message alerts you that CICS/CMS has set your `INTERVAL` option to 0. If you press ENTER to resume the test, you'll get another panel, as shown in Figure 28.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/03/86 15:25:52

Program Name: STARTER                                Line Number: STA00360

Message EFH8810I has been generated

After this screen the CICS environment will CONTINUE

The START command has successfully completed.
The transaction TONY is now scheduled ready for execution.
It may be invoked by issuing the CCMS START command.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 28. CICS/CMS display for a successful START command

As the message states, CICS/CMS has scheduled the transaction named in the `START` command and you can execute it at any time ***before the end of your CICS test session***, by issuing the command:

```
CCMS START
```

We've stressed that you must do it before the end of your CICS test session because, every time you return to panel EFH12, CICS/CMS clears all scheduled transactions. You therefore have two choices of when to execute the scheduled transaction:

1. You can wait until the application you're testing ends. You can then enter `CCMS START` as the next transaction ID.

2. You can escape from the application (using PA2) at any time after CICS/CMS has scheduled the transaction. You can then issue `CCMS START` from the `Transaction` line on panel EFH122. When the scheduled transaction ends, you can return to panel EFH122, then press PF3 to return to your suspended application.

   You can't, however, use this technique if the scheduled transaction is the first in a pseudoconversation. When a transaction that you've started from the escape panel ends, your next keystroke takes you back to the escape panel. You wouldn't be able to run any of the other transactions in the pseudoconversation.

You can also use interval control remotely from CICS/CMS, by running an application that starts a transaction defined as being on your remote CICS/VS system. We describe this in "Using interval control remotely" on page 66.

Transient data

CICS/CMS supports all three types of transient data: intrapartition, extrapartition and remote.

Intrapartition transient data

For each intrapartition transient data destination that your applications use, CICS/CMS creates a separate CMS file with a predefined filename, filetype, filemode, and maximum record length. You can set the filetype and maximum record length for all your intrapartition destinations in your EFHPROF EXEC, as described below. You can set the filename, filetype, and filemode for individual destinations in the transient data destination table, as described in "Transient data destination tables" on page 73. If you don't define the filemode of an intrapartition destination file that you write to, CICS/CMS writes it on your A-disk. If you read from an intrapartition destination without defining a specific filemode for it in the transient data destination table, CICS/CMS will search through your attached CMS disks until it finds a file with the defined filename and filetype.

The default maximum record length and filetype are defined in the IBM-supplied EFHSETP EXEC by the statements:

```
SETPARM 'INTRAMRL 32763'
SETPARM 'INTRAFT CICSTDI'
```

These define the maximum record length (32763 bytes) and CMS filetype (CICSTDI) for ***all*** intrapartition queues. If you want to change either of these defaults, you can do so using a `SETPARM` command in your EFHPROF EXEC, as explained in "Using EFHPROF to create your own environment" on page 47.

If you want to change the filename or filetype, or specify a particular filemode, for individual intrapartition queues, you will need to use a CICS/CMS destination table. To find out how to do this, see "Transient data destination tables" on page 73.

Overlength records

If you try to use records longer than your defined maximum record length for an intrapartition queue, you will raise an IOERR condition. To cater for this, you might consider including an `EXEC CICS HANDLE CONDITION` command in any applications that use intrapartition queues.

Read and write pointers

Whenever you read from an intrapartition queue, a read pointer is set to the next record. Whenever you write to an intrapartition queue, a write pointer is set to the point where the next record will be written. The interaction between CICS and CMS has the following effects on the positions of these pointers:

- The ***write*** pointer.

  The write pointer will always be at the end of the CMS file associated with an intrapartition destination. You cannot change this by closing the file, by stopping your CICS/CMS session, or even by logging off. The only way you can reset the write pointer is by erasing the CMS file itself, either with CMS's `ERASE` command, or with an `EXEC CICS DELETEQ` command on the intrapartition destination.

- The ***read*** pointer.

  If you issue a CMS `FINIS` command on the CMS file associated with an intrapartition destination, CICS/CMS resets the read pointer to the start of the file.

  If you return to the execution panel (EFH12), CICS/CMS resets the read pointers of ***all*** your current intrapartition destinations to the start of the CMS files associated with them.

  Entering the CMS subset from anywhere in CICS/CMS closes all your files, but does not reset any read pointers in intrapartition destinations.

Extrapartition transient data

Extrapartition transient data queues let you transfer sequential information between your applications and external sequential resources such as reader or tape files.

Before you can use an extrapartition queue, you must define it in two ways:

1. You must associate the queue name with a CMS virtual device using CMS's `FILEDEF` command. For input queues, you can define disk or tape files, or your virtual reader. For output queues, you can define disk or tape files, or your virtual printer or punch.

   For more information on this, see "Defining extrapartition queues to CMS" on page 58.

2. You must define the queue in the CICS/CMS transient data destination table. The definition must include the queue name, and an E to show that it is extrapartition. You also have to specify O if it is an output queue. Unless told otherwise, CICS/CMS assumes that extrapartition destinations are input queues.

   You will find some sample definitions in Figure 32 on page 74.

Defining extrapartition queues to CMS

Before you can refer to any extrapartition queue in an application program, you must associate that queue with a CMS virtual device.

For each extrapartition queue you want to use, you must execute a CMS `FILEDEF` command ***before*** you start a CICS test session. When you initialize CICS, by starting a transaction or program from panel EFH12, it opens all the files that it will use, including those associated with extrapartition queues. You can't use `FILEDEF` during a CICS test session, by executing it from the escape panel (EFH122), because the file that it associates with an extrapartition queue will not be open.

The `FILEDEF` command for each extrapartition queue specifies:

- The 4-character name of the extrapartition queue.
- The virtual device with which you want to associate that queue: `TERMINAL`, `PRINTER`, `PUNCH`, `READER`, `DISK`, `DUMMY`, `GRAF`, or `TAPn` (where `n` is the virtual tape drive number).

  Note: There are some special rules you need to know when using extrapartition queues to get input data for a program from your virtual reader. We describe these in "Reading data from your virtual reader" on page 118.
- The filename, filetype, and filemode of the CMS file, if the device is `DISK`.
- The record format: `F` for fixed, `V` for variable.

  Note: If you specify a format of `V`, you must give the file a maximum record length 4 bytes longer than you need. These 4 bytes contain control information. The second and third examples below show this rule in action.
- The maximum length of record you will write to the chosen device.

  If your application tries to use records longer than this, it will raise a LENGERR condition. We recommend that you include an `EXEC CICS HANDLE` command in your application to deal with this condition.

Printing from extrapartition queues

To test an application that writes data to a transient data queue for printing on a system printer, you need to:

1. Associate the queue with your virtual printer, or with a CMS file. You do this using the CMS `FILEDEF` command, as described above, associating the queue with the virtual device PRINTER (for the virtual printer) or DISK (for a CMS file). The third example in "Example FILEDEF definitions for extrapartition queues" on page 59 shows a `FILEDEF` command to associate an extrapartition queue with the virtual printer.

2. Define the queue in your CICS/CMS destination table file as E (extrapartition) and O (for output).

CICS/CMS will send to the printer everything that you write to that queue in a CICS test session. This doesn't necessarily mean running a single program or transaction, since you might escape during the execution and run other transactions or programs from panel EFH122.

Note that, if you're using a PC with a local printer, VM/PC usually assumes that your local printer is your virtual printer. You can change this to a remote system printer, using the SPOOL command, as explained in the *VM/PC User's Guide.*

Example FILEDEF definitions for extrapartition queues

```
FILEDEF EXT1 DISK EXQIN FILE D (RECFM F LRECL 80
```

This associates queue EXT1 with the CMS file EXQIN, which is on your D-disk with a filetype of FILE. It contains fixed-length records, 80 bytes long.

```
FILEDEF EXT2 DISK EXQOUT DATA A (RECFM V LRECL 136
```

This associates queue EXT2 with the CMS file EXQOUT, residing on your A-disk with a filetype of DATA. Its records are variable in length, with a maximum data length of 132 bytes. We've defined an extra 4 bytes for the control information needed for variable records.

```
FILEDEF EXT3 PRINTER (RECFM V LRECL 136
```

This associates queue EXT3 with the virtual printer. We advise a maximum data length of 132 (136 including the variable record control information), which suits most kinds of physical printer.

Note: If you associate a queue with the reader, punch, or printer, make sure that you define the queue appropriately in the destination table. For example, suppose you used a FILEDEF to associate a queue with the virtual printer (as shown in the third example), and then defined that queue as being for input (I) in the destination table. As soon as you entered the CICS environment from panel EFH12, you would get error messages from both CMS and CICS/CMS. CICS/CMS would ignore the definition of that queue. Although you could continue your test session, you would not be able to use the queue until you corrected the destination table, and reentered the CICS environment.

"Transient data destination tables" on page 73 tells you how to define extrapartition queues to CICS/CMS. You can find out more about the FILEDEF command in the *CMS User's Guide.*

Remote transient data

This is discussed in "Remote resources" on page 64.

Temporary storage

CICS/CMS supports all the temporary storage features of CICS/VS, except that all temporary storage is set up in main storage. If an application makes a temporary storage request with the `AUXILIARY` option, CICS/CMS assumes `MAIN`, and displays an error handler message.

You don't usually need a temporary storage table to use temporary storage queues. The only time you need one is to define temporary storage queues on a remote CICS/VS system.

For more information on using remote resources, see "Remote resources" on page 64. For more information on the CICS/CMS temporary storage table, see "Temporary storage tables" on page 76.

Local data files

Most CICS applications access data files at some point, using the CICS/VS file control application programming interface (API). The files are usually VSAM files or DL/I data base files. CICS/CMS has its own way of supporting the file control API, designed to make it as easy as possible for you to create and maintain data files.

Note: The title of this section is ***local*** data files. If you want to use DL/I data bases, or CICS/VS VSAM files, you have to access them on a remote CICS/VS system, using the remote server. We tell you how to define remote files to CICS/CMS in "Remote resources" on page 64.

How CICS/CMS supports VSAM files

VSAM supports three types of files, as follows:

- Key sequential data sets (KSDS), accessed by ***key***
- Relative record data sets (RRDS), accessed by ***relative record number***
- Entry sequential data sets (ESDS), accessed by ***relative byte address***.

CICS/CMS simulates support for all three types, to make it as easy as possible for you to use data with your test applications.

To support KSDS, CICS/CMS uses the ***keyed*** format. Each keyed file consists of two files: one contains the data, the other is an index file for that data.

To support both RRDS and ESDS, CICS/CMS uses a single file format, ***nonkeyed***, accessed by relative record number.

The only restrictions that CICS/CMS places on the maximum size of files, or of records within files, are those that CMS imposes, as follows:

- You can store a file only if you've got room for it on the disk on which you're trying to put it.
- The maximum record size for any local CICS/CMS data file is 32763 bytes.

Note, however, that when you use a keyed file in a CICS test session, CICS/CMS loads the index into virtual storage. If you test applications that use very large keyed files, you may need a larger virtual machine than the recommended 2 megabytes.

You don't have to define files in a CICS/CMS file table if they are local files that already exist. You only have to define a data file in the CICS/CMS file table in the following situations:

- When you want to use a data file on a remote CICS/VS system without using the SYSID option on EXEC CICS commands
- When an application will create a new file
- When you want to increase the maximum record length of an existing file containing variable length records.

You can find out more about when to use file tables, and how to define files within them, in "File tables" on page 77.

In the next sections we deal with each of the CICS/CMS data file types in turn, describing their exact format, and what you need to consider when using them.

The CICS/CMS keyed file format

CICS/CMS keyed files simulate VSAM KSDS support. Each CICS/CMS keyed file consists of two files: the index file and the data file. The filenames of these files must be the same. It's the filetype that distinguishes the index file from the data file.

An index file has a filetype of EFHVINDX, and a data file has a filetype of EFHVDATA. Both files must reside on the same CMS disk. As in VSAM KSDS, all keys in one keyed file must be unique, and of the same length.

To make it as easy as possible for you to use keyed files, CICS/CMS provides utilities to convert either local CMS files, or VSAM files on a remote CICS/VS system, into the two-file structure. These utilities are fully described in "Preparing local data files" on page 109.

Keyed file record structure

Records in an EFHVDATA file can be of fixed or variable length. You can define the record type and length in a CICS/CMS file table. If you don't include a definition of a keyed file in a table, CICS/CMS uses the applicable CMS values.

Records in an EFHVINDX file are a maximum of 512 bytes long. Each record contains one or more entries, depending on the length of the key. For example, if the keys are only 5 bytes long, the entry for each key will be 22 bytes long, with 2 bytes of padding, as explained below. Each record on the index file will therefore contain 21 entries.

Each entry in an index record is of fixed length, and contains:

1. A forward pointer to the next key in the sequence (4 bytes long).
2. A backward pointer to the previous key in the sequence (4 bytes long).

3. The relative record number (RRN) of the associated record in the EFHVDATA file (4 bytes long).

4. The length of the associated record in the EFHVDATA file (4 bytes long).

5. A flag to say whether the record in the EFHVDATA file is current or deleted. As we explain below, when you delete a record from the data file, CICS/CMS doesn't actually remove it. It changes this flag from X'00' (current) to X'80' (deleted). This flag is 1 byte long.

6. The fixed length key (between 1 and 255 bytes long).

7. Padding, to ensure that the entry that follows this one starts on a word boundary. There can be up to 3 bytes of padding.

The data and index files both start with a control record, as follows:

- The first 16 bytes, in both files, contain the date and time when you created or last changed the file. These values ***must*** be identical in both files. If they get out of step, you'll get a message from CICS/CMS when you try to access data from the file, warning you that it can't use the file until you correct the control records.

- In the index file, the rest of the record contains information that CICS/CMS needs to retrieve information from the data file. If this information is corrupted in any way, CICS/CMS won't be able to use the keyed file.

  In the data file, the rest of the record may contain some data, or nothing. In either case, the contents of the data file control record are unimportant to CICS/CMS, and you can ignore them.

In view of the importance of the date and time parts of both control records, and of the "retrieval" information in the index file control record, we strongly advise that you don't try to change CICS/CMS keyed files directly, using an editor.

If you ever need to re-create either of the control records, you should re-create both files, using the EFHUCMS1 utility either on the original CMS file, or on your CICS/CMS EFHVDATA file.

Deleting records from keyed files

CICS/CMS handles `EXEC CICS DELETE` commands on keyed files in a different way from CICS/VS. It removes the pointer to the deleted record from the index file, but it doesn't remove the record from the data file. You may therefore find, if you look at a data file using your editor, that records that you have deleted appear to be still there.

If you use a keyed file frequently, adding and deleting records, the data file can become quite large. "Using CCU2 to reorganize CICS/CMS keyed files" on page 116 tells you how to reorganise CICS/CMS keyed files to leave only active records, in ascending order of keys, in the data file.

The CICS/CMS nonkeyed file format

CICS/CMS uses CMS files with a filetype of EFHVNONK to support both entry sequential data sets (ESDS) and relative record data sets (RRDS).

Records in an ESDS file can be of fixed or variable length. Records in an RRDS file must be fixed.

You can define whether a nonkeyed file is ESDS or RRDS, and specify the record type and length, in a CICS/CMS file table. If you don't include a definition of a nonkeyed file in a table, CICS/CMS uses the applicable CMS values for the record format. It determines whether the file is ESDS or RRDS from the way that you access it the first time you use it in a CICS test session.

For example, if you run an application that accesses a nonkeyed file using the RRN option, and then escape, and run another application that accesses the same file, using the RBA option, the access in the second application will fail. However, if you let the first application finish, and return to panel EFH12 before running the second application, the access in the second application will work, because you are in a different CICS test session.

Differences between CICS/CMS and CICS/VS

There are a number of differences between the way that CICS/CMS supports nonkeyed files, and the way that CICS/VS supports the RRDS and ESDS files they represent, as follows:

1. CICS/CMS supports the full file control API, with one exception. Under CICS/VS, you can use `EXEC CICS DELETE` commands to delete records from RRDS files, but not from ESDS files. Under CICS/CMS, you cannot use the `DELETE` command on any nonkeyed file. If you try to do so, you will raise an INVREQ condition, and get a message from CICS/CMS.

2. Whether you specify `RBA` or `RRN` in a file control command, CICS/CMS treats the value you give in `RIDFLD` as a ***relative record number***. This will not affect the way you use RBA records, unless your application tries to compute one RBA from another.

3. CICS/CMS writes all records that you add to a nonkeyed file to the end of the file, and returns a new value to the application in `RIDFLD`. It ignores any value you give in the `RIDFLD` option of your `EXEC CICS` commands, so you will never get a DUPREC condition when you use nonkeyed files. This also means that you can't create nonkeyed files in random order, as you can RRDS files in CICS/VS.

   You can test for DUPREC on RRDS files, by running with EDF on, and changing the condition raised by the second write from NORMAL to DUPREC. You'll need to remember, however, that the application will actually have added the record to the file.

4. You must use the `EQUAL` option on all `EXEC CICS READ`, `STARTBR`, and `RESETBR` commands that use RRDS files. If you leave it out, those commands will raise an INVREQ condition, and produce a CICS/CMS message.

How to interpret EIBRCODE in CICS/CMS

There is a difference between CICS/CMS and CICS/VS in the meaning of the EIBRCODE value for file control in the exec interface block.

EIBRCODE is a 6-byte field. Byte 0 contains the condition code, and byte 1 contains the VSAM return code. These are the same for CICS/CMS as for CICS/VS. However, In CICS/VS, when there's a problem with a VSAM file, byte 2 of the EIBRCODE contains the VSAM error code. In CICS/CMS, because VSAM access is simulated using CMS file facilities, this error code is replaced by the return code from the CMS file command on which the problem arose. You'll need to look in the *VM/SP CMS Command and Macro Reference* manual to find out what these codes mean. You'll find more about how CICS/CMS uses EIBRCODE in the *CICS/CMS Application Programmer's Reference Summary*.

Remote resources

During a CICS/CMS session, you can use any of the local CICS/CMS resources described in the preceding sections. If you have access to a remote CICS/VS system, you can also use resources on that system, or on another CICS/VS system *daisy chained* to that system, through the *remote server*.

The remote server runs on the remote CICS/VS system as a transaction (CEHS). It accepts requests for remote resources from your applications and passes them to the remote CICS/VS system to which you have access.

Why would you want to use resources on a remote CICS/VS system? The main reasons are:

- To test applications that use DL/I data bases. CMS doesn't support DL/I, so you ***must*** send all requests for DL/I data bases to a remote CICS/VS system.

- To test applications that use resources in ways that you can't test locally using CICS/CMS.

  For example, you might want to test an application that computes one RBA from another when accessing ESDS files, or that accesses keyed files with alternate indexes, or that writes temporary storage files to auxiliary storage. When you use such resources remotely, the rules governing their access are those of CICS/VS, rather than those of CICS/CMS.

- To test applications that use interval control to start transactions that aren't local to CICS/CMS, but are stored on your remote CICS/VS system.

Before you can use a remote resource from an application, there are two things you must do:

1. You must tell CICS/CMS that the resource you want is on a defined remote CICS/VS system, rather than on your local CICS/CMS system.

   This is explained below.

2. You must connect to the remote CICS/VS system, start the remote server transaction (CEHS), and return to your CICS/CMS system.

Since this is something you won't need to do until you run an application that needs a remote resource, we deal with it as part of the development process, in "Starting and using the remote server" on page 105.

There are two ways of telling CICS/CMS that a resource is remote, as follows:

1. You can define the resource as remote in one of the CICS/CMS tables.
2. You can use the `SYSID` option on any EXEC CICS commands that refer to that resource. If you do this, you don't have to change any CICS/CMS tables. However, you'll have to change the application to remove the `SYSID` option when you transfer it to your CICS/VS system.

Defining remote resources in CICS/CMS tables

The resources you can define as being remote are:

- DL/I data bases
- VSAM files
- Transient data queues
- Temporary storage queues.

DL/I data bases

You have to define all DL/I program specification blocks (PSBs) that you want to access in a file called the PSB directory. Like all CICS/CMS tables, this is simply a CMS file with a particular filename and filetype, as defined in the EFHSETP EXEC or your own EFHPROF EXEC.

We describe the PSB directory with the other CICS/CMS table descriptions, in "PSB directories" on page 78.

Remote VSAM files

If you want to use a VSAM file that resides on the remote CICS/VS system, you must define it as being of type R in a CICS/CMS file table. See "File tables" on page 77.

Remote transient data queues

If you want to use a remote transient data queue, you must define it as being of type R in a CICS/CMS transient data destination table. See "Transient data destination tables" on page 73.

Remote temporary storage queues

If you want to use a remote temporary storage queue, you must define it in a CICS/CMS temporary storage table. See "Temporary storage tables" on page 76.

Using the SYSID option

There are two circumstances in which an application you're testing under CICS/CMS might have the SYSID option on EXEC CICS commands:

1. Because you're using the SYSID to access resources remote from CICS/CMS
2. Because the application will access resources remote from the CICS/VS system on which it runs when you put it into production.

In general, we recommend that you don't use SYSID for the first circumstance. If you do, you'll have to change the application to remove the SYSID option before moving it into production. It's better to define the resources you want to use in the appropriate CICS/CMS table(s). However, if you want to use interval control remotely, you'll have to use SYSID. If you do, note that the remote system name you give must be the name by which CICS/CMS refers to your remote CICS/VS system. This name is defined in the REMSYSID parameter in EFHSETP. The IBM-supplied name is REMT.

For the second circumstance, you have a choice. You can access the resources you need on the remote CICS/VS system, by giving the REMSYSID name in the SYSID option. Alternatively, you can set up the resources locally on CICS/CMS, and access them by giving your local system name in the SYSID option. This name is defined in the LOCSYSID parameter in EFHSETP. The IBM-supplied default is LOCL.

Using interval control remotely

You can use the remote server to start a task, or cancel a previous interval control request, on the remote CICS/VS system. You don't need to change any CICS/CMS tables to do this. You simply include the SYSID option in your EXEC CICS START or CANCEL commands. The remote server will pass the request to the remote system. Note, however, that, when you transfer the tested application to your CICS/VS system, you'll have to remove the SYSID option.

The remote system name which you give in the SYSID option must be the name by which CICS/CMS refers to your remote CICS/VS system. This name is defined in the REMSYSID parameter in EFHSETP. The lBM-supplied name is REMT.

If the transaction you name in the command doesn't exist on the remote CICS/VS system, you'll get a message from CICS/CMS. Otherwise, you have no way of communicating with that transaction once you've issued the EXEC CICS START command.

CICS/CMS tables

In each of the earlier sections dealing with individual resource types, we mentioned briefly those occasions when you need to define the resource in a CICS/CMS table. There are five kinds of tables in CICS/CMS, as follows:

- Program tables
- Transient data destination tables
- Temporary storage tables

- File tables
- PSB directories.

All CICS/CMS tables are CMS files containing resource definitions. When you need to define a resource of any kind to CICS/CMS, you create or update one of these files, using your editor.

You also need to tell CICS/CMS which CMS files are the CICS/CMS tables. You do this using parameters in the EFHPROF EXEC that define the filename, filetype, and filemode of the CMS file for each kind of table. Appendix D, "CICS/CMS parameters" on page 241 describes all the parameters for defining tables. "Fully defining CMS files as CICS/CMS tables" and "Partially defining CMS files as CICS/CMS tables" describe the ways in which you can use these parameters.

Fully defining CMS files as CICS/CMS tables

If you want to use a particular CMS file as the default for a CICS/CMS table, you define the filename, filetype, and filemode of that file in your EFHPROF EXEC. For example, if you include the statements:

```
SETPARM 'PROGFN APP1'
SETPARM 'PROGFT PROGTAB'
SETPARM 'PROGFM A'
```

in your EFHPROF, you define your CICS/CMS program table as being the CMS file APP1 PROGTAB A. CICS/CMS will always use this file as your program table, unless you tell it otherwise using panel EFH121, as explained in "Defining CMS files as CICS/CMS tables on the EFH121 panel" on page 69.

Partially defining CMS files as CICS/CMS tables

You can specify only the filetype of the file to be used as the default for a particular kind of CICS/CMS table, using a partial file definition in your EFHPROF EXEC. For example, if you include the statements:

```
SETPARM 'PROGFN .'
SETPARM 'PROGFT PROGTAB'
SETPARM 'PROGFM *'
```

in your EFHPROF, you define CICS/CMS program tables as having a default filetype of PROGTAB, and residing on any disk to which you currently have write access.

By defining PROGFN as "period" (.), however, you set the default filename to null. This lets you set up different program tables for different applications, giving each a unique filename, and the filetype PROGTAB. Before you test any applications that need a program table, you'll need to tell CICS/CMS which CMS file to use as the table during that test. There are three ways of doing this:

- Using PF9 on one of the EFH13 panels
- Using PF9 on the EFH11 panel
- Using panel EFH121.

Defining CMS files as CICS/CMS tables on the EFH13 panels

Selecting option 3 on panel EFH1 displays panel EFH13, as shown in Figure 10 on page 22. From this panel, you can choose to list any of your table types.

For example, suppose you've defined your program table as shown in "Partially defining CMS files as CICS/CMS tables" on page 67, and then set up three CMS files on your A-disk, with the filetype PROGTAB, and the filenames APP1, APP2, and APP3. If you select option 2 on panel EFH13, CICS/CMS will display the following panel:

```
 EFH131              FILELIST:* PROGTAB *                  Line 1 of 3

Cmd   Filename Filetype Fm Format Lrecl    Records     Blocks   Date     Time
      APP2     PROGTAB  A2 F         80         27          1  3/03/86  9:50:05
      APP3     PROGTAB  A2 F         80         22          1  3/03/86  9:49:52
      APP1     PROGTAB  A2 F         80         40          1  3/03/86  9:46:28









PF1=Help      2=Refresh  3=End       4=Sort(date)    5=Sort(size)      6=Edit
PF7=Backward  8=Forward  9=Install  10=Cursor       11=               12=

====>
```

Figure 29. CICS/CMS program table display

Let's assume that, in this session, you want to test an application that uses programs defined in APP2 PROGTAB. You need to tell CICS/CMS to use APP2 PROGTAB for the session.

You move the cursor beside APP2 PROGTAB, press PF9, and CICS/CMS "installs" that file as the program table for the rest of the CICS/CMS session, or until you change it (either by using PF9 again, or by using panel EFH121, as described in "Defining CMS files as CICS/CMS tables on the EFH121 panel" on page 69).

Defining CMS files as CICS/CMS tables on an EFH11 panel

You can use PF9 to install tables from panel EFH11 in the same way as from an EFH13 panel. On EFH11, however, you have to be more careful. PF9 works only on files with the filetype that you've defined for a CICS/CMS table. On the "EFH13n" panels this presents no difficulty, because they only list files that you've defined, either fully or partially, as containing tables. On an EFH11 panel, however, you might press PF9 against a file of the wrong filetype. If you do, CICS/CMS will issue the message:

```
EFH9090S type is not a valid filetype.
```

where `type` is the filetype of the file for which you pressed PF9.

Defining CMS files as CICS/CMS tables on the EFH121 panel

On panel EFH121, you can complete partial table definitions, or change full definitions.

To see how, let's continue with the program table example from the point we left it in "Defining CMS files as CICS/CMS tables on the EFH13 panels" above. Suppose that, during the CICS/CMS session in which you're using the table APP2 PROGTAB, you decide that you want to test an application that needs the definitions in APP1 PROGTAB. You could return to panel EFH1, then go to panel EFH131 and use PF9 on panel EFH131, as described in "Defining CMS files as CICS/CMS tables on the EFH13 panels" on page 68. It's more convenient, however, to use panel EFH121, which lets you define the new program table without going back through earlier panels.

Pressing PF7 on panel EFH12 will bring up a display looking something like the one shown in Figure 30.

```
EFH121                        PARAMETER DEFINITION

 Trace                      ===> NO           Yes,No
 EFHUSTG table entries      ===> 500          100-1000
 Trace table filemode       ===> Z            2 characters
 Trap program checks        ===> YES          Yes,No
 DEBUG program              ===>              1-8 characters
 Dynamic storage            ===> 160000       50000-4000000
 Language                   ===> COBOL        COBOL, COBOL2, ASSEMBLE or PLI
 TWA size                   ===> 1024         1024-32767
 SYSID for remote system    ===> REMT         4 characters

 Names of resource tables:       Filename Filetype Filemode
   Program/Transaction      ===> APP2     PROGTAB  *
   File                     ===>          EFHTFILE *
   Transient Data           ===>          EFHTTD   *
   Temporary Storage        ===>          EFHTTS   *
   PSB Directory            ===>          EFHTPDIR *

                                Keyword   Value
 Additional Parameter       ===>


PF1=Help          PF3=End           PF12=Quit           PA2=CMS Subset
```

Figure 30. Changing a program table on the EFH121 panel

You move the cursor to the line containing the program table definition, change APP2 to APP1, and press PF3 to return to panel EFH12. You can then start testing applications that use the definitions in APP1 PROGTAB immediately.

Any changes you make using panel EFH121 stay in effect for the rest of the CICS/CMS session, or until you make another change using either panel EFH121 or the PF9 (Install) key.

You'll find a complete description of everything you can use panel EFH121 for in "Changing your CICS/CMS environment within a session" on page 119.

The general form of table entries

Entries in CICS/CMS tables have strictly defined forms. In particular, the entries are column-dependent (that is, each field in each entry has to start in a defined column). The table descriptions that follow this general introduction tell you the forms the entries must take.

To make it easier for you to specify entries correctly, CICS/CMS provides an XEDIT macro (EFHTABH) that puts descriptions of the entries at the front of a table file, and sets tabs for the relevant column positions.

For example, suppose your defined filetype for transient data tables is EFHTTD, and you are currently editing a file called APP1 EFHTTD. If you enter the command EFHTABH from your XEDIT command line, you'll see the following lines appear at the start of your file:

```
* CICS/CMS Table for Transient Data
*
* Column definitions
* For intrapartition
* 1-4    6      8-15  17-24  26-27
* Local  Intra  File  File   File
*  name   flag   name  type   mode
* For extrapartition
* 1-4    6      8-15
* Local  Extra  Input/
*  name   flag  Output
* For remote
* 1-4    6      8-11   13-16
* Local  Intra  LRECL  Remote
*  name   flag          name
*    6 8    13  17       26
```

Notes

1. EFHTABH is an ***XEDIT*** macro. If you are not using XEDIT as your editor, you can't use EFHTABH.

2. You can tell that the lines that EFHTABH inserts are comments, because they begin with an asterisk followed by a space. You can insert comments in all CICS/CMS tables in this way.

The sections following describe the various types of CICS/CMS tables.

Program tables

There are three circumstances where you need to create or update a table of programs, as follows:

- When you are creating a pseudoconversational application, consisting of more than one program, associated with one or more transaction IDs. Here, you have to provide program table entries that associate the programs with the applicable transaction IDs.

- When you want to run a program written in a language other than your default language

- When your program name is different from the name of the CMS file that contains it.

  Here, you have to give the program's name in the program table.

Figure 31 shows the general form of program table entries, by giving some examples.

```
* CICS/CMS Program and Transaction table format
*
* 1) Associate PROG1 with the transaction, TRAN
*
PROG1      PROG1      COBOL      INCLUDE    TRAN
*
* 2) PROG2 is to be executed by pressing PF9.
*    It is stored in the file, PROGFILE.
*
PROGFILE   PROG2      COBOL      INCLUDE   /PF09
*
* 3) PRPROG is a PL/I program, associated with two
*    transactions, RITE and BIGY.
*
PRPROG     PRPROG     PL/I       INCLUDE    RITE
                                             BIGY
```

Figure 31. Sample program table

Each program has a single entry in the file, consisting of up to five fields, as follows:

1. Filename of the file containing the program
2. Program entry point name
3. Language in which the program is written
4. Load method
5. Transaction identifier or PF key

Filename field

The first field in each program table entry gives the filename of the CMS file containing the program. It must start in column 1.

Entry point field

The second field in each program table entry gives the program's entry point. This must start in column 11.

You can leave this out if the name of the program is the same as the filename of the file that contains it. The program name is what you give in a PROC statement for PL/I, a PROGID statement for COBOL or COBOL II, or as the CSECT name for assembler.

Example 2 in Figure 31 defines an entry point name different from the filename.

Language field

The third field in each program table entry defines the language in which you have written the source program. This must start in column 21.

Valid entries are:

PLI	for PL/I
COBOL	for COBOL
COBOL2	for COBOL II
ASSEMBLE	for assembler

Load method field

The fourth field in each program table entry defines the method by which the program is to be loaded into memory. This must start in column 31.

If you leave the field blank, CICS/CMS assumes INCLUDE as the load method, which means that the named program will be loaded into the user memory. This is the way we recommend that all user programs should be loaded. You can find the address of a loaded program by finding the INCLUDE for it in the LOAD MAP file.

The other choice is NUCXLOAD. This loads the program into the CMS nucleus, as a nucleus extension. This is the way that CICS/CMS loads its own modules. You can find the address of a nucleus extension with the `NUCXMAP` command.

In general, we recommend that you don't use NUCXLOAD for anything but assembler programs (if at all), for the following reasons:

1. If you want to load one of your COBOL, COBOL II, or PL/I programs with NUCXLOAD, you will have to link-edit the program with all the library routines it needs. This means creating new files containing the link-edited forms of the programs; these files will take up a lot of disk storage.

2. Every time you load the program, you also load the library routines, using up a lot of virtual storage.

3. When you end a CICS/CMS session, CICS/CMS doesn't release any nucleus extensions. This includes programs that you've loaded into the nucleus with NUCXLOAD. To release the storage those programs occupy, you have to use a `NUCXDROP` command for each one.

If, after considering the points above, you still want to load your programs with NUCXLOAD, you'll need to read the information below to find out how to link-edit your programs.

The way you link-edit depends on whether you're using VM/SP or VM/PC, and which version of VM/SP you're using, as follows:

- ***On VM/SP Release 3***, you copy the TEXT file containing your translated and compiled program to a new file (filetype TEXT), and add the following commands to the end of the file:

```
INCLUDE EFHLIB(DFHEPI)
ENTRY program
NAME program(R)
```

where `program` is your program name.

You then put the new file in the load library, EFHPRIV LOADLIB, using the commands:

```
FILEDEF EFHLIB DISK EFHXLIB TXTLIB *
LKED fn (LIBE EFHPRIV
```

where `fn` is the filename of the file you just created.

- ***On VM/PC or VM/SP Release 4***, you have to make a relocatable module from the file containing your translated and compiled program, using the commands:

```
LOAD fn (RLDSAVE
GENMOD fn
```

where `fn` is the filename of your TEXT file.

Transaction or PF key field

The fifth field in each program table entry gives the transaction identifier or PF key that initiates the program. Transaction IDs must start in column 41; PF key definitions must start with a / in column 40, as shown below.

If you want to associate a single program with more than one transaction ID, you can list the IDs against the program name, as shown in example 3 in Figure 31.

Note: The program table is the only one in which you can do this. In all other tables, if you leave out a field in a definition, CICS/CMS applies some defined default.

The form of the definition for a PF key is:

```
/PFxx
```

where xx is the PF key number. A typical PF key definition is shown in example 2 in Figure 31.

Transient data destination tables

As explained earlier, you usually need to define transient data destinations in CICS/CMS only if they are extrapartition or remote. You create intrapartition destinations as CMS files, simply by referring to them in your applications. You define the default filetype of these files in your EFHPROF EXEC. The only time, therefore, that you need to include a table entry for an intrapartition destination, is when:

- You want to specify a particular filemode for the file
- You want to specify a CMS filename different from the transient data destination name
- You want to specify a filetype different from the default defined in EFHSETP or EFHPROF.

You define a transient data destination with an entry in a transient data destination table. Like all CICS/CMS tables, this is a CMS file, with a defined structure, and a filetype defined in your EFHPROF EXEC, as explained in "CICS/CMS tables" on page 66. You can therefore create and change it using your CMS editor.

The sample table in Figure 32 shows typical entries for intrapartition, extrapartition, and remote transient data destinations.

```
* CICS/CMS Sample TD Destination Table
*
* Define alternative file types and modes for
*    intrapartition destinations
*
IN01 I IN01     IDFILE   *
IN02 I IN02              Z
*
* Define extrapartition destinations for
*   input (EXTI) and output (EXTO)
*
EXTI E I
EXTO E O
*
* Define remote destinations
*
RDS1 R 0060
RDS2 R 0080 LDS2
*
* End of TD Destinations
```

Figure 32. Sample transient data destination table

Defining intrapartition destinations

The general form of definition for intrapartition destinations is:

1. The destination name, starting in column 1.
2. The letter I, signifying an intrapartition destination. This must be in column 6.
3. The filename of the file associated with the destination, starting in column 8.
4. The filetype, starting in column 17.
5. The filemode, starting in column 26.

The first two entries in Figure 32 show the most usual reasons for defining intrapartition destinations in the table: to use a filetype other than the default, or specify a particular filemode, for the CMS file associated with an intrapartition destination.

Using something other than the default filetype

The first entry in Figure 32 is:

```
IN01 I IN01     IDFILE   *
```

It shows you how to override the intrapartition destination filetype set up in the EFHSETP EXEC or your own EFHPROF EXEC, for a particular destination.

It associates the destination IN01 with a CMS file with the same filename, and a filetype of IDFILE. The * defines the filemode, telling CICS/CMS to search all disks in the defined CMS search order for the file. If you are reading from IN01, CICS/CMS will search only for a file of that filename and filetype. If it doesn't find one, it will give you an appropriate error message. If you are writing to IN01, CICS/CMS will first see if one already exists, and, if it doesn't, it will create a new one on your A-disk.

Using a specific filemode

The second entry in Figure 32 on page 74 is:

```
IN02 I IN02              Z
```

It defines the filemode (Z) for the destination IN02.

Usually, when you refer to an intrapartition destination in a CICS program, CICS/CMS looks for a CMS file of the same name as the destination, and with the filetype that is specified in the EFHSETP or EFHPROF EXEC. It searches the CMS disks, in your specified search order, until it finds such a file.

Suppose you have two files, both with the same filename and filetype, but on different disks (that is, with different filemodes). CICS/CMS will always use the one on the disk that is earlier in the search order, unless you tell it specifically to use the other one.

The entry above shows you how to do this. It associates the destination IN02 with a CMS file with the same filename, and the default filetype (specified by leaving the field blank). However, it sets the filemode to Z. CICS/CMS will search the Z-disk, ***and only the Z-disk***, for the file.

Defining extrapartition destinations

The form of definition for extrapartition data sets is:

1. Name (EXTI and EXTO in Figure 32), starting in column 1.
2. The letter E, defining the destination as extrapartition. This must be in column 6.
3. The letter I (for an input destination) or O (for an output destination). This must be in column 8.

You don't define the format of extrapartition data sets in the destination table. You define it in the CMS `FILEDEF` commands that you need for all extrapartition data sets (see "Defining extrapartition queues to CMS" on page 58).

Defining remote destinations

The form of definition for a remote destination is:

1. Name (RDS1 and RDS2 in Figure 32), starting in column 1.
2. The letter R, indicating a remote destination. This must be in column 6.
3. The maximum logical record length (60 and 80 bytes in Figure 32 on page 74), starting in column 8.

4. The name by which the destination is known on the remote CICS/VS system, starting in column 13. In the definitions in Figure 32 on page 74, RDS2 is known as LDS2 on the remote system. Since we have given no remote destination name for RDS1, CICS/CMS will assume that RDS1 is the name by which it is known on the remote system.

Temporary storage tables

Like all other CICS/CMS tables, a temporary storage table is a CMS file with a defined filetype, as explained in "CICS/CMS tables" on page 66. You can create and change this file using your editor.

The only time that you need to put an entry for a temporary storage queue in the table is when that queue is on your remote CICS/VS system. You don't need table entries for any local temporary storage queues.

The form of the table entries can be seen in Figure 33.

```
* ************************************************************************
* Example of a CICS/CMS temporary storage table
* ************************************************************************
* Define remote temporary storage ID
* Local name (1-8 characters) followed by remote name (1-8 characters)
* Both must be the same length (CICS requirement)
RLOCAL3  RREMID3
LOCTSQ   REMQID
```

Figure 33. Sample temporary storage table

The general form of a temporary storage table entry is:

1. The name by which your temporary storage queue is known to your CICS/CMS system (RLOCAL3 and LOCTSQ in the example above). This name can be between 1 and 8 characters long, and must start in column 1.

2. The name by which your temporary storage queue is known on the remote CICS/VS system (RREMID3 and REMQID in the example above). This name must start in column 10.

   If you include this, it must be the same length as the local name. If the name you specify is less than 8 characters long, you can use the name as a prefix. For example, the second definition in the sample temporary storage table defines the local queue name as LOCTSQ. If an application referred to a temporary storage queue called LOCTSQ1, the queue that it would use would actually be the remote queue, REMQID1.

   If you leave the remote name out, CICS/CMS assumes that the queue is known to the remote system by the same name as it is known to the local system.

File tables

You can use files with CICS/CMS without doing anything to a table. The only circumstances where you have to change, or add to, a CICS/CMS file table are:

- When you want to use a remote file, through the remote server
- When an application creates a new file
- When you want to increase the maximum record length of an existing file that contains variable-length records.

To do this, you have to set up a CICS/CMS table containing definitions of the data files you need. This table itself is a CMS file with a defined filename, as explained in "CICS/CMS tables" on page 66.

You can see what file table entries look like from the sample table in Figure 34.

```
* A CICS/CMS sample file table
*
* A local keyed file, to be created on disk A1. It contains
* fixed-length records, 80 bytes long. It has 1-byte keys,
* starting in the 10th byte (the offset is 9)
*
AKEY1    L K F 80    A1 9     1
*
* A similar file, with 4-byte keys
*
AKEY2    L K F 80    A1 9     4
*
* Another, but with variable records, and 10-byte keys
* which start in the first byte (the offset is 0)
*
AKEY3    L K V 80    A1 0     10
*
* An ESDS file containing variable-length records
*
ESDSFIL  L E V 72    A1
*
* A pair of RRDS files, both containing fixed-length records,
* 80 bytes long. The first will have a filemode of A4,
* the second a filemode of A1
*
RRDS1    L R F 80    A4
RRDS2    L R F 80    A1
*
* The files for the primer application
*
ACCTFIL  L K F 383   A1 0     5
ACCTIX   L K F 63    A1 0     17
*
* A remote ESDS file, known remotely as R1REM
*
R1FILE   R E F 75    R1REM
*
* A remote RRDS file, known remotely as R2XYZ
*
R2       R R F 50    R2XYZ
*
* A remote KSDS file, known remotely as R3MKW, with 2-byte keys
*
R3       R K F 80    R3MKW    2
```

Figure 34. Sample file definition table

The fields in a file table entry give:

1. The filename by which the file is known to CICS/CMS. This field must start in column 1.

2. Whether the file is local (L) or remote (R). This field must be in column 10.

3. The structure of the file: KSDS (K), ESDS (E), or RRDS (R). This field must be in column 12.

4. The format of the records: variable-length (V) or fixed-length (F). This field must be in column 14. All RRDS files must contain fixed-length records.

5. The maximum record size, starting in column 16.

6. For ***local*** files, the filemode of the file, that is, the CMS disk on which it's stored.

   For ***remote*** files, the name by which the file is known on the remote CICS/VS system.

   In both cases, the field must start in column 22.

7. The offset of the key within each record. This field must start in column 25. It applies to local keyed files only.

8. The length of the key, starting in column 31.

PSB directories

PSB directory entries define DL/I program specification blocks. Since you cannot use DL/I data bases locally, you must define every PSB you want to use in a PSB directory. The directory is a CMS file with a defined filetype, as explained in "CICS/CMS tables" on page 66.

Each entry in a PSB directory has two fields:

1. The name by which CICS/CMS will refer to the PSB locally

2. The remote PSB name, starting in column 10.

The example in Figure 35 shows a PSB directory containing three entries.

```
* CICS/CMS Sample PSB Directory
*
* Define three PSBs, LOC1 LOC2 and LOC3, known as
*    REM1 REM2 and REM3 on the remote system
*
LOC1     REM1
LOC2     REM2
LOC3     REM3
*
* End of PSB Directory
```

Figure 35. Sample PSB directory

What's next?

In the next part of the book, we tell you how to develop applications using CICS/CMS: from creating programs and maps to shipping your tested applications to a CICS/VS system.

Part three—Application development

This Part of the book describes:

Each step of using CICS/CMS to develop an application, in detail. The topics covered are:

- Creating source programs and maps
- Assembling maps, and translating and compiling programs
- Preparing your test environment
- Testing applications
- Testing printer applications
- Correcting and retesting applications
- Transferring tested applications to a CICS/VS system.

There is also a chapter on shortcuts: ways in which more experienced CICS/CMS users can speed up their application development.

Chapter 6. Creating and editing programs and maps

You can create and edit source programs and BMS maps within CMS. This means that you don't have to start CICS/CMS to start developing an application.

You store source programs and maps as CMS files on one of your CMS disks, usually the A-disk. It doesn't matter which of the CMS-supported editors you use to create them. In this guide, we always refer to XEDIT, because that's the standard editor that both VM/SP and VM/PC support.

Source program files must be unpacked, and must have a record length of 80 (if they contain fixed-length records), or a maximum record length of 130 (if they contain variable-length records). When CICS/CMS translates and compiles your source files, it checks to ensure that they obey these rules and, if they don't, displays a message and stops the operation.

CICS/CMS distinguishes between the different types of source program files by their filetype, as follows:

Source Language	Filetype
COBOL	Anything beginning with the letters COB, other than COBOL2
COBOL II	Anything beginning with the letters COBOL2, or anything beginning with the letters COB, if you include COBOL2 as a translator option
PL/I	Anything beginning with the letters PLI
H Assembler	Anything beginning with the letters AS
F Assembler	ASSEMBLE

If you are using the F Assembler, your BMS source maps must have a filetype of ASSEMBLE. If you are using the H Assembler, BMS source maps can have any filetype.

Note: VM/PC only supports the H Assembler. By default, CICS/CMS under VM/SP uses the F Assembler. This doesn't, however, mean that you can't use the H Assembler on VM/SP. If that is the Assembler you normally use, your system administrator can set it up with a few small changes to some of the CICS/CMS EXECs, as explained in "Customizing CICS/CMS" on page 211.

CICS/CMS support for high-level languages

CICS/CMS supports programs written in the high-level languages of COBOL, COBOL II, and PL/I. In general, the CMS guides for those languages give you the information you need to use them with CICS/CMS. The relevant guides are:

COBOL *CMS User's Guide for COBOL*

COBOL II *VS COBOL II Application Programming: Supplement for CMS Users*

PL/I *OS/PL/I Optimizing Compiler: CMS User's Guide*

For general guidance on using high-level languages with CICS/VS, see the *CICS/VS Application Programmer's Reference Manual.* The advice there applies to using the languages with CICS/CMS, unless we say something different in this book.

The rest of this section on high-level languages gives specific rules governing your use of these languages with CICS/CMS, which you won't find in any of the publications mentioned above. It also gives guidelines on how to use the languages efficiently with CICS/CMS.

General information on using high-level languages in CMS

The information here applies generally to using high-level languages in CMS; it's not specific to CICS/CMS.

Auxiliary directory errors

If you're compiling a program locally on a PC, and you get the message:

```
ERROR SETTING AUXILIARY DIRECTORY
```

you've probably accessed the compiler on a disk of the wrong mode.

You must access your PC compiler disk using the same CMS mode as the host system disk containing that compiler. For example, if the COBOL compiler is on the P-disk on the host system, you must access your PC COBOL compiler disk as the P-disk.

Similarly, on VM/SP, you must access your compiler disks using the same mode as the one on which the compilers were generated. If you don't, you'll get the CMS message:

```
ERROR BUILDING AUXILIARY DIRECTORY
```

and the CICS/CMS messages EFH9156I and EFH9147S.

Using a mode 2 disk for the PL/I compiler

If you are using PL/I, you must access the compiler disk as mode 2. For example, if your PL/I compiler and its associated libraries are on disk 196, with a mode of G, you must always access it using:

```
ACCESS 196 G/G * * G2
```

The reason for this is that the CMS loader expects certain PL/I libraries, particularly IBMBPGDA and IBMBSTVA, to be part of PLILIB TXTLIB, rather than individual TEXT type files. By restricting your access to PLILIB TXTLIB G2, you "hide" the individual TEXT files.

If you don't access the compiler disk as a mode 2 disk, three things can happen:

- The directory in your virtual machine might be too large, using up all your virtual storage.
- The compiler will not use the auxiliary directory, which might affect run-time performance.
- Some programs might not run.

General information on using high-level languages with CICS/CMS

The information here applies to using all high-level languages with CICS/CMS.

For each language you use, you must execute a `GLOBAL TXTLIB` command before trying to execute any program. You can put the command in your CMS PROFILE EXEC (to affect a whole CMS session), or in your CICS/CMS EFHPROF EXEC (to affect just your CICS/CMS sessions).

The `GLOBAL TXTLIB` command needed for each language is:

COBOL `GLOBAL TXTLIB COBLIBVS EFHXLIB`

COBOL II `GLOBAL TXTLIB VSC29TXT VSC2LTXT EFHXLIB`

PL/I `GLOBAL TXTLIB PLILIB EFHXLIB`

EFHXLIB is the CICS/CMS library, and is therefore needed in all `GLOBAL TXTLIB` commands.

If you use more than one language, you can use a single `GLOBAL TXTLIB` command for all the libraries you need. For example, for PL/I and OS/VS COBOL, you would use:

```
GLOBAL TXTLIB PLILIB COBLIBVS EFHXLIB
```

OS/VS COBOL information for CICS/CMS

Default COBOL compiler options

Whatever your OS/VS COBOL system defaults, the CICS/CMS EXEC that compiles CICS programs (EFHTC) applies the following defaults:

```
BAT,APO,LIB,NOTRU,NOCOU,NOEND,NOFLO,NODYN
NOSTA,NOSYM,NOTEST,NORES,OSDECK
```

You can override these options when you use EFHTC as a command, as described in "Translating and compiling programs" on page 144.

Sequencing COBOL programs

You'll find debugging easier if you add line numbers to (sequence) your COBOL programs. The line numbers will appear in the following types of messages:

- Translator
- Compiler
- CICS/CMS error handler
- EDF panel.

You can then relate the messages to your source program without referring to translator or compiler listing files.

To sequence a COBOL program, you put line numbers in columns 1 to 6 of the source program, using appropriate editor options.

PL/I information for CICS/CMS

The following guidelines apply if you write your CICS applications in PL/I.

Default PL/I compiler options

Whatever your PL/I system defaults, the CICS/CMS EXEC that compiles CICS programs (EFHTC) applies the following defaults:

```
INCLUDE,NUMBER,GONUMBER,SEQUENCE(73,80),OSDECK
```

You can override these options when you use EFHTC as a command, as described in "Translating and compiling programs" on page 144.

Sequencing PL/I programs

You'll find debugging easier if you add line numbers to (sequence) your PL/I programs. The line numbers will appear in the following types of messages:

- Translator
- Compiler
- CICS/CMS error handler

- PL/I library run-time
- EDF panel.

You can then relate the messages to your source program without referring to translator or compiler listing files.

To sequence a PL/I program, do the following:

1. Use 80-byte fixed-length records for the PL/I source file.
2. Put sequence numbers in columns 73 to 80 of the source program using appropriate editor options.
3. Use the `SEQUENCE(73,80)` translator option. This is a default in the IBM-supplied system.
4. Use the `NUMBER`, `GONUMBER`, and `SEQUENCE(73,80)` compiler options. These are defaults in the IBM-supplied system.

SYSPRINT output

CICS/CMS doesn't use the CICS transient data destinations, CPLI and CPLD. Instead, it treats SYSPRINT output in the same way as CMS does. It sends the output to the "file" defined for SYSPRINT in a `FILEDEF` command. This "file" may be a CMS file, or the system printer, or even your screen. For example, if you include the command:

```
FILEDEF SYSPRINT TERM
```

in your EFHPROF EXEC, CICS/CMS will display all SYSPRINT output on your terminal screen.

If you don't use SYSPRINT at all, you can improve performance by changing the default PL/I load library from EFHPLIXX to EFHPLIYY (the version without SYSPRINT support).

For example, you could put the following statement in your EFHPROF EXEC:

```
SETPARM 'PLILOAD EFHPLIYY'
```

Handling program checks

CICS/CMS has its own way of reporting program checks. If you set the SPIE option to YES in EFHPROF or EFHSETP, CICS/CMS uses its own *specify program interruption exit* (SPIE) routine when there's a program check. It constructs an error panel which gives you detailed information on the state of things at the time of the program check, to help you find the problem that caused it. This is fully explained in "Program checks" on page 194.

If you want CICS/CMS to report program checks in this way, don't enable PL/I SPIE or STAE. They will override the CICS/CMS SPIE option. If you don't declare PLIXOPT in your programs, you will get NOSPIE and NOSTAE by default. If you do declare PLIXOPT, you'll need to include NOSPIE and NOSTAE among the options.

If you execute a program with PL/I SPIE and STAE in effect, you'll find it more difficult to deal with program checks. If you get a program check in your

application, or in a PL/I library, you will get a PL/I message in SYSPRINT. CICS/CMS will then pass control to any ON unit you've set up, just as if CICS were not there. If you get a program check within CICS/CMS, register 12 will not point to the PL/I task control area (TCA), and you won't be able to get any useful information.

PLIDUMP and REPORT output

CICS/CMS treats all PLIDUMP and REPORT output in the way defined for PL/I under CMS, rather than the way defined for PL/I under CICS/VS. It doesn't use the transient data destination, CPLI, but writes all PLIDUMP and REPORT output to the CMS file, PLIDUMP. It ignores any CICS PLIDUMP options.

Use of storage

In CICS/VS, PL/I programs obtain their initial storage allocation (ISA), and any increments to the ISA, using CICS commands of the form:

```
EXEC CICS GETMAIN LENGTH ...
```

These storage allocations are restricted to 64K bytes or less.

CICS/CMS does not have this restriction. For example, under CICS/CMS, a PL/I program could ask for an ISA of 100K bytes, or allocate an AUTOMATIC array of 200K bytes.

You need to remember that, if you take advantage of this leniency, you might develop and test PL/I programs that run successfully on CICS/CMS, but that fail with storage violations on CICS/VS.

CICS abends

CICS abends that PL/I issues under CICS/VS (APLC, APLG, and so on) cannot occur in CICS/CMS.

COBOL II information for CICS/CMS

The following guidelines apply if you write your CICS applications in COBOL II.

Default COBOL II compiler options

Whatever your COBOL II system defaults, the CICS/CMS EXEC that compiles CICS programs (EFHTC) applies the following defaults:

```
APOST,LIB,NOTRUNC,SIZE MAX
```

You can override these options when you use EFHTC as a command, as described in "Translating and compiling programs" on page 144.

SYSPRINT output and DEBUG input/output

CICS/CMS uses the same temporary storage queues for SYSPRINT output and DEBUG input/output as CICS/VS. You can therefore use CECI and CEBR to read, write, and check these queues, as explained in the *CICS/VS Application Programmer's Reference Manual.*

Improving performance

CICS/CMS supplies a module, EFHCO2XX, as a member of its EFHXLIB TXTLIB. This module defines a set of library routines that CICS/CMS loads each time you run a COBOL II program. Any modules that are not a part of EFHCO2XX are loaded dynamically during the program's execution using the command:

```
EXEC CICS LOAD ...
```

If your COBOL II programs always use particular IGZ modules that are not included in the default EFHCO2XX module, you can decrease the number of EXEC CICS LOAD commands needed, and thus improve performance, by adding them to that module.

Use the following procedure to set up your own version of the EFHCO2XX module:

1. Use CICS/CMS trace output to find out which IGZ modules your applications always (or very frequently) load.

2. Create your own version of EFHCO2XX on your A-disk. This must have a filetype of ASSEMBLE, and we strongly suggest you give it a name other than EFHCO2XX, to avoid confusion. For example, you could call it MYC2LIB ASSEMBLE.

   The file consists of a CSECT statement, followed by a series of EXTRN statements, naming IGZ modules to be loaded. For example, it might look like this:

   ```
   MYC2LIB CSECT
           EXTRN IGZESNP
           EXTRN IGZEDBW
           EXTRN IGZCDIF
                 •
                 •
                 •
           END
   ```

   You should include all the modules in the default EFHCO2XX file, as well as the modules you want to add to the default set. At the end of this description, we list all the modules in the default EFHCO2XX.

3. Assemble your file to produce the TEXT file that you'll need when running CICS/CMS.

4. Add a SETPARM statement to your EFHPROF EXEC to ensure that CICS/CMS uses your version of the COBOL II library routine file instead of EFHCO2XX. For example, if your file is called MYC2LIB TEXT, you need the statement:

   ```
   SETPARM 'CO2LOAD MYC2LIB'
   ```

Modules in the default EFHCO2XX file

```
IGZESNP     IGZEDBW     IGZCDIF     IGZCLDR
IGZELDL     IGZESPM     IGZCPAC     IGZCPCC
IGZEINI     IGZETID     IGZETSU     IGZETCL
IGZEABX     IGZETUN     IGZEMSG     IGZCVLD
IGZCMST     IGZCMSF     IGZEABN     IGZESAT
IGZERID     IGZCRSU     IGZCPRS     IGZCLLM
IGZETRM     IGZCRCL     IGZCPRC     IGZTRCL
IGZCFDP
```

CICS/CMS support for the CICS/VS API

Appendix B, "Differences Between CICS/CMS and CICS/VS," gives a general account of features of CICS/VS that are either not supported by CICS/CMS, or supported in a different way. The *CICS/CMS Application Programmer's Reference Summary* provides a complete account of the application programming interface (API) supported. It tells you what EXEC CICS commands you can use, and how you can use them. There are, however, a few important points that are worth stating here.

One of the biggest differences between CICS/VS and CICS/CMS is that CICS/CMS does not support applications written in macros; it supports only the command-level interface.

Apart from this, the features of CICS/VS that CICS/CMS supports either differently, or not at all, fall into four general categories:

1. Features related to devices that are neither supported nor simulated by CICS/CMS. An example of this is the `EXEC CICS ISSUE` command.

2. Features related to SNA sessions, LU 6.2 devices, and other similar facilities. An example of this is the `EXEC CICS CONNECT` command.

3. Features that are inappropriate to the single-user environment of CICS/CMS. BMS routing is an example.

4. Features that are inappropriate to the interactive nature of CICS/CMS, and for which CICS/CMS provides something similar, but more appropriate. Dump facilities are a good example.

If you include any of these features in your source programs, they will be translated and compiled correctly (if you've coded them correctly). But what happens when you execute the program?

Usually, CICS/CMS ignores the unsupported feature, and passes on to the next program statement. If you're running the program using the execution diagnostic facility (EDF), you will see that the response code is NORMAL.

Some features are diagnosed by the CICS/CMS error handler, which puts out an information or warning display. The `POST` and `DELAY` commands are handled in this way. The error handler display gives you the option of ignoring the message and carrying on, or stopping the program.

The *CICS/CMS Application Programmer's Reference Summary* tells you exactly how CICS/CMS handles all the CICS/VS API features it doesn't support.

CICS/CMS provides EDF and CECI to help you check the syntax of your EXEC CICS commands. You can also use EDF to test alternative execution paths, even those that contain unsupported, or partially supported, features.

In general terms, if your programs use CICS/VS features that CICS/CMS supports fully, you can be confident that they will run on a CICS/VS system, with the same results as on CICS/CMS. Programs that use CICS/VS features other than these will probably run on a CICS/VS system, but will certainly need further testing on a test CICS/VS system for assurance.

Finally, remember that CICS/CMS is based on CICS/OS/VS Version 1 Release 7. If you are developing applications that will run on an earlier release of CICS/OS/VS than that, or on a CICS/DOS/VS system, you must ensure that you use only EXEC CICS commands that are part of that system.

Naming your application files

It's up to you how you name the files that make up your applications. Indeed, your installation may impose standards that restrict your choice in naming application files.

What we describe in this section is simply a suggestion. It's a way of naming application objects that lets you see all the files and resources associated with the application, without seeing unrelated objects. It does so by exploiting the relative freedom that CICS/CMS gives you in naming application objects and resources.

Suppose you are developing an application to manage a small library index. If you always use the letters LIB as the start of each object name, you'll be able to list them together, without listing any other objects.

You can do this outside a CICS/CMS session, using a CMS `FILELIST` command with a filename parameter of `LIB*`.

Once you've started CICS/CMS, option 1 on panel EFH1 gives you a list of your application objects (or files). It's really just the result of a CMS `FILELIST` command, and could just as well list files that had nothing to do with CICS/CMS. To ensure that you list only those files applicable to your current application development, you have to use the `Criteria` line in panel EFH1. The kind of display you might get from CICS/CMS is shown in Figure 36.

```
 EFH11              FILELIST: LIB* * A1                 Line 1 of 12


Cmd Filename Filetype Fm Format Lrecl Records Blocks   Date     Time
    LIB00    COBOL    A1 F          80      94      4  3/01/86  9:28:00
    LIB01    COBOL    A1 F          80      88      4  3/06/86  8:42:00
    LIB02    COBOL    A1 F          80       3      1  3/26/86  1:52:52
    LIBINIT  ASSEMBLE A1 F          80      31      2  3/07/86 15:57:24
    LIBUPD   ASSEMBLE A1 F          80      12      1  3/07/86 12:21:00
    LIBERR   ASSEMBLE A1 F          80      57      3  3/08/86  9:54:00
    LIBFIL   EFHVDATA A1 F          80     148      6  3/27/86  9:00:00
    LIBFIL   EFHVINDX A1 F         512      11      5  3/27/86  8:40:00
    LIB      EFHTPROG A1 F          80       1      1  3/30/86  9:39:37




PF1=Help     2=Refresh 3=End       4=All this name 5=Translate/Compile 6=Edit
PF7=Backward 8=Forward 9=Install  10=MAP-TEXT&ADS 11=Execute          12=
===>
```

Figure 36. Sample file list of application objects

As you can see, the "LIB" application consists of three COBOL source programs, three BMS maps, a CICS/CMS keyed pseudo-VSAM file (LIBFIL), and a program table (if EFHTPROG is the defined filetype for program tables).

Chapter 7. Preparing your application for testing

This chapter tells you how to use options on the CICS/CMS EFH11 panel to convert your source maps and programs into assembled maps and translated and compiled programs, ready for testing.

However, you don't have to use panel EFH11, or even have CICS/CMS running to do any of this. Chapter 12, "Shortcuts for experienced CICS/CMS users" on page 143, tells you how to assemble, translate, and compile without using CICS/CMS panels.

The CICS/CMS temporary disk (Z-disk)

CICS/CMS uses the CMS Z-disk as a temporary disk to hold listing output from assemblies, translations, and compilations. You can set up the Z-disk in three ways:

1. On a PC, you can define your Z-disk before running CICS/CMS, using the VM/PC configurator, as explained in "The steps in organizing PC storage" on page 38.

2. If you're using VM/SP, or you haven't predefined your Z-disk on VM/PC, CICS/CMS will create a Z-disk for you the first time you use one of the CICS/CMS EXECs to assemble or translate and compile in any CMS session. It does this using its EFHTSPAC EXEC.

3. You can run the EFHTSPAC EXEC yourself, as a command, from the `CP/CMS Command` line on panel EFH12. EFHTSPAC looks to see if you've got a Z-disk currently accessed. If you have, it does nothing; if you haven't, it defines one.

   If you're using VM/SP, you may like to consider including an `EXEC EFHTSPAC` command in your EFHPROF EXEC. This will ensure that, if necessary, CICS/CMS will set up a Z-disk whenever you start CICS/CMS.

You can use the Z-disk for other CICS/CMS files, such as trace and error logs, by specifying Z as the filemode in the appropriate EFHSETP or EFHPROF parameters.

You must always remember, however, that, unless you are using a predefined Z-disk on a PC, the Z-disk is a ***temporary*** disk, and you may have to take steps to safeguard its contents. The life of a Z-disk is as defined below:

- The Z-disk survives between successive CICS/CMS sessions during a single ***CMS*** session.

- If you re-IPL CMS during a terminal session, the Z-disk will still be there, but you'll have to reestablish access to it, using the CMS command:

  ```
  ACCESS 199 Z
  ```

  If you don't reestablish access to the Z-disk, and you run EFHTSPAC, you'll get a completely new Z-disk, and lose the contents of the previous one.

- If you log off, you will lose the Z-disk. If you want to keep any of the files on a Z-disk between terminal sessions, you should copy them from the Z-disk to one of your permanent CMS disks before logging off.

Assembling maps

CICS/CMS provides an EXEC (EFHMAPCR) that creates both physical maps and symbolic description maps (DSECTs) from your BMS source files. The EXEC is associated with a PF key on panel EFH11, so you can assemble each source map with a single keystroke.

You can use EFHMAPCR in two ways:

1. Through the CICS/CMS application objects panel, EFH11, or;

2. By executing the EXEC as a command. We describe this method in detail in "Assembling BMS maps" on page 143.

To assemble a map from panel EFH11, using PF10, you first have to get CICS/CMS running by entering:

```
CICSCMS
```

When it starts, CICS/CMS first displays panel EFH1. If you select option 1 on this panel, CICS/CMS displays panel EFH11, looking something like the one in Figure 36 on page 91.

On this panel, PF10 is set to `MAP-TEXT&ADS`. It assembles your source map, and produces both the physical map and the DSECT. To assemble a map, therefore, you move the cursor alongside a file of filetype ASSEMBLE, containing BMS macros, and press PF10.

Output from assembling a map

File output

The main output from assembling a map is your physical map and/or DSECT. The physical map is written to a file that has the same name as your source file, and a filetype of TEXT. The DSECT is added to your private macro library, a CMS file with the filetype MACLIB. The name of this library depends on the language you are using (that is, what you have coded in the `LANG` option of your DFHMSD macro). The possible names are:

EFHCUSER	COBOL or COBOL II
EFHPUSER	PL/I
EFHAUSER	Assembler

If you don't already have a private macro library file when you assemble a map, CICS/CMS creates one, on your A-disk, and puts the DSECT for that map in it. From then on, every time you assemble a BMS map, the DSECT is added to the end of the library file. If you reassemble a map, CICS/CMS erases the existing DSECT, and adds the new one to the end of the library file.

You will also get listing files for each stage of the assembly. EFHMAPCR writes these to the Z-disk. The files have the same filename as your source map file, and the following filetypes:

ASMLIST1	for the DSECT listing
ASMLIST2	for the physical map listing

Screen output

Screen output from the assembly is as follows:

- If this is the first assembly, translation, or compilation in the current CICS/CMS session, and you haven't pre-defined a Z-disk, you'll get a message as CICS/CMS defines one.
- While EFHMAPCR assembles the map, it will display messages as it starts and ends each stage. You will also get messages which look like this:

```
EFH9183I DSECT assembly rc=n
EFH9187I MAP assembly rc=n
```

  where n is the CMS return code. If this is anything but 0, you will get error messages from EFHMAPCR indicating what the problem is.

You'll find an example of output from an assembly in Figure 3 on page 15.

Translating and compiling/assembling programs

Before you can test a program using CICS/CMS, you have to translate the EXEC CICS commands into equivalent statements in the language of your program, and then compile the program to produce an object program on a file of filetype TEXT. CICS/CMS provides an EXEC (EFHTC) to translate ***and*** compile your source programs.

EFHTC works in a similar way to EFHMAPCR, as described in "Assembling maps" on page 94:

- The only input required by the EXEC is your source program. You don't have to add job control language (JCL), submit batch jobs, or change systems between creating the program file and translating and compiling the program.
- You can use EFHTC either by pressing PF5 on panel EFH11, or by executing the EXEC as a command. We describe the latter in "Translating and compiling programs" on page 144.

  Note that, when you translate and compile using the PF key, you always get the same options on your translator and compiler commands. These default options are either those that IBM supplies with CICS/CMS, or those that your system administrator sets up when he or she tailors CICS/CMS to your

installation's requirements. The IBM-supplied defaults are described in the following sections:

COBOL "Default COBOL compiler options" on page 86

PL/I "Default PL/I compiler options" on page 86

COBOL II "Default COBOL II compiler options" on page 88.

If you want to use anything other than these options, you have to execute the EXEC as explained in "Translating and compiling programs" on page 144; you can't change options when you use the PF key.

If you look again at panel EFH11 (Figure 2 on page 13), you will see that PF5 is set to `Translate/Compile`. To translate and compile one of your source program files, move the cursor alongside the filename of the file and press PF5.

Note: CICS/CMS also provides an EXEC (EFHT) which translates programs without compiling them. For more information, see "Translating programs" on page 146.

Output from the translate phase

File output

When it translates a program, EFHTC produces two output files, as follows:

- A file containing the output from the translator, ready to be compiled. It has the filename TRANOUT, and the same filetype as your source file. If you are translating an assembler program, this file also contains any diagnostic messages.

  Note: The translator output file is filemode Z3. This means that, when EFHTC has finished reading it in the compilation phase, the file is erased.

- A file containing the translator listing. It has the same filename as your source file, and a filetype of TRANLIST. If you are translating a COBOL, COBOL II, or PL/I program, this file also contains the diagnostic messages. The records in the file are of variable length, with a maximum size of 121. Messages, however, never go beyond column 80, to make it easier for you to read them on your screen.

  If you want to review translator diagnostic messages from an assembler program, look in the listing file from the assembly: a file with the filetype ASMLIST.

Screen output

Screen output from the translate phase depends on whether the translation works, as follows:

- If the translation works, you will get two messages, as shown in the following output from a successful translation of the sample COBOL program, ACCT00.

```
EFH9166I Translating ACCT00 COBOL    A1  ...
EFH9167I No Translator messages.
```

EFHTC will then go on and compile or assemble the TRANOUT file.

- If there are errors in the translate phase, you will get messages alerting you to check the listing file, for example:

```
EFH9166I Translating ACCT00 COBOL    A1  ...
EFH9170E Errors detected - see ACCT00 TRANLIST Z1
```

  EFHTC will then stop; it won't continue with the compilation or assembly.

Output from the compilation phase

File output

If your program translates successfully, EFHTC compiles or assembles the TRANOUT file to produce your object program. It produces two files as a result:

- The compiler or assembler listing file is written on your Z-disk. It has the same filename as your source file, and a filetype of COBLIST, PLILIST, or ASMLIST, depending on your source language.
- The compiled or assembled program file is written on your A-disk. It has the same filename as your source file, and a filetype of TEXT. Note that whether the compilation is successful or not, EFHTC always erases the TRANOUT file.

Screen output

On your screen, you will get messages indicating whether the compilation has worked successfully, and error messages if it has not. Figure 5 on page 17 shows all the messages from a successful translation and compilation of the sample program, ACCT00.

Figure 37 on page 98 shows the kind of messages you can expect when things go wrong. We have "corrupted" the source of the sample program, ACCT01, so that it contains an illegal `PIC` definition.

```
EFH9166I Translating ACCT01 COBOL A2 ...
EFHCTRAN (NUM
EFH9167I No translator messages.


EFH9094I Invoking COBOL compiler to process translated O/P ....
COBOL TRANOUT (BATCH APOST LIB NOTRUNC


REL2.4 OS/VS COBOL IN PROGRESS
*STATISTICS*     SOURCE RECORDS =  1057     DATA DIVISION STATEMENTS =   669
 PROCEDURE DIVISION STATEMENTS =   269
*OPTIONS IN EFFECT*     SIZE =  131072  BUF =   12288  LINECNT = 57  SPACE1, FLA
GW,   SEQ,   SOURCE
*OPTIONS IN EFFECT*     NODMAP, NOPMAP, NOCLIST, NOSUPMAP, NOXREF, NOSXREF,   LO
AD, NODECK, APOST, NOTRUNC, NOFLOW
*OPTIONS IN EFFECT*       TERM, NONUM,   BATCH, NONAME, COMPILE=01, NOSTATE, NOR
ESIDENT, NODYNAM,   LIB, NOSYNTAX
*OPTIONS IN EFFECT*     NOOPTIMIZE, NOSYMDMP, NOTEST,   VERB,   ZWB, SYST, NOEND
JOB, NOLVL
*OPTIONS IN EFFECT*     NOLST , NOFDECK,NOCDECK, LCOL2,  L132,   DUMP ,   ADV ,
NOPRINT,
*OPTIONS IN EFFECT*     NOCOUNT, NOVBSUM, NOVBREF, LANGLVL(2)
 002 COMPILATION ERRORS.HIGHEST SEVERITY  C
 CARD   ERROR MESSAGE

 17     IKF2039I-C     PICTURE CONFIGURATION ILLEGAL. PICTURE CHANGED TO 9 UNLES
S USAGE IS 'DISPLAY-ST',
                       THEN L(6)BDZ9BDZ9.
 17     IKF2156I-W     PICTURE DOES NOT CONTAIN A SIGN. SIGN DROPPED FROM VALUE
CLAUSE LITERAL.

ERROR MESSAGES ISSUED.
EFH9148E Compilation errors.  COBOL return code= 8 ; see ACCT01 COBLIST Z




                                                              MORE...
```

Figure 37. Output from a failing compilation

Whenever it finds an error in the compilation, EFHTC starts by giving you statistics and telling you the compiler options that are in effect. It then gives the compiler error messages, as you would expect. Finally, it gives the CMS return code, and alerts you to look at the listing file on the temporary disk.

Translating programs without compilation/assembly

CICS/CMS provides an EXEC (EFHT) which will translate a program without compiling or assembling the resulting TRANOUT file. We describe this EXEC, and suggest when you might want to use it in "Translating programs" on page 146.

CICS/CMS macro libraries

When you compile a program, the compiler satisfies external references by searching libraries. Most CICS programs contain external references to CICS objects, such as copybooks and BMS DSECTs. CICS/CMS keeps these objects in three macro libraries, which are files with the filetype MACLIB. To satisfy external references in your program to CICS objects, EFHTC searches these libraries, in the order given below:

1. EFHxUSER

   This file resides on your A-disk. It is your private library, containing your own copybooks and the DSECTs created from your source BMS map assemblies.

2. EFHxSTD

   This file usually resides on the CICS/CMS system disk, It contains what CICS/CMS needs for the sample application. For example, the macro library, EFHCSTD, contains the copybooks for the sample application data files. Your system administrator may well use this library for CICS objects needed generally in your installation.

3. EFHxMAC

   This file usually resides on the CICS/CMS system disk. It contains CICS requirements, such as the BMS attribute constants (copybook DFHBMSCA), and the standard attention identifier list (DFHAID).

In each of the file names above, "x" will be replaced by a letter representing the language you are using, as follows:

C COBOL or COBOL II
P PL/I
A Assembler

Changing your user macro library

We saw in "Output from assembling a map" on page 94 how you make additions or changes to the BMS DSECTs in your EFHxUSER macro libraries. But how do you add, change, or delete copybooks from those files?

The CMS `MACLIB` command lets you manipulate macro library files. You can add members to, delete members from, or replace members within those files, as explained in the following sections.

Adding members to a macro library

First, you need to create a file containing the copybook statements that you want to add. This file must have a filetype of COPY. If you want to put more than one set of copybook statements in a single COPY file, you must start each set with a line saying:

```
*COPY xxxx
```

where xxxx is the unique identifier for that set.

You then use the MACLIB command with the ADD option to add the copybook statements to your CICS/CMS user library. For example, suppose you have a COPY file called FILES that you want to add to your COBOL user library. The structure of FILES COPY is:

```
*COPY FILE1
 Copybook statements for FILE1
*COPY FILE2
 Copybook statements for FILE2
```

If you issue the command:

```
MACLIB ADD EFHCUSER FILES
```

you will add two new members to EFHCUSER: FILE1 and FILE2. If you look at the macro library after the add, you'll find that the MACLIB command has added the new members after any existing members, and added their names to its directory, which is at the end of the library.

Replacing members in a macro library

The MACLIB replace (REP) function lets you add a new copybook set, with the same name as an existing copybook set, to a macro library.

If you look at the macro library after the replace, you'll see that both copybook sets are there, with the new one at the end of the file. However, the MACLIB command has added a directory entry for the new one, and removed the directory entry for the previous one. If you want to remove the previous version from the library, you have to use the MACLIB compress function, as described below.

For example, if you wanted to replace the FILE1 copybook set, you'd create a file, FILE1 COPY. You'd then use the command:

```
MACLIB REP EFHCUSER FILE1
```

Deleting members from a macro library

The MACLIB delete (DEL) function removes the directory entry for a copybook set from a macro library. It doesn't delete the copybook set itself until you use the compress function, as described below.

For example, if you wanted to delete the FILE2 copybook set, you'd use the command:

```
MACLIB DEL EFHCUSER FILE2
```

Compressing a macro library

As we've seen, when you replace or delete library members, the members themselves stay in the library. It is only the directory that changes. After you've been using a library for some time, you'll probably find that it's taking up more disk space than necessary, due to the number of unwanted entries.

The MACLIB compress (COMP) function physically removes all library members that have no directory entries.

For example, to compress EFHCUSER, you'd use:

```
MACLIB COMP EFHCUSER
```

Further information on macro libraries

If you want to find out more about the MACLIB command, look in the *VM/SP CMS User's Guide*.

Chapter 8. Preparing to test an application

As we showed in "Testing a CICS program" on page 18, CICS/CMS lets you test single programs as soon as you have translated and compiled them. Usually, of course, individual programs will form only a part of a conversational or pseudoconversational transaction. However, being able to test the elements of an application individually should help you get your applications working sooner.

Once you've got your complete application ready for testing, however, there are various things you may have to do to ensure that the application will work, and that it will work in conditions very similar to the conditions it will face when it's put into production.

Before you test your application therefore, you may need to do some of the following:

- Update CICS/CMS tables to define the resources your application needs
- Start the remote server, if your application uses any resources on your remote CICS/VS system
- Convert any local CMS files or remote VSAM files that your application needs to the format CICS/CMS requires
- Set up the resources needed if your application uses extrapartition transient data.
- Make temporary changes to your CICS/CMS environment to run this particular test.

The rest of this chapter tells you how to go about each of these preparation items.

Updating CICS/CMS tables

Chapter 5, "How CICS/CMS handles CICS resources" on page 51 gives you all the information you need on ***how*** to update tables. In this section, we'll just remind you about ***when*** you need to update tables.

Program table

You will need to change the program table for your application if:

- You need to associate programs with transaction IDs
- You want to test a program with a name different from the filename of the file that contains it
- The application programs are written in a language other than the default defined in EFHPROF or EFHSETP.

"Program tables" on page 70 tells you how to change the program table.

Transient data destination table

You will need to change the transient data destination table for your application if:

- You want to associate intrapartition data sets with CMS files that have a filetype different from that defined in the EFHSETP or EFHPROF EXEC (by default, CICSTDI).
- You want to use extrapartition data sets.

  Remember that you must also execute a CMS `FILEDEF` command for each extrapartition data set, associating it with a CMS file, as described in "Defining extrapartition queues to CMS" on page 58.
- You want to use remote transient data sets without adding the `SYSID` option to any EXEC CICS commands that refer to them.

"Transient data destination tables" on page 73 tells you how to change the transient data destination table.

Temporary storage table

You will need to change the temporary storage table for your application only if you want to use remote temporary storage queues without adding the `SYSID` option to any EXEC CICS commands that refer to them. "Temporary storage tables" on page 76 tells you how to change this table.

File table

You will need to change the file table for your application if:

- Your application is going to create any new CICS/CMS pseudo-VSAM files
- You need to increase the maximum record length of any existing pseudo-VSAM files containing variable length records
- Your application uses any remote VSAM files without including the `SYSID` option in any EXEC CICS commands that refer to them.

"File tables" on page 77 tells you how to change the file table.

Tables for remote resources

"Defining remote resources in CICS/CMS tables" on page 65 tells you how to prepare tables for all remote resources, particularly DL/I data bases. As explained there, changing a CICS/CMS table is only the first step in ensuring that your application can use remote resources. Before your application can access any remote resource, you need to start CEHS, the remote server transaction. CEHS runs on your remote CICS/VS system, and passes your application's remote resource requests to that system. This is described in the next section, "Starting and using the remote server."

Starting and using the remote server

To start the remote server, you access your remote CICS/VS system and enter the transaction ID CEHS. There is a difference between the transfer method for a PC/370 user and that for a host VM terminal user.

Transferring to a remote CICS/VS system from a PC

When you're using a PC under VM/PC, one of your keys acts as a "hot" key, letting you switch between two active sessions. One session is your local PC session; the other runs a terminal emulator. If your PC has a 3278/3279 emulation adapter, you can use the emulation session to connect to a remote CICS/VS system, and use the remote server.

Note: If your PC only has a 3277 emulation adapter, you can use the emulation session to connect to a remote CICS/VS system, but you can't run the remote server. You cannot therefore run CICS/CMS applications that use resources on the remote CICS/VS system.

The key that you use as the hot key depends on what kind of PC you are using. On the XT/370, it is the Scroll Lock key; on the AT/370 it is the Sys Req key.

Assuming that you already have VM/PC and CMS running, and that your PC is connected to the host VM system, your procedure for transferring to the remote CICS/VS system is as follows:

1. Press the hot key. You will get the VM/PC Session Selection menu shown in Figure 22 on page 43.

2. Select option 1. This starts up your emulation session. Your PC becomes a 3270-type terminal, connected to your VM host system. If you are currently logged-on to VM at the host, you must log off or disconnect, and clear the VM logo that will be displayed.

3. If the CICS/VS system you want to use is running as a guest of the VM host system to which you're connected, you can DIAL to that system.

   If the CICS/VS system is remote from your host VM system, you'll have to DIAL a VM pass-through userid. Your system administrator will set this userid up as part of the installation process. From the pass-through machine, you can connect to the remote CICS/VS system.

4. When you are connected to a CICS/VS system that is ready to receive transaction requests, you can start the remote server.

Note: Once you've gone through the procedure above, both your sessions are permanently available. Unless your remote system goes down, or you bring down VM/PC at your end, you will continue to be able to switch from one session to the other by pressing the hot key.

Now read "Starting the remote server" on page 107.

Transferring to a remote CICS/VS system from a terminal

Unlike VM/PC, VM/SP has no hot key to start a second session. CICS/CMS therefore has its own way of simulating the VM/PC facility.

First, you need to get CICS/CMS running (if you haven't already). Type:

```
CICSCMS
```

Select option 2 on panel EFH1 to display panel EFH12. Both EFH12, and the escape panel (EFH122), have PF8 defined as `VM Session`. Pressing this key automatically displays the logon panel for another VM session. What you do from there depends on where the CICS/VS system you want to use is, as follows:

- If the remote CICS/VS system is running on an MVS or VSE system that is a guest of the same VM system you are using, you can DIAL to it.
- If the remote CICS/VS system is running on an MVS or VSE system remote from your host VM system, you'll have to DIAL a VM pass-through userid. Your system administrator will set this userid up as part of the installation process. From the pass-through machine, you can connect to the remote CICS/VS system.

When you are connected to a CICS/VS system that is ready to receive transaction requests, you can start the remote server.

Having once established this link, you can switch between CICS/CMS and your remote CICS system at will. Pressing PF8 on the CICS/CMS panels EFH12 and EFH122 will switch you to the remote session. Pressing PA2 when you're connected to the remote session will display the remote session escape panel (EFH124) shown in Figure 38.

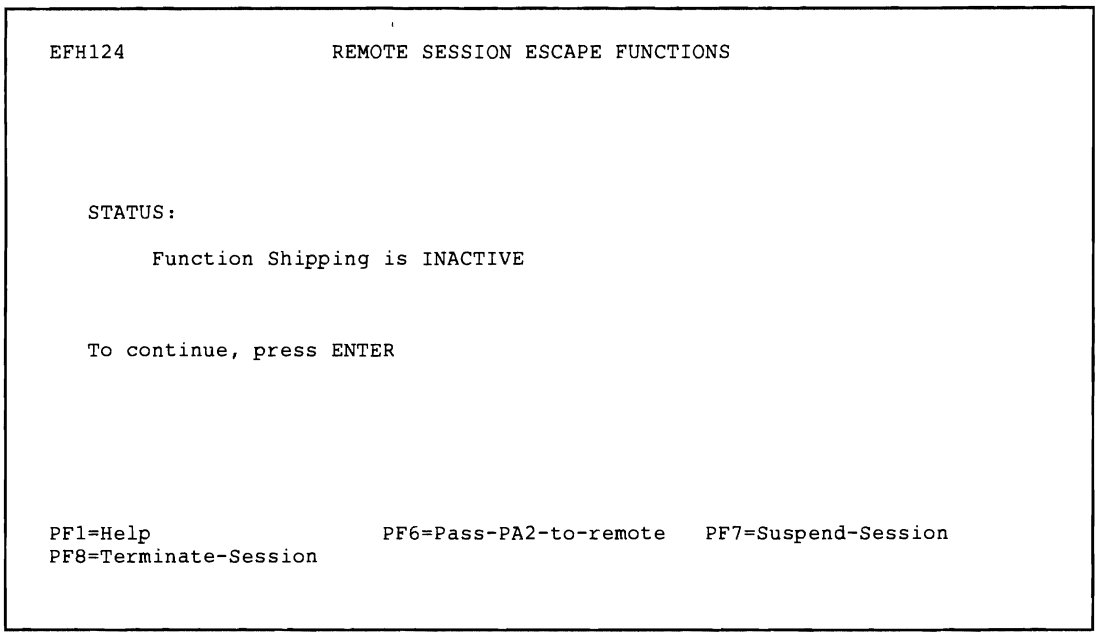


Figure 38. Remote session escape display

Pressing PF7 on this display will switch from your remote session to your CICS/CMS session.

Notes

1. You can stop and restart a CICS/CMS session without affecting your remote session. The remote session stops only when you:
   - Log off
   - Re-IPL CMS
   - Press PF8 on the remote session escape panel (EFH124).
2. When the CEHS transaction isn't running, you can still use PA2 on the remote system to display the remote system escape panel. However, you will see the `INACTIVE` message, as shown in Figure 38.

   If you escape to EFH124 while the remote server is doing something, you will see an appropriate `STATUS` message.

Starting the remote server

If you're using VM/SP, we advise you to stop your terminal receiving messages before you start the remote server. If you receive a message while using the remote server, you can lose your connection to the remote CICS/VS system. To stop your terminal receiving messages, type the CMS command:

```
SET MSG OFF
```

Whether you're using a PC or a terminal, the next step is the same. On your remote CICS/VS system, enter the transaction ID:

```
CEHS
```

The remote server transaction starts, and displays its logo, as shown in Figure 39 on page 108.

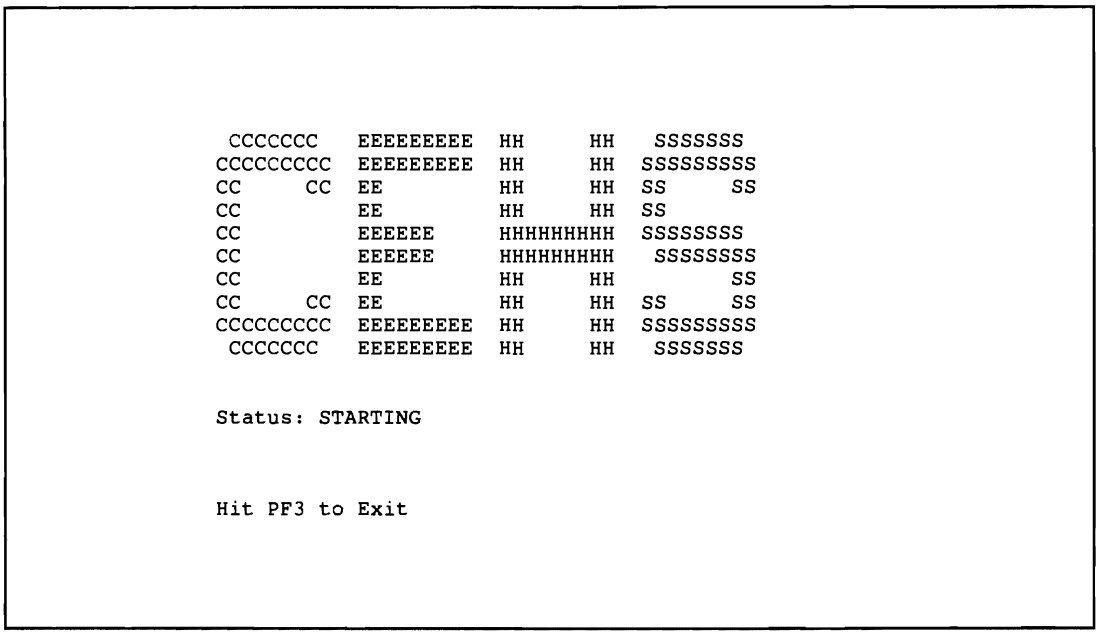


Figure 39. Remote server logo

Once the remote server has established communications with CICS/CMS, it displays its logo again, with the `Status` field saying:

```
Status: RUNNING
```

You can now return to your CICS/CMS session. If you're using a PC, press your hot key; if you're using a terminal, press PA2, followed by PF7.

Caution: Make sure that the CEHS logo is displaying a status of `RUNNING` before you return to CICS/CMS. If you return to CICS/CMS before the remote server is properly up and running, it won't accept requests from your applications.

You are now ready to start running applications that use remote resources.

You can verify, from your local CICS/CMS session, that the remote server is running, as follows:

- If you are using a PC, you just look at the display's status line. While the remote server is running, there will be a minus sign (−) to the left of the session identifier.
- If you are using a host-connected terminal, switch to the remote system, using PF8 on panel EFH12 or panel EFH122. Check that the CEHS logo panel still shows the status as `RUNNING`, and return to your CICS/CMS session with PA2/PF7.

Stopping the remote server

When you no longer need access to remote resources, you should stop the remote server to release the CICS/VS resources it has acquired.

Switch from CICS/CMS to your remote CICS/VS session (using the hot key on the PC, or PF8 on panel EFH12 or panel EFH122 on a terminal). Press PF3. The remote server will shut down, redisplaying its logo with the following status field:

```
Status: STOPPING
```

The screen will then clear, and you will get the message:

```
CEHS Terminated, clear screen and type next transaction identifier
```

You can then return to your local CICS/CMS session, using your hot key on a PC, or PA2/PF7 on a terminal.

If you have finished with the remote CICS/VS system, and you are using it from a terminal, you should drop the connection to the remote system. Press PA2 to display panel EFH124, then press PF8 to terminate the connection.

Recovery of remote resources

CICS/CMS doesn't support locally the CICS/VS recovery features, such as dynamic transaction backout (DTB) or syncpoints. However, when your applications are communicating with a remote CICS/VS system, using the remote server, they are using resources that may be shared by many others, and that therefore need protection.

Whenever one of your transactions that is using remote resources ends abnormally, CICS/CMS ships a DTB function request, ensuring that any changes that the application made to remote resources are removed. The same applies if something happens that stops the link between CICS/CMS and the remote CICS/VS system, such as CICS/CMS itself failing.

When a remote application ends normally, CICS/CMS ships a `SYNCPOINT` request on remote resources you have used, to ensure that changes are preserved. This only applies to "first level" transactions, not to any transactions that you run from the escape panel, EFH122.

Preparing local data files

As explained in "How CICS/CMS supports VSAM files" on page 60, CICS/CMS lets you supply data locally to your applications by supporting special files which simulate VSAM files:

- CMS files with a filetype of EFHVNONK simulate VSAM ESDS and RRDS files.
- Pairs of CMS files with the same filename and with filetypes of EFHVDATA and EFHVINDX simulate VSAM KSDS files.

The sections that follow tell you how to create both types of simulated VSAM file.

Converting local CMS files to CICS/CMS nonkeyed files

CICS/CMS treats any CMS file with the filetype EFHVNONK as a pseudo-VSAM nonkeyed file. You can therefore create nonkeyed files using CMS facilities.

For example, if you want to use the file APPLIC DATA A as a CICS/CMS nonkeyed file, you can execute a `FILELIST` command to list the file, then move the cursor alongside the filename, type:

```
COPYFILE / = EFHVNONK =
```

over the file ID, and press ENTER.

You can also create nonkeyed CICS/CMS files from VSAM files stored on your remote CICS/VS system. We describe this in "General file conversion" on page 113.

Converting local CMS files to CICS/CMS keyed files

The utility to convert a sequential CMS file into the CICS/CMS two-file keyed structure is a CICS/CMS program called EFHUCMS1. You can run it either by selecting option 4 on panel EFH1, as explained below, or by using `EFHUCMS1` as a command, as explained in "Converting CMS files to the CICS/CMS structure" on page 146.

Selecting option 4 on panel EFH1 displays panel EFH14, as shown in Figure 40.

```
EFH14                    CONVERT CMS FILE TO CICS/CMS FILE

   Specify for the CMS file to be converted:
     Filename          ===>
     Filetype          ===>
     Filemode          ===> *
     Key offset        ===> 0         Offset of key within record
     Key length        ===>           1-255 bytes

   Specify for the output files:
     Filename          ===>           Default - Input Filename
     Record length     ===>           Maximum ever to be written to data file
     Replace           ===> No        Yes,No

   Specify the records to be processed (default - entire file):
     Start record      ===> 1         Record at which processing starts
     Number            ===>           Processed sequentially from start record
     OR Key range: from ===>
                   to   ===>

   Trace required      ===> No        Yes,No

PF1=Help          PF3=End          PA2=Enter CMS Subset
```

Figure 40. Converting CMS files to keyed CICS/CMS files (panel EFH14)

The fields in the EFH14 panel are described below.

EFH14 fields for defining the input file

The first set of fields in panel EFH14 define the input file, as follows:

- The filename, filetype, and filemode of the CMS file to be converted. The only limitation on the size of this file is that it must not be so large that CICS/CMS cannot hold the complete index in virtual storage.
- The offset of the key within each record of the input file. This is counted from byte 0 (the default), so if the key starts in the fourth byte, for example, you must specify an offset of 3.
- The length of the key (between 1 and 255 bytes).

EFH14 fields for defining the output files

The second set of fields defines the output files, as follows:

- The filename of the output files. EFHUCMS1 will use this filename for both output files: the data file (which has a filetype of EFHVDATA), and the index file (which has a filetype of EFHVINDX). You can leave this out, and your two new files will have the same filename as the input file.
- The maximum length of record that will ever be written to the data part of the keyed file (filetype EFHVDATA). If you leave this out, or specify some value less than the maximum record length of the input file, your maximum record length will be that of the current longest record. In fact, as we describe in "File tables" on page 77, you can increase this value later, using an entry in the file table.

  If you specify this option, and the source file contains fixed-length records, the utility ignores it.
- A replace option, to specify what you want to happen if files already exist with the output filename, and a filetype of EFHVDATA or EFHVINDX.

  If you specify `yes` for replace, any existing files with the output filename and a filetype of EFHVDATA or EFHVINDX will be overwritten by the new files.

  If you specify `no` (the default), existing files will be protected, and the utility will stop with an appropriate message.

EFH14 fields for defining records to convert

The third set of fields defines the set of records from the input file that EFHUCMS1 will convert. You can either:

- Specify a number of records to convert, or
- Specify a range of keys. EFHUCMS1 will convert all records with keys lying within that range.

You use the first two fields in the set to define a number of records to convert, as follows:

- The number of the record at which to start the conversion. This gives you the chance to create pseudo-VSAM files that are subsets of existing CMS files.

If you leave the option alone, the conversion will start with the first record in the input file.

- The number of input records from the source file to be converted.

  If you leave this out, conversion will start with the "start" record defined above, and continue to the end of the file.

You use the last two fields in the set to define a key range. EFHUCMS1 looks at the `from` and the `to` key that you specify to see which is the lowest key in EBCDIC terms. It then converts all records with keys that lie between the low key and the high key, inclusive.

Both keys must be the same length. If they are shorter than the key length you specified, EFHUCMS1 treats them as generic keys.

EFH14 field for requesting trace

The last field on panel EFH14 specifies whether you want a trace of the conversion. The default is to provide no trace. The trace records each step of the conversion: checking the files, copying each record, closing the files on completion.

Note that, if you specify a range of keys to be processed, the keys that you entered will appear in the trace output in uppercase, no matter how you entered them, or how they appear in your input file.

EFHUCMS1 writes its trace output to your screen. If you want to keep a copy of the output, you will need to do the following:

1. Display panel EFH14, and specify all the options you need, including `yes` for `Trace required`.

2. **Before** pressing ENTER, press PA2 to enter the CMS subset, and type the CP command:

   ```
   SPOOL CONSOLE * START
   ```

   This tells CP to copy all your screen output to a file.

3. Type `RETURN` to return to panel EFH14, and then press ENTER to run the utility. Trace output will appear on your screen, and CP will write it to a file.

4. When the utility ends, press PA2 to enter the CMS subset, and type the CP command:

   ```
   SPOOL CONSOLE CLOSE STOP
   ```

   This closes the file CP has created, sends the file to your virtual reader, and stops CP from writing future screen output to a file.

5. Type:

   ```
   RDRLIST
   ```

   to list the files in your virtual reader. You'll see a file with a filename and filetype of `none`. To read this file to your A-disk, you need to enter the `RECEIVE` command with appropriate filename and filetype parameters. For

example, to read the file onto your A-disk with a filename of TRACUCMS, and a filetype of LOG, you type:

```
RECEIVE / TRACUCMS LOG
```

6. Press PF3 (to leave the reader list), then type `RETURN` to return to the panel EFH14 display.

Output from the EFHUCMS1 utility

The principal output from the utility is the CICS/CMS keyed file, consisting of a pair of files, one of type EFHVDATA the other of type EFHVINDX, as described in "The CICS/CMS keyed file format" on page 61. These files will be written to the same CMS disk as the source file, if you have read/write (R/W) access to that disk. If not, they will be written to your A-disk.

Even if you don't ask for trace, you will also get some screen output from the utility. If it is successful, you'll get a message confirming that it has created the new files. If anything goes wrong, you will get messages to help you find the problem.

General file conversion

The general file utility for converting and copying files is a CICS transaction (CCU2) within CICS/CMS. It has three functions, as follows:

1. Loading a VSAM file from your remote CICS/VS system, using the remote server; converting that file into a CICS/CMS keyed or non-keyed file; and saving the resulting file(s) on the CMS disk specified for that filename in a CICS/CMS file table.

2. Doing the opposite to the process above. Converting CICS/CMS files into VSAM files and sending the VSAM files to your remote CICS/VS system.

   Before you use this function of CCU2, you will not only have to ensure that the remote server is running, but also that you've prepared your remote CICS/VS system to receive the converted file(s). Any files you want to store on that system will need space set aside for them, and entries made in the system's file control table (FCT).

3. Creating CICS/CMS files that are subsets of existing CICS/CMS files. Because this does not involve the remote CICS/VS system, it's the one function of the CCU2 utility for which you don't need the remote server.

This section deals specifically with getting data files ready for testing an application. We will concentrate therefore on the first function, but we will also tell you how to use the utility for functions 2 and 3.

Note: Apart from its general use in copying subsets of CICS/CMS files, function 3 has a particular application in reorganizing CICS/CMS keyed files that have grown too large. We describe this separately, in "Using CCU2 to reorganize CICS/CMS keyed files" on page 116.

For functions 1 and 2, the utility has to load a file from, or copy a file to, your remote CICS/VS system. You therefore have to start the remote server on that system before you can use this utility for either of its first two functions. We won't

tell you how to do this here; it's described in "Starting and using the remote server" on page 105.

Once you've started the remote server, and returned to your CICS/CMS session, you can run CCU2. On panel EFH12, you enter `CCU2` (the utility transaction ID) on the `Transaction` line. This will display the panel shown in Figure 41.

```
CCU2              GENERAL FILE COPY UTILITY


  Specify for the file to be converted:
     Name             ===>
     Access type      ===>          Key(default), RBA or RRN.
     Key Length       ===>          1-255 bytes, if access type is key.

  Specify for the output files:
     Filename         ===>          Different to input file name.

  Specify the records to be processed (default- entire file)
     Start value      ===>          RBA or RRN value. Default = 0.
     Number           ===>          Processed sequentially from start record.
     OR
     Key range: from ===>
                to   ===>




PF1=Help          PF12=Quit         PA2=Escape
```

Figure 41. Converting remote VSAM files to CICS/CMS files (panel CCU2)

The fields in the CCU2 panel are described below.

CCU2 fields for defining the file to be converted

The first set of fields defines the input file to CCU2, as follows:

- The name of the file to be converted.

  When you're converting a remote VSAM file into a CICS/CMS file, you must define this file as remote in your current file table.

  For functions 2 and 3, this will be a local CICS/CMS file.

- The type of access used for the input file (key, RBA or RRN). If you omit this option, CCU2 assumes `key`.

  If the input file is keyed, and contains fixed-length records, they must be at least 16 bytes long. All RRDS files contain fixed-length records.

- The length of the key (if you've specified `key` as the type of access). This can be between 1 and 255.

CCU2 field for defining the output file

In the CCU2 field starting `Filename`, you must give the filename of the output file (or files if you've specified `key`). The output filename must ***not*** be the same as the name of the file to be converted. When you're converting files between CICS/CMS and VSAM format (functions 1 and 2), this might not be what you want. Normally, you'll probably want the file to have the same name on CICS/CMS as on CICS/VS. Otherwise, you'll have to change your application after testing it on CICS/CMS, before transferring it to CICS/VS.

You can get round this by defining both files in the CICS/CMS file table, giving the remote file a dummy local name. For example, suppose you want to convert a remote file called MYDATA into a local CICS/CMS file with the same filename. In your file table, you can include the following definitions:

```
DUMMY    R K V 80    MYDATA   5
MYDATA   L K V 80    A1 0     5
```

The first definition specifies that the remote file, MYDATA, is to be known by CICS/CMS locally as DUMMY. The second definition describes the local file, MYDATA. When you use CCU2 to create the local file MYDATA, you can then specify DUMMY as the input file, and MYDATA as the output file, thus getting round the filename restriction.

You can use the same definitions for converting a CICS/CMS file into VSAM format, and copying that file to your remote CICS/VS system. In that case, however, MYDATA will be the ***input*** file to CCU2, and DUMMY the ***output*** file.

For function 3, both the input and output files will be local CICS/CMS files.

CCU2 fields for defining records to be processed

The last set of fields defines the set of records from the input file that CCU2 will convert. You can either:

- Specify a number of records to convert, or
- Specify a range of keys. CCU2 will convert all records with keys lying within that range.

You use the first two fields in the set to define a number of records to convert, as follows:

- The number of the record in the input file at which to start the conversion. You can use this option only for ESDS or RRDS files. If you leave it out, and don't specify a key range, the conversion will start with the first record in the input file.
- The number of input records from the source file to be converted.

  If you leave this out, conversion will start with the "start" record defined above, and continue to the end of the file.

If you use the last two fields in the set to define a key range, CCU2 will convert all records with keys within the range defined by the `from` key (the low key), and the `to` key (the high key), inclusive. The records in the input file don't have to be in ascending order, but they will be in the output file. If you specify a `from` key that is higher, in EBCDIC terms, than the `to` key, CCU2 will stop without converting

any records. Both keys must be the same length, and you must enter them in the case in which they appear in the input file. If the keys you enter are shorter than the key length you specified, CCU2 treats them as generic keys.

Output from CCU2

The output from CCU2 depends on what you have requested. For functions 1 and 3, you will get either a single CICS/CMS file of filetype EFHVNONK, or the two files that make up a CICS/CMS keyed file, one of filetype EFHVDATA, the other of filetype EFHVINDX.

For function 2, you will get a single VSAM file, which CCU2 creates on your remote CICS/VS system, using the remote server.

You will also get a message telling you if the transaction has completed successfully, as follows:

```
Utility complete. xxxx records have been processed.
CLEAR screen and continue
```

To get back to the execution panel, EFH12, you clear the screen and enter:

```
CCMS QUIT
```

Using CCU2 to reorganize CICS/CMS keyed files

When you delete a record from a CICS/CMS keyed file, CICS/CMS doesn't remove the record from the data file (filetype EFHVDATA). Instead, it flags the record's entry in the index file (filetype EFHVINDX) to show that the record is no longer active. When you add a new record, CICS/CMS will overwrite a deleted record whenever possible. However, it can only do so if the new record is exactly the same length as the inactive record. If a file contains fixed-length records, this will always be true. But, if a file contains variable-length records, new records will often not be the same length as existing inactive records. CICS/CMS will therefore add new records to the file without replacing any existing records. As time goes on, the number of inactive records may reach a level that interferes with your testing in the following ways:

- Both the data and index files will occupy an unnecessarily large amount of disk storage.
- The index file is loaded into storage when you access the file. You may therefore be forced to increase your dynamic storage area (CICSDSA parameter) just to accommodate the file.
- It will take longer to retrieve data from the file, because CICS/CMS will have to read additional index entries unnecessarily.

To avoid this, we recommend that, when necessary, you use CCU2 to reorganize variable-length CICS/CMS keyed files. The sequence below describes the process step-by-step, using example filenames of OLDFILE (for the file to be reorganized) and NEWFILE (for the reorganized file that CCU2 creates).

1. Add an entry to your file table to define NEWFILE. Apart from the filename, all the other fields in the file table entry must match the OLDFILE's structure. See "File tables" on page 77 for a description of file table entries.

2. Run CCU2, by entering CCU2 in the Transaction line of panel EFH12 and pressing ENTER.

3. Specify OLDFILE in the Name field of panel CCU2.

4. Specify NEWFILE in the Filename field of panel CCU2.

5. Press ENTER to run CCU2. By leaving the Start value and Number fields blank, you tell CCU2 to start at the first record in OLDFILE and continue to the end of the file. CCU2 ignores all inactive records, and writes only the active records to NEWFILE.

6. Once you've ensured that CCU2 has completed successfully, and checked NEWFILE EFHVDATA and NEWFILE EFHVINDX, erase OLDFILE EFHVDATA and OLDFILE EFHVINDX. You can then use the CMS RENAME command to rename the NEWFILE files to OLDFILE EFHVDATA and OLDFILE EFHVINDX.

Safeguarding your data files

Because it is a single-user application development system, CICS/CMS does not recover local resources when an application fails. This does not apply to remote resources; any data your application accesses via the remote server will have the same protection as any other resources on your remote CICS/VS system.

If you are using local data files, therefore, you need to take steps yourself to ensure that, if you need to correct and rerun your application, you can return all data files to their original state.

The easiest way of protecting local data files is to keep master versions of the files, and run your application against copies of these files. You can create these copies using the CMS COPYFILE command. If you need to rerun your application, you can then erase the copies that the first run used, and create new copies from the masters.

The best way of doing this is probably to write a short EXEC that copies the master versions. It can be as simple as:

```
/*   REXX EXEC to reset ACCTFIL from master copy */
'COPYFILE ACCTFIL MASTDATA A = EFHVDATA  A(REP'
'COPYFILE ACCTFIL MASTINDX A = EFHVINDX  A(REP'
SAY 'ACCTFIL has been reset from master'
EXIT
```

The example above copies the EFHVDATA and EFHVINDX files for the sample application file ACCTFIL. We've given the original versions of the files the filetypes MASTDATA and MASTINDX. Every time we want to restore ACCTFIL to its original state, we simply run the EXEC.

Don't forget that, if your files are CICS/CMS keyed files, you will always need to copy both the EFHVDATA and EFHVINDX files from your masters, as shown in the example EXEC above.

Reading data from your virtual reader

There is an application of extrapartition data sets that you might find useful. It's best described by example.

Suppose you are developing an application that reads an extrapartition data set created by another programmer outside CICS. One way of ensuring that your application can read that extrapartition data set is to ask the other programmer to send the data set to you. You can then read it into a file on your A-disk, and define that file as an input extrapartition data set.

There may be occasions, however, when this is impractical. For example, if the data set is very large, you might not have room for it on your A-disk.

You can get round this by associating the extrapartition queue with your virtual reader. You can then use the file the other programmer has sent directly.

However, there are some special rules governing this, and you need to be aware of these before running any application that uses extrapartition queues in this way. The rules are:

- The FILEDEF command must take the form:

  ```
  FILEDEF xxxx READER (RECFM F LRECL 80
  ```

  where xxxx is your extrapartition queue name.

- The status of the reader file must be NOHOLD. You can set its status using the CP command:

  ```
  CHANGE RDR spoolid NOHOLD
  ```

  where spoolid is the VM spool ID of the reader file.

- Your reader class must match the class of the file you are trying to read. You can use the CP QUERY command to find the reader class.

- Your extrapartition queue file must be the first file in the reader. You can use the CP ORDER command to ensure that it is, as follows:

  ```
  ORDER RDR spoolid1 spoolid2...
  ```

  The spoolids identify the files in your reader, in the order in which they will appear in the reader. spoolid1 should therefore be your extrapartition queue file.

- CMS will remove the file from your reader after you have read the last item from the extrapartition queue. You can avoid this by executing a CP SPOOL command with the HOLD option before you run the program that reads the file.

- You should be careful if you set up the queue by creating a file with the CMS PUNCH command, and then transferring that file to your reader. By default, the PUNCH command adds a header record to the files it creates, which will confuse CICS/CMS when it tries to read the queue. If you want to create the file for your extrapartition queue in this way, you must always use the NOHEADER (or NOH) option in the PUNCH command.

Changing your CICS/CMS environment within a session

As explained in Chapter 4, "Setting up your CICS/CMS environment," the EFHSETP and EFHPROF EXECs define your CICS/CMS environment: the language you are using, your table names (if any), your TWA size, and so on. This environment (or profile) is set up every time you start CICS/CMS.

If you want to change the environment in any way, you will have to do so before you run the application that requires the change. For example, one feature of the environment is the programming language you are using. If EFHSETP defines your default language as COBOL, and you've written a program in PL/I, you will have to tell CICS/CMS not to treat it as a COBOL program before you run it. You can do this either by defining the program as PL/I in a CICS/CMS program table, or by changing the default programming language to PL/I for a single CICS/CMS session.

PF7 (`Set parameters`) on panel EFH12 gives you the chance to change your environment for a particular CICS/CMS session, without changing your EFHPROF EXEC or a program table. It displays your current values on the parameter definition panel (EFH121) shown in Figure 42.

```
EFH121                        PARAMETER DEFINITION

 Trace                      ===> NO           Yes,No
 EFHUSTG table entries      ===> 500          100-1000
 Trace table filemode       ===> Z            2 characters
 Trap program checks        ===> YES          Yes,No
 DEBUG program              ===>              1-8 characters
 Dynamic storage            ===> 160000       50000-4000000
 Language                   ===> COBOL        COBOL, COBOL2, ASSEMBLE or PLI
 TWA size                   ===> 1024         1024-32767
 SYSID for remote system    ===> REMT         4 characters

 Names of resource tables:       Filename Filetype Filemode
   Program/Transaction      ===> CICSCMS  EFHTPROG *
   File                     ===>          EFHTFILE *
   Transient Data           ===>          EFHTTD   *
   Temporary Storage        ===>          EFHTTS   *
   PSB Directory            ===>          EFHTPDIR *

                                Keyword   Value
 Additional Parameter       ===>

PF1=Help          PF3=End            PF12=Quit            PA2=CMS Subset
```

Figure 42. CICS/CMS parameter definition panel (EFH121)

The seventh line of the panel defines the language. If you wanted to test a PL/I program that you hadn't defined in a CICS/CMS program table, you would change `COBOL` to `PLI`.

You can change any of the parameters shown. Among other things, you can:

- Change the filenames of resource tables to use different tables for different applications.
- Turn on CICS/CMS trace for a particular session.
- Use a debugging program of your choice.

- Change any of the CICS/CMS parameters not shown specifically in the panel, using the `Additional Parameter` line. If you type just the keyword, and press ENTER, CICS/CMS will display the current value. If you want to clear the existing value, put a period (.) in the `Value` field.

  The keywords for all CICS/CMS parameters, and the values you can set, are described in Appendix D, "CICS/CMS parameters" on page 241. You must always be careful to ensure that you know exactly which values are valid for a particular keyword. CICS/CMS does not check the validity of values when you enter them in the `Additional Parameter` line. You'll only find out you've made a mistake later, when you try to use the feature for which you set the parameter.

You can display a similar panel (EFH1221) from the escape panel (EFH122) as shown in Figure 43.

```
EFH1221                        PARAMETER DEFINITION

 Trace                      ===> NO           Yes,No
 Entries in EFHUSTG table      : 500          100-1000
 Trace table filemode          : Z            2 characters
 Trap program checks           : YES          Yes,No
 DEBUG program              ===>              1-8 characters
 Dynamic storage               : 160000       50000-4000000
 Language                   ===> COBOL        COBOL, COBOL2, ASSEMBLE or PLI
 TWA size                      : 1024         1024-32767
 SYSID for remote system       : REMT              4 characters

 Names of resource tables:       Filename Filetype Filemode
   Program/Transaction         : CICSCMS  EFHTPROG *
   File                        :          EFHTFILE *
   Transient Data              :          EFHTTD   *
   Temporary Storage           :          EFHTTS   *
   PSB Directory               :          EFHTPDIR *

                                  Name    Value
 Additional Parameter       ===>

PF1=Help          PF3=End           PF12=Quit          PA2=CMS Subset
```

Figure 43. Changing CICS/CMS parameters from the escape panel

This panel displays the values of your main CICS/CMS parameters, but the only ones you can change are those flagged with arrows (===>).

You can, for example, turn on trace in the middle of testing an application (as described in "General debugging tools" on page 171). You cannot, however, change things like your table file IDs or your remote SYSID. All you can do is find out what the current values are, by entering the keyword (`Name`) in the `Additional Parameter` line, and pressing ENTER.

General rules for using panels EFH121 and EFH1221

There are some general rules and guidelines for using panels EFH121 and EFH1221, as explained in the following sections.

Scope of the changes

Changes you make using panels EFH121 and EFH1221 stay in effect throughout your CICS/CMS session. The parameters revert to their default settings only when you return to CMS or enter `EFHSETP` from the `CP/CMS Command` line of panel EFH12.

Scope of the language definition

The language you specify in EFH121 or EFH1221 applies only to programs that you haven't defined in the active CICS/CMS program table.

Blank table names

If all your resources are local, you probably won't have any tables for CICS/CMS files, transient data, or temporary storage. In this case, the resource table names fields in panel EFH121 will be blank, as shown in both examples for every table except the program table.

Using panel EFH121 to define how program checks are handled

If you specify `yes` for `Trap program checks` on panel EFH121, CICS/CMS will construct an error handler message for program checks, and then issue an ASRA abend.

If you specify `no`, program checks will produce a CMS message. You can then proceed as you would for a program check in a CMS system that wasn't running CICS/CMS. You'll find a complete description of VM techniques for debugging program checks in the *VM/SP System Programmer's Guide.*

You will find a description of how CICS/CMS handles program checks in "Program checks" on page 194.

Changing terminal characteristics from panel EFH121

Since you will not often want to change your terminal characteristics from those defined in the EFHSETP EXEC, or in your own EFHPROF EXEC, neither panel offers a direct way of changing them. You can, however, use the `Additional Parameter` line on panel EFH121 to do so.

Changing the dynamic storage area size from panel EFH121

Be careful when changing the size of the dynamic storage area (DSA) on panel EFH121. If you make it too large, you might not leave CICS/CMS enough room to load your programs. If you make it too small, you might not leave CICS enough room for its own working space.

The default that IBM supplies in EFHSETP, 256000 bytes, will be enough if you are only using some of the CICS/CMS tables, and if they are relatively small. However, as your tables grow, so should your DSA.

You will also need a larger DSA if you run pseudoconversational applications that use many temporary storage queues, because CICS/CMS holds all such queues in main storage.

Cancelling changes made on panels EFH121 and EFH1221

Every time you press ENTER, CICS/CMS applies the changes you've made. When you've made all the changes you want, you can press PF3 to save the last set of changes and return, either to the execution panel (from EFH121), or to the point from which you escaped (from EFH1221). If you make several changes and, before you press ENTER, you realize that you've made a mistake and need to clear the changes, you can press CLEAR. The panel will refresh, showing the values that were in effect when you last pressed ENTER.

To reset all the parameters you've changed during a CICS/CMS session and return to your default environment, you can run EFHSETP from the `CP/CMS Command` line of EFH12. When EFHSETP ends, it will execute EFHPROF.

Chapter 9. Testing an application

When you're ready to start testing your application, you need to display the CICS/CMS execution panel (EFH12). Enter `CICSCMS` to start CICS/CMS, if necessary, and select option 2 on the selection panel (EFH1).

Panel EFH12 is as shown in Figure 7 on page 19. On the line that says:

```
Transaction      ===>
```

type the transaction ID of the first transaction in your application, and press ENTER.

If your transaction needs data passed with the transaction ID, there are two ways you can do it:

1. If the transaction ID and all the data will fit on the `Transaction` line, you can enter it there. The transaction ID and the data are translated into uppercase. The maximum number of data characters you can enter in this way is 47.

2. If you need to enter more data than will fit on the `Transaction` line, you can press PF6 on panel EFH12 to get a blank screen. You can then enter your transaction ID and data on that screen. The data will only be translated into uppercase if the UCTRAN parameter is set to YES in EFHSETP or EFHPROF.

You also need to use PF6 to get a blank screen if you want to start a transaction with a PF key. CICS/CMS doesn't let you run "PF key" transactions from panel EFH12 itself, in case your PF key definition conflicts with any of its predefined PF key settings.

If your application finishes successfully, CICS will wait for you to enter another transaction ID. If you want to return to the panel EFH12, you can:

1. Type `CCMS QUIT`.

2. Press PA2, to get to the escape panel (EFH122), then press PF12 to return to panel EFH12.

3. Press ENTER, which will display an EFH125 error handler panel. Press PF12 to return to panel EFH12.

The CICS environment

When you start a transaction or program from panel EFH12, CICS/CMS initializes CICS, and you enter the *CICS environment.* You stay in that environment until you return to panel EFH12. The time you spend in the CICS environment is known as a *CICS test session.*

There are some important points to remember about what you can do in the CICS environment, as explained below.

Using PA2 in the CICS environment

While you are in the CICS environment, you can press PA2 at any time that CICS is waiting for input on the screen, and display the CICS/CMS escape panel, EFH122. You can therefore escape:

- From your own transactions, when they are waiting for input
- At the end of a transaction, when CICS is waiting for a new transaction ID
- From any CEDF, CECI, or CEBR display.

Pressing PA2 on the CICS/CMS panels outside the CICS environment puts you in the CMS subset.

You'll find a complete description of the CICS/CMS escape feature in "Using the CICS/CMS escape feature" on page 127.

Resources lost at the end of a CICS test session

Whenever you end a CICS test session, and return to panel EFH12, you will lose:

- All temporary storage queues
- Any transactions that CICS/CMS has scheduled for execution using interval control, including the CSPS transaction for displaying BMS pages
- The input positioning of transient data queues.

Returning to panel EFH12 at the end of a CICS test session

When a transaction, or the last transaction in a pseudoconversation, ends, the CICS environment expects you to enter either:

1. Another transaction ID, or
2. The `CCMS QUIT` command, to end the CICS test session and return to panel EFH12.

If you forget that you're in the CICS environment at this point, and press ENTER or a PF key that you haven't associated with a transaction in your active program table, CICS/CMS will treat your entry as an attempt to execute an illegal transaction ID. It will display an appropriate error handler panel. On this panel (EFH125), you can press PF12 to return to panel EFH12.

Testing tools

CICS/CMS provides a wide range of tools that make it easy for you to test your applications. The main ones are the CICS/VS interactive tools: CEDF, CEBR and CECI. To let you take full advantage of having CICS and CMS running together, however, CICS/CMS has the escape feature.

Whenever CICS is waiting for input, you can press PA2 to display the escape panel, EFH122. You can escape from one of your application's displays, or from a panel produced by one of the testing tools, such as EDF. Panel EFH122 offers similar facilities to those on the execution panel (EFH12). "Using the CICS/CMS escape feature" on page 127 suggests some ways of using this feature.

The execution diagnostic facility

CICS/CMS supports all the EDF functions described in the *Application Programmer's Reference Manual.* However, because CICS/CMS is a single-user system, you can't use EDF in "two-terminal" mode to use one interactive terminal to monitor a transaction running on another interactive terminal. You can only use EDF in two terminal mode to monitor a printer transaction from your interactive terminal.

CICS/CMS EDF features

There are some things that you can do with EDF in CICS/CMS that you can't do in CICS/VS, as follows:

- Whenever an application abends, CICS/CMS starts EDF (if you haven't got it running already), so that the next screen displayed will be the standard EDF screen for reporting abends. It then displays an error handler panel (EFH125) giving a detailed description of the abend.
- You can use the escape feature on any EDF display. Pressing PA2 displays panel EFH122, from which you can execute commands within the CMS subset, run CECI, and so on.
- If you are using a PC, you can print any EDF screen on your PC printer, by pressing the Shift and PrtSc keys at the same time.

Turning EDF on and off

In the EFHSETP and EFHPROF EXECs, there is a parameter that defines the status of EDF at the start of every CICS test session. You can ask for EDF to be either on or off by default.

When you display panel EFH12, its `Status` line will tell you whether EDF is on or off. You can change its status by pressing PF9. For example, if, when you display panel EFH12, the `Status` line says `EDF is OFF`, and you want EDF on for your application test, you press PF9. The `Status` line will change to `EDF is ON`, and the definition of PF9 on the display will change from `EDF ON` to `EDF OFF`.

Once you've turned EDF on, it stays on until:

- You turn it off by pressing PF9, on either panel EFH12 or the escape panel (EFH122)

- You turn it off by pressing PF3 on any EDF panel which has PF3 defined as `END EDF SESSION`
- You turn it off on the EDF display at the end of a transaction, by leaving the EDF status field as `NO`

If you change the status of EDF by pressing PF9 on panel EFH122, or PF3 on an EDF display, you change it only for that CICS test session. When you return to panel EFH12 at the end of the test, you will find that EDF is still in the state it was in when you left panel EFH12 at the start of the test.

You can turn EDF on while an application is running, by pressing PA2, and starting EDF from panel EFH122.

The command level interpreter

CICS/CMS supports the command level interpreter (CECI) fully. You can start it by pressing PF4 on either panel EFH12 or panel EFH122. Alternatively, you can press PF6 on panel EFH12, to get a blank screen, and type `CECI`.

Note: The commands that you can ***execute*** using CECI are governed by the same rules that apply to all CICS/VS commands in CICS/CMS, as described in the *CICS/CMS Application Programmer's Reference Summary*. However, you can use CECI in CICS/CMS to check the ***syntax*** of all the CICS/VS commands that it recognizes.

Using CECI from panel EFH122 gives you a useful testing tool. For example, suppose you are running an application in which one transaction writes to a temporary storage queue, and a later transaction reads the data from that queue. If you are running the application with EDF on, and, for some reason, the first transaction fails to create the temporary storage queue correctly, you can press PA2, and display panel EFH122.

From there, you can execute CECI, and create the temporary storage queue with `EXEC CICS WRITEQ` commands. When you've finished, you press PF3 once (to end CECI), again (to return to panel EFH122), and then again (to return to your application).

You can then continue with your test, and check to see whether there are any problems in the second transaction, even though the first, on which the second depends, has failed.

For a complete description of CECI, see the *CICS/VS Application Programmer's Reference Manual.*

The temporary storage browse facility

You can use the interactive transaction CEBR to browse the contents of a named temporary storage queue. CEBR offers several facilities, including the ability to get data from, and write data to, transient data queues.

One feature of CEBR that you might find useful is the `GET` subcommand. You can use it to create test temporary storage data. It retrieves data from a CMS file and adds it to a defined temporary storage queue. The CMS filename must be four characters long, because CEBR treats it as a transient data queue. Its filetype must be the one defined in the INTRAFT parameter in EFHSETP or EFHPROF.

CICS/CMS supports CEBR fully. You can execute it from the CICS/CMS panels EFH12 and EFH122 by pressing PF2. You can also use it from an EDF display, by pressing PF5 (to display working storage), then pressing PF2 on the working storage display.

For a complete description of CEBR, see the *CICS/VS Application Programmer's Reference Manual.*

Using the CICS/CMS escape feature

The CICS/CMS escape feature lets you suspend the transaction you are running, perform some related (or unrelated) function, and return to the transaction, picking up exactly where you left off.

You can escape to panel EFH122 at any time that CICS is waiting for input. You can therefore escape from one of your own transactions whenever it is waiting for input from the terminal, from CICS between transactions, or from any EDF, CECI or CEBR display. You escape by pressing PA2. This will display the panel shown in Figure 20 on page 32.

Panel EFH122 offers the same sort of facilities as the execution panel, EFH12. The sections that follow describe these facilities and suggest ways of using them.

Note that we only describe those uses of the escape feature most immediately concerned with testing applications. We describe some more in Chapter 14, "How CICS/CMS reports problems" on page 159. We don't want to imply, however, that these are the only uses to which you can put the feature. By providing access to the CMS subset, panel EFH122 gives you the freedom to switch between developing applications using CICS/CMS, and anything else you want to do, without having to stop your CICS test session.

Turning EDF on and off (PF9)

Suppose you are testing an application that consists of four pseudoconversational programs. You know from a previous run that there is a problem in the final program, and you want to run it with EDF. If you turn on EDF before running the application, you will spend some time pressing ENTER to step through a series of EDF screens that you don't need to see.

Instead, you can start the application without EDF on.. When you reach a convenient point (such as when the third program displays a panel), press PA2. On panel EFH122, you press PF9, followed by PF3. Your application will continue its run, but it will now have EDF enabled.

You can also use PF9 the other way round. You can start testing an application with EDF on, then escape to turn it off when you don't need it any more.

Executing CECI (PF4)

There are limitations to the usefulness of CECI in the escape environment. Its function as an online help facility for the syntax of EXEC CICS commands has little application when you're executing transactions or programs. However, you may find its "repair" capability useful to correct temporary storage queues, transient data queues, or even data files, while you are executing an application. We describe one possible use of CECI from the escape panel in "The command level interpreter" on page 126.

Executing CEBR (PF2)

Suppose that you are running a pseudoconversational transaction in which a program reads from a transient data queue created by a previous program. You can make sure that the previous program wrote to the queue correctly by pressing PA2 before the reading program starts. On panel EFH122, you can then press PF2 to start CEBR, and check the queue. If it seems to be correct, you then press PF3 twice to get the transaction going again. If you find something wrong with the queue that makes you want to stop the run, you can press PF12 on panel EFH122. This will return you to panel EFH12 without finishing the transaction.

Starting another VM session (PF8)

If you are using CICS/CMS under VM/SP, you can press PF8 on panel EFH122 to start another VM session and log on to another VM userid. You'd usually use this to connect to your remote CICS/VS system and start the remote server.

Using the CP/CMS command line

From the CP/CMS command line, you can enter any CP or CMS command that is valid within the CMS subset. For example, you can:

- Use CMS `FILELIST` to check the existence of files, erase unwanted files to free disk space, and so on
- Edit data files
- Edit source program files
- Use CP or CMS debugging facilities
- Do things that may be quite unrelated to your application test, such as sending messages to other VM users.

You can't use CMS commands that aren't valid in the CMS subset, such as `COPYFILE`. Nor can you run any CICS/CMS EXECs. You can, however, do so outside the CICS environment, from the CP/CMS command line on panel EFH12.

Testing an application's use of PA1 and PA2

Some applications ask their users to press either PA1 or PA2 to request a particular function. You can't test the use of these PA keys directly using CICS/CMS, because both keys have defined functions. PA1 takes you to CP; PA2 takes you either to the CMS subset or, within a CICS test session, to panel EFH122.

So that you can test applications that need the PA keys, we've provided PF5 and PF6 on panel EFH122. If you want to pass either PA1 or PA2 to your application, without their being intercepted by CICS/CMS, you have to escape from your application (press PA2), and press PF5 (for PA1) or PF6 (for PA2). In these cases, you don't then have to press PF3 to get back to your application. Both PF5 and PF6 issue a PF3 implicitly.

Running nested transactions

From panel EFH122, you can execute a program or transaction by typing its name on the relevant input line, just as you can from the execution panel (EFH12). The one difference is that you can't start transactions by pressing PF6 to get a blank screen, as you can from panel EFH12. PF6 is reserved for passing PA2 to an application.

Note also that you can't run pseudoconversational applications from panel EFH122. Whenever a transaction that you've started from the escape panel ends, your next keystroke returns you to the escape panel.

The only real limit on the number of times you can escape and run transactions is the amount of virtual storage you have available. In theory, there's no reason why you shouldn't run an application, escape to use CECI, then escape from CECI to run a different application, then escape from that application to run CEBR, and so on. In practice, you'd quickly run out of virtual storage, and you'd get a "short-on-storage" message, after which CICS/CMS would stop. Anyway, nesting escapes is not something you're likely to want to do in the way described above, but it's quite easy to nest one or two escapes, and then forget where you are.

The line on panel EFH122 beginning `Nest level` tells you where you are. Every time you start a new transaction, whether it's one of your own, or a CICS-supplied transaction such as CECI, CICS/CMS puts a slash (/) in that line. Every time a transaction executes an `EXEC CICS LINK` to a program, CICS/CMS puts an asterisk (*) in that line. When a program ends, CICS/CMS removes its asterisk. When a transaction ends, CICS/CMS removes its slash, and the asterisks for any programs associated with the transaction.

For example, suppose you start a transaction and then escape from the first program in the transaction and start CECI. If you then escape from CECI, you will see the following nest level indicator on the escape panel:

```
/*/*
```

The first `/*` is your transaction and program; the second `/*` is CECI, and its associated program.

One important point to remember is the effect of PF12 when you press it on panel EFH122. PF12 ***always*** takes you back to the execution panel (EFH12), no matter how much nesting you've done. The same applies when a transaction abends. No matter how many times you may have escaped before starting that transaction, CICS/CMS always returns to panel EFH12.

Testing CICS/VS features that CICS/CMS does not fully support

In "CICS/CMS support for the CICS/VS API" on page 90, we listed some features of CICS/VS that CICS/CMS does not fully support. These are mainly features that have no sensible meaning in the CICS/CMS environment, such as those related to a multiuser environment, or those related to devices that CICS/CMS neither supports nor simulates. Generally, CICS/CMS takes no extreme actions when it finds these features in your applications. In most cases, it displays a message warning you that you've used a feature that CICS/CMS doesn't support, and continues with the test.

The only way that you can properly test applications that use features that CICS/CMS doesn't support is to test them on a test CICS/VS system. However, you can use CICS/CMS to test the logic of applications containing unsupported features. The tool that lets you do this is EDF.

Let's assume that you want to test a transaction that writes journal records to the system log. CICS/CMS does not support journal control, and will ignore all EXEC CICS JOURNAL commands. (That is, it won't execute them, but will carry on with the application as if they had worked.) It will alert you, however, by displaying warning messages like the one shown in Figure 44.

```
EFH125                    ERROR HANDLER FUNCTIONS    10/18/85 10:24:55

Program Name: EMILY                                  Line Number: EMI00010

Message EFH9400S has been generated

After this screen the CICS environment will CONTINUE

Journal Control is not supported by CICS/CMS.

Request ignored, but subsequent execution may fail




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 44. Message panel for a journal control request

If you were running the program EMILY with EDF turned on, the next thing you would see would be the usual EDF execution completion panel showing a NORMAL response.

Whenever CICS/CMS treats a command in this way, the "normality" of the response is limited. CICS/CMS updates the EIBRCODE in the exec interface block (EIB) to show that the command has worked correctly but, because the command hasn't in fact done anything, CICS/CMS leaves all the other EIB fields just as they were before the command. This is why the error handler message warns you that, although it intends to carry on with your application, it can't guarantee its success.

Whether the rest of the test works or not, the way that CICS/CMS treats commands it doesn't support leaves you with an apparent problem. How can you find out what will happen if a journal control command fails? How can you check the logic that you've put in your program with EXEC CICS HANDLE CONDITION commands to cater for anticipated errors? This is where EDF comes in.

If you execute a transaction containing EXEC CICS JOURNAL commands, with EDF on, you will get EDF displays for each command. Eventually you will reach the point where a JOURNAL command has been executed, and you are looking at an EDF display showing a NORMAL response. By changing the response from NORMAL to one of the errors associated with the command, you can make sure

that you test any HANDLE CONDITION commands you have specified for the command, and also find out what will happen for errors that your program doesn't handle.

By trying every possible condition, using EDF, you should be able to test all the execution paths through your application. You cannot be 100% certain that your application will work in production, but you you will have have ensured that it "fits together" properly. You should be able to progress with confidence to testing it on a test CICS/VS system.

Chapter 10. Testing applications that print

On a ***CICS/VS*** system, an application can produce printed output in two ways:

1. It can write output to an extrapartition data set associated with a system printer.

2. It can direct output to a 3270 printer terminal, using either:

   - Terminal control `EXEC CICS SEND FROM` commands to send 3270 printer data streams, or

   - BMS SEND MAP, TEXT and/or CONTROL commands to generate data to be printed by a printer transaction.

   In this case, you usually start the printer transaction using the interval control function `EXEC CICS START`. Alternatively, you can associate the printer terminal with an intrapartition queue in the destination control table (DCT), and specify that the printer transaction is to run when the number of records in the queue reaches a defined "trigger level".

On a ***CICS/CMS*** system, you can use the first CICS/VS method by defining a suitable transient data destination. You can associate this directly with your virtual printer, or direct the output to a CMS file so that you can check it interactively using your editor. This is described in "Printing from extrapartition queues" on page 58.

CICS/CMS also lets you test applications that print output on 3270 printer terminals. This is described in "Testing applications that print on a 3270 printer terminal" on page 134. You can use CICS/CMS to test:

- Printer transactions started by `EXEC CICS START` commands

- Printer transactions that, on CICS/VS, will be triggered from a transient data queue.

  However, as explained in "Testing applications that print on a 3270 printer terminal" on page 134, you can only test the transactions themselves; you can't test the triggering mechanism.

Note: CICS/CMS supports neither of the CICS/VS "immediate" print facilities: the ISSUE PRINT command and the local copy key. You can, however, get the effect of the local copy key if you're using a PC, by pressing the Shift and PrtScr keys to copy the current screen contents on your local printer.

Testing applications that print on a 3270 printer terminal

CICS/CMS lets you test applications that direct output to 3270 printer terminals by simulating those terminals. It does this by directing all such output to a CMS file that you define in your EFHPROF EXEC. This gives you the choice of printing the output using the CMS `PRINT` command, or inspecting the file using your editor.

CICS/CMS lets you test applications that:

- Generate 3270 data streams and direct them to the printer terminal using commands of the form:

  ```
  EXEC CICS SEND FROM (data-area)
  ```

  where `data-area` contains the 3270 printer data stream.

- Use the BMS commands `SEND MAP`, `SEND TEXT`, and `SEND CONTROL`, with or without the `NLEOM` option, to generate and direct data to the printer terminal.

You'll find detailed information on both these techniques in the *CICS/VS Application Programmer's Reference Manual.*

There are limitations on what printer functions you can test in two areas, as follows:

1. 3270 printer data streams
2. SNA character string (SCS) printers.

CICS/CMS support for 3270 printer data streams

CICS/CMS supports the following 3270 *format control orders*:

FF form feed

CR carriage return

EM end of message

NL new line

CICS/CMS does not support all the 3270 *data stream orders.* It supports the data stream orders that apply to all 3270 devices:

SBA set buffer address

SF start field

IC insert cursor

PT program tab

RA repeat to address

EUA erase all unprotected to address

It does not support data stream orders that require the structured field and attribute processing options to be installed in the printer control unit:

SFE start field extended

MFA modify field

SA set attribute

It does not support any of the SCS orders, such as vertical tabs.

If CICS/CMS finds any unsupported orders in a 3270 data stream, it treats them as unprintable characters, so you won't be able to rely on the accuracy of the output.

There is one further restriction on what you can put in a 3270 data stream. 3270 printers support only 12-bit addresses with their printer control orders. To simulate 3270 printers as closely as possible, CICS/CMS treats any 14-bit or 16-bit addresses your applications use as 12-bit addresses, with unpredictable effects on the output.

For the rules governing the creation of 3270 printer data streams, see the *3270 Information Display Data Stream Programmer's Reference* manual (GA23-0059).

Testing SCS printer applications

CICS/CMS doesn't fully support SNA Character String (SCS) printers. However, you can test applications destined to be run on SCS printers, as long as they conform to the restrictions described below.

In CICS/VS, applications will work on both SCS and 3270 printers under the following conditions:

- If an application generates its own 3270 data streams, the `EXEC CICS SEND FROM` commands must specify ERASE and HONEOM, and the data streams must not include SBA sequences.
- If an application uses BMS, with the NLEOM option, it will produce the same results on both 3270 and SCS printers.
- If an application uses BMS, without the NLEOM option, it will produce the same results on both 3270 and SCS printers, as long as:
  - The map data fields don't contain NL or EM characters when the SEND commands include the L40, L64, or L80 options.
  - There are no completely null lines when the SEND commands include the L40, L64, or L80 options.
  - The BMS page size matches the buffer capacity in rows of the length specified by either L40, L64, L80, or HONEOM (printer width).

You can therefore use CICS/CMS to test applications destined for SCS printers under the following conditions:

- Applications generating 3270 data streams must have the ERASE and HONEOM options on the SEND commands, must avoid vertical and horizontal tabs, and must include an extra EM control at the end. For production use on

an SCS printer, you could simply remove the last EM. CICS/VS will ignore the ERASE and HONEOM options.

- Applications using BMS must always have the NLEOM option on the SEND commands.

EFHPROF definitions for printing

Before you can test printer applications, you need to tell CICS/CMS how and where you want the printed output to appear. You have to:

1. Define the page and printer size.
2. Define the simulated printer

Defining the page and printer size

In CICS/VS, you define a printer terminal using appropriate terminal control table (TCT) entries, or resource definition online (RDO) statements. In CICS/CMS, your system administrator will make similar definitions in EFHSETP EXEC, and you can make your own in EFHPROF EXEC.

The parameters you can set are:

PRTBMSHT/PRTBMSWD

Together these parameters define the shape of the BMS page buffer. PRTBMSHT defines the number of lines in a BMS page; PRTBMSWD defines the number of characters in each line. They are equivalent to the CICS/VS TCT PGESIZE entry for the printer.

BMS uses PRTBMSHT and PRTBMSWD to format its output. For NLEOM data streams, BMS inserts a new line (NL) order when a line reaches a width of PRTBMSWD characters. For non-NLEOM data streams, BMS uses PRTBMSWD as the page width when calculating buffer addresses. It calculates the buffer address of the nth line in the output using the formula:

```
(n-1) * PRTBMSWD + 1
```

The value you specify for PRTBMSWD depends on whether your printer transaction uses the NLEOM option, as follows:

- ***For BMS output with the NLEOM option***, the PRTBMSWD value should be less than or equal to the PRTFRMWD value (see below).

  If the PRTBMSWD value equals the PRTFRMWD value, BMS may insert the NL order in the data stream at the position represented by PRTFRMWD+1. This produces an extra blank line in the output. This will only happen if your application writes a line that is exactly PRTFRMWD characters wide.

  If your application produces a map that is wider than PRTBMSWD or longer than PRTBMSHT, it will raise the INVMPSZ condition.

  Note: When you use BMS with the NLEOM option, the line length options (L40, L64, and L80) have no effect on the output.

- ***For BMS output without the NLEOM option***, the value you give to PRTBMSWD depends on whether your application uses the "L" options (L40, L64, and L80) in its `EXEC CICS SEND` commands as follows:

  - If the application specifies the line length using one of the "L" options, the PRTBMSWD value must be the same as the line width specified. The "L" options set up the write control character (WCC) to indicate a formatted output width of 40, 64, or 80. PRTBMSWD is used to calculate buffer addresses. If you set PRTBMSWD to anything other than the "L" value your application uses, your printer output will not format correctly.

  - If the application doesn't use one of the "L" options to specify the line width, the PRTBMSWD value must be 40, 64, or 80, or the same value you give to PRTFRMWD. If you specify PRTBMSWD as 40, 64, or 80, your application will send a formatted data stream of the width selected. Otherwise, your application will send an unformatted data stream, so the buffer addresses that BMS calculates must correspond to those used by the printer.

PRTFRMHT/PRTFRMWD

Define the maximum number of lines on each printer page, and the maximum width of each line. These correspond to controls on a 3270 printer. PRTFRMHT defines the page size that will be used when an application issues a form feed (FF). If an application uses HONEOM, and doesn't contain NL controls, PRTFRMWD defines when to start new lines. If an application uses L40, L64, or L80, PRTFRMWD is ignored.

PRTBUFF

Defines the printer buffer size.

Note: To ensure that that the CICS/CMS simulated printer file matches the printed output as closely as possible:

- The values you specify for PRTFRMHT and PRTFRMWD should match the corresponding values you'd set on the printer terminal using switches.

- The value you give PRTBUFF should match the printer terminal's hardware buffer size.

The EFHSETP EXEC that IBM supplies sets the printer parameters as follows:

```
SETPARM 'PRTBMSHT 24'
SETPARM 'PRTBMSWD 80
SETPARM 'PRTFRMHT 66'
SETPARM 'PRTFRMWD 132'
SETPARM 'PRTBUFF 1920'
```

Defining the simulated printer

CICS/CMS simulates the 3270 printer by writing output destined for the printer to a CMS file. You can define this file in your EFHPROF, and your system administrator can set a default in EFHSETP. The EFHSETP that IBM supplies contains the following definitions:

```
SETPARM 'P3270FN CICSCMS'
SETPARM 'P3270FT P3270'
SETPARM 'P3270FM A1'
```

If you use these definitions, all output that your applications write to a 3270 printer terminal will go to the file CICSCMS P3270 on your A-disk.

You also need to name the printer terminal, that is, you need to provide a unique identification that will distinguish the simulated printer from your screen. There are two parameters for naming terminals in CICS/CMS: TERMID and TERMID2. TERMID names your interactive terminal; TERMID2 names the printer terminal. For example, the following EFHPROF statements give the interactive terminal the name L866, and the printer terminal the name PRNT:

```
SETPARM 'TERMID L866'
SETPARM 'TERMID2 PRNT'
```

The EFHSETP statements that IBM supplies with CICS/CMS set TERMID to CONS, and TERMID2 to PRNT.

You use the TERMID2 specification in the TERMID option of the EXEC CICS START command for the printer transaction, or as an option on the CCMS START command, as explained in "Running printer transactions" below.

Running printer transactions

In CICS/VS, you usually start printer transactions in one of two ways:

- By executing an interval control EXEC CICS START command that names a printer terminal in the TERMID option, or
- By setting a trigger level on a transient data queue that starts a printer transaction once that queue contains a defined number of records.

You can use CICS/CMS to test printer transactions that, in production, will start by either of these methods. You can:

- Start the printer transaction directly, without running any transactions that start it. This is the only way of testing triggered transactions.
- Start the printer transaction using interval control. This lets you test an application that contains an EXEC CICS START command for a printer transaction.

Both methods use the `CCMS START` command to run the printer transaction.

Starting printer transactions directly

To run a printer transaction that will be triggered when you run it on CICS/VS, or to run a printer transaction without running the transaction that starts it, you use a CCMS command of the general form:

```
CCMS START termid transid
```

where `termid` is the name defined for the printer terminal in TERMID2, and `transid` is the transaction ID, defined in a CICS/CMS program table. For example, if you issued:

```
CCMS START PRNT PICT
```

CICS/CMS would execute the transaction PICT, on the terminal PRNT, defined as the simulated printer in EFHPROF.

You can issue the CCMS command in all the ways that you can execute any transaction in CICS/CMS:

- In the `Transaction` line of an EFH12 or EFH122 panel
- On the blank screen produced when you press PF6 on panel EFH12
- On the blank screen left after a transaction ends in a CICS test session.

Starting a printer transaction using interval control

As explained in "Interval control" on page 54, when you start a transaction using EXEC CICS START, CICS/CMS puts that transaction onto a queue. You can then execute the transaction when convenient, using the CICS/CMS control transaction CCMS START. You can use this technique to test printer transactions.

First, you execute a transaction that issues an EXEC CICS START command with a TERMID option set to the name given for the simulated 3270 printer in the EFHPROF or EFHSETP TERMID2 parameter. This schedules the printer transaction. You then issue CCMS START and, assuming the printer transaction is the first in the queue of scheduled transactions, it will execute, writing its output to the CMS file defined in the P3270xx parameters.

"Interval control" on page 54 gives a complete description of running transactions using interval control.

The `termid` option on the CCMS START command gives you the opportunity to check your printer output on the screen, instead of writing it to your simulated printer.

For example, suppose that, after running the transaction that schedules the printer transaction, you decide that you want to run the printer transaction, but display the output directly on your screen. You can use the `termid` option on CCMS START to run the transaction on your virtual console. This lets you change the destination of the output without altering either your EFHPROF definitions or the transaction itself. For example, if your EFHPROF contains the definitions:

```
SETPARM TERMID L866
SETPARM TERMID2 PRNT
```

and a transaction has scheduled the printer transaction with the command:

```
EXEC CICS START TRANSID(PICT) TERMID(PRNT)
```

you can direct its output to your screen by issuing:

```
CCMS START L866
```

Notes

1. You can only use the `termid` option to switch between your interactive terminal and your printer terminal if your transaction uses BMS to create the output for printing. If the transaction generates its own 3270 data stream, it must also set the "print" bit in the write control character (WCC). You can't, therefore, run the transaction to produce screen output without changing the WCC. The only way round this is to include extra code in the transaction. You can use the command `EXEC CICS ASSIGN TERMCODE` to find the device type, and only set the WCC "print" bit on if the device is a printer.

2. The form of CCMS described here, without the `transid` parameter will only work if you have already scheduled the printer transaction. If you want to run a printer transaction without scheduling it, you must use CCMS START with both the `termid` and `transid` options, as described in "Starting printer transactions directly" on page 139.

Output from a 3270 printer transaction

When a printer transaction finishes, you have a CMS file that contains your printed output. As long as your transaction has created that output using only those methods that CICS/CMS supports, the file output should look the same as your printed output would look if you had run the transaction on a real printer. You can use the CMS `PRINT` command to copy it to your virtual printer. If you're using a PC, `PRINT` will usually print the output on your local printer, as explained in the *VM/PC User's Guide*. Alternatively, you can inspect the output with your editor. To make it as easy as possible for you to view it in this way, CICS/CMS uses blanks, rather than nulls, for spaces between items, blank lines, and so on, and replaces unprintable characters with a minus sign (−).

General rules for 3270 printer transactions

1. A printer transaction must not try to get input from the terminal. CICS/CMS diagnoses any EXEC CICS RECEIVE commands in a printer transaction, and displays a warning message. This message won't stop the transaction, as you might be testing a transaction destined for an SCS printer, which can transmit input to an application.

2. You can enable EDF for a printer transaction. As in CICS/VS, the EDF output will appear on your screen.

3. CICS/CMS always `appends` printer transaction output to the CMS file you've defined for your simulated printer. Unless you arrange for that file to be empty before running each printer transaction, you should end each printer transaction with a FORMFEED. This will ensure that output from a subsequent printer transaction will start at the top of a simulated printer page.

Chapter 11. Making program corrections and retesting

Let's assume that you've run your application, and that it has failed. You now want to make any necessary corrections, and try again with the corrected version.

At this stage, you may be looking at a CICS/CMS error handler panel (EFH125). Although you can escape from this panel, using PA2, you can't assemble maps, translate and compile programs, or re-create data files using the CICS/CMS file utilities, from panel EFH122. You'll need to press PF12 on the error handler panel, and return to the execution panel (EFH12). From there, you can press PF3 to get back, either to the panel EFH1, or to panel EFH11, or to CMS, depending on how you got to panel EFH12 in the first place.

If you go back to panel EFH1, make sure that the `Criteria` line contains sufficient information to give you a list of at least those files you want to change. Select option 1, and you will get panel EFH11. You can then use your editor to change those source programs and maps that need corrections. When you've made all the changes you want, you can reassemble your source maps (press PF10), and retranslate and recompile your source programs (press PF5). You don't have to erase the existing TEXT files before doing so. CICS/CMS automatically replaces any existing TEXT files when it creates new ones from source files with the same filename.

Assuming all your assemblies, translations, and compilations are successful, you can go to panel EFH12 by first pressing PF3 on panel EFH11, then selecting option 2 on panel EFH1. Before retesting your application, you may need to redo some of the preparation you did before the last test.

You won't need to update any tables. Once you've defined the resources you need for an application, you don't need to change those definitions unless you change the name (or sometimes the form) of those resources.

If your application uses remote resources, you will need to have the remote server running on your remote CICS/VS system. If you started the remote server for the last test, it will still be running, unless it failed or you stopped it yourself. If you're using a PC, you can check this simply by looking at the bottom right-hand corner of your screen. If you see a minus sign (−), the remote server is still running. If you're using a host-connected terminal, you will have to press PF8 on panel EFH12. If your remote system is still available, and the remote server is still running, you will see the remote server logo, with a status of `RUNNING`.

If your application used any local data files, it may have changed them, or even corrupted them, in some way. Before retesting the application, you will need to restore them to their original state. If you have followed the suggestion in "Safeguarding your data files" on page 117, you can make new copies from your master files. If you haven't got master files, but you created your data files using either of the CICS/CMS conversion utilities, you can restore your files by running

the applicable utility, as described in "Converting local CMS files to CICS/CMS keyed files" on page 110 and "General file conversion" on page 113.

If your application used an extrapartition queue associated with your virtual reader for input, and you didn't use the CP `SPOOL` command with the `HOLD` option before reading the queue, you will need to re-create the queue.

Finally, if you had any problems that were so severe that you had to restart CICS/CMS or CMS, your CICS/CMS environment will have returned to the default set in EFHSETP and EFHPROF. If you made any temporary changes to your working environment using the parameter panel (EFH121), you will have to make them again. "Changing your CICS/CMS environment within a session" on page 119 tells you how to do this.

Chapter 12. Shortcuts for experienced CICS/CMS users

As you become familiar with CICS/CMS, you may find that having to go through the panels slows your work down.

This chapter tells you how to use commands to run some of the CICS/CMS EXECs without using the panels. In most cases, you can use these commands:

- From the `CP/CMS command` line of panel EFH12
- From any CMS `FILELIST` display
- As commands in CMS.

You can't, however, use any of these commands from the `CP/CMS Command` line of the escape panel (EFH122), because you can't run CICS/CMS EXECs within the CMS subset.

Assembling BMS maps

The CICS/CMS EXEC that assembles BMS maps is called EFHMAPCR. You can assemble maps by entering the `EFHMAPCR` command alongside the name of a map file in any CMS `FILELIST` display, or by using it as a stand-alone command in CMS. The full form of the command is:

```
EFHMAPCR fn ft fm (style
```

or, from a file list:

```
EFHMAPCR / (style
```

where:

`fn` is the filename of the file containing the source of your map. You must include this.

`ft` is the filetype. If you omit this, EFHMAPCR assumes the filetype ASSEMBLE. If you are using the F Assembler (the default for VM/SP users), the file ***must*** be of filetype ASSEMBLE. If you are using the H Assembler (the only possibility for PC users), the filetype can be anything acceptable to the H Assembler, except ASMLIST1 and ASMLIST2 (which EFHMAPCR uses for the listing files), and COPY (which EFHMAPCR uses when it adds the DSECT to the macro library).

`fm` is the filemode of the file. If you omit this, EFHMAPCR will search all the disks to which you currently have access, starting with your A-disk.

When you use EFHMAPCR on a file list, `fn ft fm` are replaced by `/`, which tells EFHMAPCR to use the file over which you enter the command.

`style` is the kind of output you want from the EXEC. You must separate this from the file identifier parameters with an open parenthesis. The style can be:

`MAP` to produce the physical map only

`DSECT` to produce the DSECT only.

If you omit the option, EFHMAPCR produces both.

The output from EFHMAPCR is as defined in "Output from assembling a map" on page 94.

Sample EFHMAPCR commands

```
EFHMAPCR MYMAP
```

This command assembles the file MYMAP ASSEMBLE, producing both the physical map and the DSECT.

```
EFHMAPCR PCMAP ASHTYPE B (DSECT
```

This command assembles the file PCMAP ASHTYPE, residing on the CMS B-disk, and produces only the DSECT. Note that the filetype ASHTYPE implies that EFHMAPCR will use the H assembler. If you tried to use this command on a VM/SP system running with the F assembler, it would fail.

Translating and compiling programs

The CICS/CMS EXEC that translates and compiles programs is called EFHTC. You can translate and compile by entering the `EFHTC` command alongside the name of a program file on any CMS `FILELIST` display, or by using it as a stand-alone command in CMS.

Using EFHTC on a file list can be very useful when you want to translate and compile a number of programs using the same options. You simply enter your EFHTC command alongside the first program file in the list, then enter = beside those that follow it, then press ENTER.

The full form of the EFHTC command is:

```
EFHTC fn ft fm (toptions...!coptions...
```

or, from a file list:

```
EFHTC / (toptions...!coptions...
```

where:

`fn` is the filename of the file containing the program to be translated and compiled.

`ft` is the filetype. This can be:

COBxxxxx for COBOL and COBOL II programs (see note below)

PLIxxxxx for PL/I programs

ASxxxxxx for assembler programs.

If you leave it out, the EXEC assumes COBOL.

Note: If your program is written in COBOL II, you can distinguish it from a COBOL program in one of two ways:

- You can use a filetype of COBOL2, which is reserved for COBOL II programs.
- You can use a filetype starting with the letters COB, and specify COBOL2 as your first translator option.

`fm` is the filemode of the file. If you leave it out, the EXEC searches the disks to which you currently have access, starting with your A-disk, until it finds the file.

When you use EFHTC on a file list, `fn ft fm` are replaced by `/`, which tells EFHTC to use the file over which you enter the command.

`toptions` are translator options. These are as described in the *CICS/VS Application Programmer's Reference Manual.* Note, however, that you must use the abbreviated forms of any options having more than 8 characters. For example, you must specify OS instead of OPSEQUENCE.

You must separate the translator options from the file identifier options with an open parenthesis.

`coptions` are compiler options. They are separated from the translator options by an exclamation mark (`!`). You can leave blanks before and after the parenthesis, to make your command easier to read, but it's not necessary. To find out which compiler options you can use, you should look in the user manual for the language you are using.

If you leave out either the translator or compiler options, you will get the default options set in the EFHTC EXEC. These are also what you get when you translate/compile on panel EFH11, using PF5. The default options will be either those that IBM supplies with CICS/CMS, or those that your system administrator has set up for your installation. "Customizing CICS/CMS" on page 211 tells system administrators how to change EFHTC. The IBM-supplied defaults are described in the following sections:

COBOL "Default COBOL compiler options" on page 86

PL/I "Default PL/I compiler options" on page 86

COBOL II "Default COBOL II compiler options" on page 88.

The output from EFHTC is as defined in "Output from the translate phase," and "Output from the compilation phase" on page 97.

Sample EFHTC commands

```
EFHTC MYCOB
```

This command translates and compiles the program source file, MYCOB. CICS/CMS compiles the program with the COBOL compiler, and uses the default translator and compiler options.

```
EFHTC MYCOB2 COBFILE B (COBOL2 FLAG(W) ! BUF=8K
```

This command translates the file MYCOB2 COBFILE, which resides on the CMS B-disk, using the translator options, `COBOL2`, to specify a COBOL II program, and `FLAG(W)`, to get only warning and severe messages. The command then compiles the translated result, using the COBOL II compiler, with 8K buffers.

Translating programs

CICS/CMS also has an EXEC that just translates a program, without compiling it. The EXEC is called EFHT, and you can use it in the same way as EFHTC. The full form of the EFHT command is as defined for EFHTC earlier, but you cannot, of course, include any compiler options. The output from EFHT is as defined in "Output from the translate phase" on page 96.

Note: The principal output from EFHT is the file containing your translated program. This file has the same filename as your source program file, a filetype of TRANOUT, and a filemode of Z3. The result of the filemode is that, immediately after anything reads the file, it will be erased. To avoid this, you should use a CMS `RENAME` command to change the filemode to Z1 before reading the file.

You probably won't find any reason to use EFHT during CICS/CMS application development. There's little point in translating a program without compiling it. However, as we explain in "Transferring maps and programs to CICS/VS" on page 151, when you transfer programs from CICS/CMS to a CICS/DOS/VS system, you have to recompile them on that system, but you don't have to retranslate them. You could therefore run the source programs through EFHT, and transfer the translated programs to CICS/DOS/VS.

Converting CMS files to the CICS/CMS structure

There are two ways that you can use the file utility for converting CMS files, without using option 4 on panel EFH1. You can use both from the `CP/CMS` command line on panel EFH12, or from CMS.

You can display panel EFH14 directly by typing:

```
EFH14
```

You can then enter all the parameters you need on the panel, as described in "Converting local CMS files to CICS/CMS keyed files" on page 110. When you've

finished the conversion and you press PF3, you will go back to where you came from: either panel EFH12 or CMS.

You can execute the utility even more directly, avoiding the panel display, by entering the name of the program that the utility runs, followed by a list of the input parameters you require. If you're selecting records from the input file using a range of keys, you must include all the parameters. If you're selecting records by any other method, you can leave out the last two parameters.

The command you need to issue is:

```
EFHUCMS1 params
```

The `params`, in the order you must specify them, are:

1. The input filename.
2. The input filetype.
3. The input filemode.
4. The displacement of the key within each record in the input file. This is counted from byte zero; so, for example, if the key starts in the fourth byte, you must specify a displacement of 3.
5. The length of the key. This must be between 1 and 255.
6. The name of the output files. If you specify an asterisk (*), EFHUCMS1 will give the output files the same name as the input file.
7. The maximum length of records in the output files. If you specify a length that is less than the existing maximum length in the input file, CICS/CMS will ignore your specification, and use the input file's maximum record length. If you specify a record length for a file containing fixed-length records, EFHUCMS1 will ignore the specification, and use the input file's record length.
8. The number of records from the input file that you want EFHUCMS1 to convert.

   If you specify `0`, EFHUCMS1 will convert all records, starting from the record number you give in parameter 9.

   If you specify asterisk (*), you must provide a range of keys identifying the records to be converted (parameters 12 and 13).
9. The record at which you want EFHUCMS1 to start converting. The utility will ignore this option unless you have also specified parameter 8 as a number, including 0.
10. Whether you want trace output.

    If you specify `TR`, in upper or lower case, the utility will write trace output for the conversion to your screen. If you specify anything else (for example, `NOTR`), you won't get trace output.
11. Whether you want existing output files to be overwritten.

If you specify REP, and the utility finds that output files already exist with the same filename as your chosen output file, and a type of EFHVDATA or EFHVINDX, it will replace the existing files. If you specify anything else (for example, NOREP), and the conditions described above exist, the utility will stop, displaying an appropriate error message.

You only need the last two parameters if you have specified that you are going to supply a key range (*) in parameter 8.

12. First key value. This is a string of alphanumeric characters, enclosed in single quotes. If the key you want to specify contains a single quote, you must specify the quote as two single quotes. For example, if your key is JOHN'S, you must enter it in this command as 'JOHN''S'.

    If the value you give is shorter than your key length, EFHUCMS1 will treat it as a generic key.

13. Second key value, for which the rules are the same as for the first key value.

Both key values must be the same length, and you must specify them in the case in which they appear in the file. For example, if a key is key01 in the input file, and you specify KEY01 in the command, EFHUCMS1 will not find the key.

EFHUCMS1 looks at both keys to find which is the lower in EBCDIC terms. It then converts all input file records with keys that lie between the low key and the high key value. The records in your output data file will always appear in ascending key order.

You must include all options in the order given above, so it's probably better to run EFHUCMS1 using the EFH14 panel. If you want to execute it as a command, the best way is to write an EXEC, containing an EFHUCMS1 command that presets most of the parameters to the values you usually want, leaving only a few (such as the input filename), as variables. In this way, you can avoid having to remember the parameter order, and reduce the risk of error.

Sample EFHUCMS1 commands

```
EFHUCMS1 FILE DATA * 3 8 * 80 0 1 TR NOREP
```

This command converts the CMS file FILE DATA. EFHUCMS1 will search all currently accessed disks until it finds FILE DATA. The keys in FILE DATA start in the fourth byte of each record, and are 8 bytes long. The output from the conversion is a CICS/CMS pseudo-VSAM file, comprising two CMS files. These have the same filename as the input file, and filetypes of EFHVDATA and EFHVINDX. The maximum length of records in the output file is 80. EFHUCMS1 will convert all the records in FILE DATA, starting with the first. It will produce trace output. If either of the files FILE EFHVDATA or FILE EFHVINDX already exists, the utility will stop without creating the new files.

```
EFHUCMS1 OLDFILE DATA B 0 5 NEWFILE 132 * 0 NOTR REP 'KATE' 'DAVE'
```

This command converts the CMS file OLDFILE DATA, residing on the CMS B-disk. The keys in OLDFILE DATA start in the first byte of every record, and are 5 bytes long. The output from the conversion is a CICS/CMS pseudo-VSAM file, comprising two CMS files: NEWFILE EFHVDATA and NEWFILE EFHVINDX. The maximum length of records in the output data file is 132. The utility uses a generic key search to select records to convert. Because

DAVE is lower (in EBCDIC terms) than KATE, EFHUCMS1 will copy all records from the input file that have keys with values between DAVE and KATE. The utility won't produce trace output, and, if it finds existing files with the file IDs NEWFILE EFHVDATA or NEWFILE EFHVINDX, it will replace them with the output files it creates.

Preparing tables

CICS/CMS tables are CMS files. You can therefore create your tables using your editor, without displaying any CICS/CMS panels. You must remember, however, either to use the filename and filetype set up in EFHSETP or EFHPROF, or to use the parameter panel (EFH121), or PF9 (`Install`) on an EFH13 or EFH11 panel, before accessing a table from an executing application. "CICS/CMS tables" on page 66 tells you in detail how to define tables to CICS/CMS.

Starting CICS/CMS

If you've done all your preparation outside CICS/CMS, you won't need any of the options on CICS/CMS's initial panel (EFH1) except option 2, which takes you to the execution panel EFH12. You can avoid EFH1 completely, and go straight to the execution panel, by typing:

```
EFH12
```

instead of `CICSCMS` to start CICS/CMS. You can then work completely from panel EFH12. However, if you do this, you have no way of getting to panel EFH1 during the session. If you enter CICS/CMS using `EFH12`, pressing PF3 on panel EFH12 takes you straight to CMS, rather than to panel EFH1.

Chapter 13. Transferring tested applications to CICS/VS

Once you have tested an application as fully as you can on CICS/CMS, you need to try it on a test CICS/VS system before moving it into production.

At this stage, you will have to do some of the things involved in the "traditional" method of developing CICS applications. You may have to ask a system programmer to update tables or resource definitions, and set up any test files you need. You may also have to check with other application programmers to make sure that your tests won't interfere with theirs. You can then transfer your application files to the test CICS/VS system and finish your testing.

What do we mean in this context by "application files"?

The three main kinds of files involved are:

- The files containing your programs and maps
- Any data files that you've created during the CICS/CMS tests, and that you want to continue to use on your CICS/VS system.
- If you're transferring your application to a CICS/DOS/VS system, any DSECTs or copybooks that you've created for your application and that you've put in one of the CICS/CMS macro libraries (probably the EFHxUSER library)

The rest of this chapter deals with each of these in turn.

Transferring maps and programs to CICS/VS

Before transferring any application programs or BMS maps, you need to decide what form of these objects you need to transfer. Do you need to transfer your source map and program files and go through the stages of assembly, translation and compilation all over again?

The quick answer is "no, in ***most*** cases".

The rules governing this depend on the kind of programs or maps you've created.

For ***high level language programs***, if you are moving the programs to a CICS/OS/VS system, you just have to transfer your TEXT files and link-edit them.

However, the compilers that CICS/CMS uses for PL/I and COBOL are the OS/VS compilers. If you are transferring PL/I or COBOL programs to CICS/DOS/VS, therefore, you must transfer either the source programs, or the translated (but not compiled) forms of the programs. You then translate (if necessary), compile, and

link-edit the programs on the CICS/DOS/VS system. VSE does not support COBOL II.

To translate your source program files without compiling them, you run the CICS/CMS EFHT EXEC, as described in "Translating programs" on page 146.

For ***assembler programs***, you can transfer the TEXT files to any CICS/VS system.

For ***BMS maps***, if you are transferring the maps to a CICS/OS/VS 1.7 system, you can transfer the TEXT files. However, if you are transferring the maps to an earlier level of CICS/OS/VS or to a CICS/DOS/VS system, you must transfer the source maps, and reassemble them on your CICS/VS system.

Note: The BMS DSECTs you create using CICS/CMS are not stored as separate files, but as members of the appropriate EFHxUSER macro library. For information on transferring them to a CICS/VS system, see "Transferring macro library members to CICS/DOS/VS" on page 153.

For each file that you transfer to a CICS/VS system, you will need to wrap the file in appropriate JCL to tell the target system what to do with the file. For example, if you are transferring a COBOL program to CICS/DOS/VS running on a remote VSE system, you'll need JCL statements:

- To identify it as a VSE job
- To access the remote system libraries containing the compiler
- To access the library to which the result is to be written.

Your system administrator will probably supply standard files containing appropriate JCL. In the examples below, we assume that you keep your sample JCL files separate from your application object files, and transfer them to your remote system in a way that ensures that they arrive there as a single file.

The way you transfer your files depends on where your CICS/VS system is running:

1. As a guest of a VM machine in the same VM host system as you, or
2. On a remote system.

Transferring programs and maps to a guest CICS/VS system

To transfer program and map files to a CICS/VS system running as a guest on your host VM system, you use the commands:

```
SPOOL PUNCH TO userid CONT CLASS x
PUNCH pjcl JCL A (NOH
PUNCH fn ft A (NOH
PUNCH sjcl JCL A (NOH
SPOOL PUNCH CLOSE NOCONT OFF
```

The first SPOOL command specifies that anything that you send to your virtual punch should be transferred to the virtual card reader of `userid`. Here, `userid` must be the VM ID of the virtual machine on which the CICS/VS system is running. The CONT option ensures that the separate JCL and application object files will be punched as one. The CLASS option gives the class (x) of the reader for the batch partition of the CICS/VS system. Your system administrator will tell you what to use for x.

The PUNCH commands send the named files to the virtual punch. The first sends the JCL prefix file, pjcl; the second sends your application object file, fn ft; the third sends the JCL suffix file, sjcl. You must include NOH in each command to ensure that CMS doesn't put any header information on the files.

The final SPOOL command closes the punch, which sends the files. NOCONT returns the virtual punch to non-continuous operation.

Transferring programs and maps to a remote CICS/VS system

To transfer your program and map files to a CICS/VS system running on a remote MVS or VSE system, you use similar commands to those described above. Instead of spooling your punch to the VM userid where CICS/VS is running, you spool it to the VM userid that controls the RSCS network. The network then transfers the files to the remote system you've identified with a CP TAG command. The following commands show this in action:

```
TAG DEV PUNCH remsys
SPOOL PUNCH TO rscsid CONT CLASS x
PUNCH pjcl JCL A (NOH
PUNCH fn ft A (NOH
PUNCH sjcl JCL A (NOH
SPOOL PUNCH CLOSE NOCONT OFF
```

The TAG command specifies that punch files are to be transmitted to the remote system, remsys. Your system administrator will tell you what remote system ID to use.

The SPOOL command directs punch files to the userid of the virtual machine controlling the RSCS network (rscsid). The CONT option ensures that the separate JCL and application object files will be punched as one. The CLASS option gives the class (x) of the reader for the batch partition of the CICS/VS system. Your system administrator will tell you what to use for rscsid and x.

Finally, the PUNCH commands punch your file to the virtual punch, close the file, and then spool it to the virtual reader of the RSCS virtual machine. That machine then sends your file across the network to the remote system.

Transferring macro library members to CICS/DOS/VS

While developing applications using CICS/CMS, you'll probably create a number of macro library members. If you are transferring your application to a CICS/DOS/VS system, you will have to transfer the macro library members that your program(s) will need when you recompile. You may need to transfer:

- The DSECTs for your BMS maps, which CICS/CMS writes to one of the EFHxUSER files, as explained in "CICS/CMS macro libraries" on page 99.
- Copybooks containing common source code needed by applications, such as record definitions for data files.

You can send members of MACLIB files to a CICS/VS system using the commands given in "Transferring programs and maps to a guest CICS/VS system" on page 152 or "Transferring programs and maps to a remote CICS/VS system." However, you will need additional JCL to identify the macro library on the remote CICS/DOS/VS system.

You will also have to do some work on your CICS/CMS system to prepare the macro library members for transfer. You need to:

- Extract the individual members that you want to transfer from the macro libraries that contain them
- Remove the final record (containing / /) that CMS adds to macro library members.

The technique described below deals with both of these.

Preparing macro library members for transfer

1. From CMS, or the `CP/CMS Command` line of panel EFH12, type the command:

   ```
   MACLIST maclib
   ```

   where `maclib` is the filename of the macro library containing the members(s) that you want to prepare. MACLIST produces a file list of the individual members of a macro library. For example, if you want to list the members of the user COBOL macro library, type:

   ```
   MACLIST EFHCUSER
   ```

2. For each member that you want to transfer to CICS/DOS/VS, move the cursor alongside the member name, and invoke your editor.

3. From the editor's command line, enter the command to save the library member that you're editing as a new file with a specified filename and filetype. For example, using XEDIT, you'd use the command:

   ```
   FILE filename filetype
   ```

   You can use anything you like for `filename` and `filetype`, ***except*** the filename of the macro library from which you're extracting members, and the filetype MACLIB.

4. Return to CMS (or the EFH12 panel) by pressing PF3 on the file list of macro library members.

The result of this sequence, for each member for which you use it, is a file, on your A-disk, containing just the library member, without any control records. That's the file that you transfer to CICS/DOS/VS.

Transferring data files to CICS/VS

You can use the CICS/CMS general file transaction, CCU2, to convert CICS/CMS files to VSAM files, and to ship them to your CICS/VS system using the remote server. This is explained in "General file conversion" on page 113.

Don't forget that you'll have to make sure that the files you want to ship are defined in the file control table (FCT) of your CICS/VS system.

What's next?

In the next part of the book, we describe the help that CICS/CMS can give you in diagnosing and solving problems in your applications and in CICS/CMS itself. We also give some suggestions on the best ways of taking advantage of this help.

Part four—Diagnosing and solving problems

This Part of the book describes:

- The way CICS/CMS reports problems
- The tools available to help you solve problems
- The best ways of using the problem-solving tools
- What to do if you have problems with CICS/CMS itself.

Chapter 14. How CICS/CMS reports problems

In Chapter 9, "Testing an application," we told you how to execute your applications in CICS/CMS, and how to use CICS/VS tools to test different execution paths through those applications.

In this part of the book, we deal more directly with what happens when problems arise, whether in your applications, in your use of CICS/CMS, or in CICS/CMS itself.

This chapter describes, in general terms, the kinds of problems that can arise, and the way that CICS/CMS reports them, using its error handler. The next two chapters introduce the debugging aids available. We give a general guide to the main types of application problems, and suggest ways of dealing with each type, using the debugging aids.

Most of this information should be useful both to application programmers with a problem in their program, and to systems people trying to help them.

The final chapter, however, is aimed specifically at systems people. It tells them how to diagnose that a problem is being caused by CICS/CMS rather than by a user program, and what information they will need to gather to report the problem to IBM.

Note: This part of the book is intended to make you familiar with the kind of problem reporting you can expect from CICS/CMS, and to introduce the facilities that can help you diagnose problems from those error reports. It is ***not*** a general guide to the process of problem diagnosis. There are many other books dealing with this subject, and you will find a list of some of them in "Where to find more information" on page 196.

Messages you can receive when using CICS/CMS

As we explained in "Testing a CICS program" on page 18, apart from the time you spend in CMS, a typical terminal session using CICS/CMS can be divided into two parts:

1. The CICS/CMS session, which is everything you do between typing `CICSCMS` or `EFH12` and returning to CMS

2. The CICS test session, which is everything you do between leaving panel EFH12 to run a transaction or program, and returning to panel EFH12.

Messages during a CICS test session

During a CICS test session, the CICS/CMS error handler displays nearly all the messages. These are mainly:

- Warnings that you are using a CICS/CMS feature incorrectly
- Information messages, or warnings that you are using CICS/VS features that CICS/CMS does not support
- CICS/CMS versions of CICS/VS messages
- Abend messages.

The error handler displays these messages in a common form, as explained in "The CICS/CMS error handler display" on page 161.

There are a few messages that can appear during a CICS test session that are not displayed by the error handler. These are:

- Some CICS/VS messages associated with BMS paging and the field engineering transaction (CSFE).
- Some remote server abend codes.

Messages outside the CICS test session

Outside the CICS test session, but within a CICS/CMS session, most messages are written straight to your terminal without the intervention of the error handler. These messages are mainly of the three following types:

- Messages from CMS
- Messages from CICS/CMS EXECs
- Messages from CICS/CMS utilities

Messages from CMS

If a problem arises from CICS/CMS using a CMS facility, you will usually get at least two messages. The "standard" CMS message for reporting the problem, and a CICS/CMS message giving more detail.

If you use a CMS command incorrectly, CMS will report the error itself. This applies whether you are using CMS "on its own", that is, while CICS/CMS is not running, or from within CICS/CMS, either by issuing single commands from the execution and escape panels, or by entering the CMS subset from those panels.

In general, you'll probably find the CMS messages a useful addition to the CICS/CMS messages. However, if you find that you're getting the information you need from the CICS/CMS messages, and you want to reduce the amount of screen output, you can suppress the CMS messages by typing:

```
SET EMSG OFF
```

Messages from CICS/CMS EXECs

You may get messages from CICS/CMS EXECs warning you that you've used a feature incorrectly, reporting errors in executing an EXEC, and reporting on the progress of a running EXEC.

Messages from CICS/CMS utilities

You may get messages from the CICS/CMS utilities, EFHUCMS1, EFHUMAP, and EFHUSTG.

Messages written to CMS files

While using CICS/CMS, you will also get some messages that are written to CMS files. These are translator messages and compiler/assembler messages.

Translator messages

The CICS/CMS EFHTC and EFHT EXECs invoke the CICS/VS translator. The translator checks all your EXEC CICS commands for incorrect syntax. If it finds any errors, it gives a return code greater than 4, and EFHTC or EFHT displays a message at your terminal advising you to check the listing file. In the listing file, you will find self-explanatory translator messages describing the problem.

For further information on the listing files for the translator, see "Output from the translate phase" on page 96.

Compiler/assembler messages

Whatever source language you use, the CICS/CMS EFHTC EXEC calls the appropriate compiler or assembler, which checks your program for syntax and potential execution errors. If there are any errors, EFHTC displays messages at your terminal, advising you to check the listing file. If you need more information about the error messages you find there, you will need to consult the reference manual for the compiler or assembler you are using.

You'll find more about compilation error reports in "Output from the compilation phase" on page 97.

The CICS/CMS error handler display

All error handler displays have the general appearance shown in Figure 45 on page 162.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/04/86 10:33:14

Program Name: name                                   Line Number: nnnnnnnn

Message EFHccnns has been generated

After this screen the CICS environment will (CONTINUE/TERMINATE)

Up to 8 lines of extra information. This will normally include
further information about the error, suggestions about the
cause of the error, references to further information and,
if necessary, contents of applicable parts of storage,
registers, and so on.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 45. General form of error handler display

Error handler display fields

The fields in the EFH125 panel are described in the sections following.

"Program Name" line

The line beginning `Program Name` gives the name of the application program, if any, that was executing, and the line within that program (nnnnnnnn) at which the message was generated. If you've sequenced your program, as we advise in the individual compiler sections in "CICS/CMS support for high-level languages" on page 84, you can check this line number in your source program. Otherwise, you can look it up in your translator listing. If no application program was executing, the `Program Name` could be the name of one of the CICS/CMS modules. If the message is displayed while CICS/CMS is starting, or just before a user program starts, this line will say `Unknown` for both the program name and line number.

"Message" line

The line beginning `Message` gives the message code, and is the same format for all messages, as follows:

EFH identifies the message as coming from CICS/CMS.

cc identifies the CICS/CMS component that issued the message. For example, 09 identifies file Control, 10 identifies terminal Control.

You'll find a complete list of all the `cc` codes, and the components they represent, in the *CICS/CMS Messages and Codes* manual.

nn identifies the specific message code.

s is a severity code, indicating the seriousness of any problem, as follows:

I Information only.

Messages with this code usually tell you that some function is in progress or has ended, or that CICS/CMS has ignored a CICS/VS feature that it doesn't support. They have no effect on execution.

W Warning.

Messages with this code warn you of something that could lead to an error, but will not usually interrupt execution.

E Error.

Messages with this code report an error that ***may*** lead to other errors.

S Severe error.

Messages with this code usually follow an abend. Something has happened that makes it impossible for the current transaction or program to continue.

CICS environment status line

Immediately after the `Message` line comes a single line telling you what will happen if you press ENTER to try to carry on. In most cases, this line will say:

```
After this screen the CICS environment will CONTINUE
```

This means that CICS/CMS will at least try to continue the CICS test session.

For the most severe problems (usually those with a severity code of S), the line will say:

```
After this screen the CICS environment will TERMINATE
```

In this case, pressing ENTER will have the same effect as pressing PF12: you'll return to the execution panel (EFH12).

Explanation lines

After the CICS environment status line come the lines of explanation. In most cases, these will provide enough information for you to understand the message immediately, or to decide what other sources of information you will need. The display examples later in this section show the kind of information you can expect to find in these lines.

Function key meanings

The final lines of the display tell you what the function keys do on an error handler display, as follows:

ENTER If the CICS environment status line says that the CICS environment will continue, pressing ENTER will let your application carry on. If it says that the CICS environment will terminate, pressing ENTER will take you back to panel EFH12.

PF1 displays a help panel, giving you a brief description of panel EFH125, and of the functions of the PF and PA keys.

PF3 is equivalent to pressing ENTER.

PF5 lets you stop unwanted messages.

As we said earlier, CICS/CMS will display messages when you use features supported by CICS/VS, but not by CICS/CMS. If your application includes a loop that contains an unsupported command, you will keep getting the same message, telling you that a statement at a particular line number in the program is using that unsupported feature.

You can suppress this message by pressing PF5. This will suppress only message panels that are identical to the one on which it is pressed. Thus you can use PF5 to stop messages from a single program statement that is executed a number of times. However, you can't use it to stop messages from other statements using the same feature, because the message panels will include a different program line number.

PF6 redisplays the last screen before the error handler panel. This will usually be the last screen that your application displayed, and can be useful in tracing when your application raised the condition that generated the error message.

PF9 restarts the display of all the messages you've suppressed using PF5. You can do the same thing from the escape panel by pressing PF10.

PF12 gives an immediate return to panel EFH12. No matter what the CICS environment status line says, PF12 will always stop the current CICS test session.

PA2 suspends the CICS test session, and displays the escape panel (EFH122). This can be particularly useful if the message suggests that your application is about to abend. Pressing PA2 lets you suspend the application, and use appropriate debugging tools to investigate the problem.

For an assembler program, you might use CP or CMS debugging tools to check the contents of storage.

For a COBOL II program, you might use CEBR to check the temporary storage queues that the COBOL II debug facilities create.

For a PL/I program, you might look at the PL/I diagnostic information. This is usually written to SYSPRINT, so you'd need to

define SYSPRINT as a CMS file, using a CMS FILEDEF command, if you wanted to check it in this way.

For all programs, particularly COBOL programs, you'll probably get useful information by running the program again with trace on, and using the EFHUMAP command to find storage addresses relevant to your program, as described in "The EFHUMAP utility" on page 176.

Error handler displays for abends

The displays produced by the error handler for abends are identical in format to the message displays. The severity code will, of course, always be S, and the lines of additional information will always start with the abend code.

Example error handler panels

The next few pages contain some examples of CICS/CMS error handler panels.

Example information message

Figure 46 shows an information message, telling you that you've used the unsupported DELAY option in a PL/I program called EMILY.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/04/86 08:44:30

Program Name: EMILY                                  Line Number: EMI00030

Message EFH8808I has been generated

After this screen the CICS environment will CONTINUE

The requested Interval Control function (DELAY) is not supported by CICS/CMS.

The request is ignored.




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 46. Sample information message

The message severity is I (information only), and the explanatory lines of the message tell you that you've used an unsupported feature, and what that feature is (in this case, DELAY). The line number (EMI00030) is taken from columns 73 to 80 of the source program; it's the sequence number.

Example warning message

Figure 47 shows the warning message you would get if the control records of the files that made up a CICS/CMS keyed file got out of sequence. The program is a COBOL program.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/04/86 08:47:33

Program Name: ACCT02                                 Line Number:    310

Message EFH0922W has been generated

After this screen the CICS environment will CONTINUE

The date and time stamps in the control records of the EFHVDATA and
EFHVINDX files for ACCTIX   do not match.
The EFHVDATA date/time stamp is 02/04/8616:15:52
The EFHVINDX date/time stamp is 03/04/8616:15:52
You can correct this situation by editing either file from the ESCAPE panel.
If you do not do so, the condition DSIDERR will be returned.



Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 47. Sample warning message

As the message says, the date and/or time in the control record of the ACCTIX EFHVDATA file does not match that in the ACCTIX EFHVINDX file. It shows you the date and time stamps so that you can see what the difference is. It also tells you what you can do to correct the error, and what will happen if you continue the application without correcting it.

Note: Although you can follow the advice given in the message, and correct the file while your program is "in flight", we wouldn't recommend it. You shouldn't create or change CICS/CMS keyed files by any method other than the CICS/CMS file utilities, EFHUCMS1 and CCU2.

Example error message

Figure 48 on page 167 shows the error message displayed if a program's name doesn't match the filename of the CMS file containing it, or if its entry point is defined incorrectly in the program table.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/04/86 08:56:04

Program Name: Unknown                                Line Number: Unknown

Message EFH0420E has been generated

After this screen the CICS environment will CONTINUE

The TEXT of the PL/I program JENNY    has been successfully loaded
   but the address of the program cannot be determined.

The Program Table indicates that the external procedure
name is JENNY    but this name is not present in the LOAD MAP




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 48. Sample error message

Here, the `Program Name` is `Unknown`, because the problem arose before the application program JENNY had started. You can't deal with this problem within the CICS test session. You will need to stop the test session (by pressing PF12), and make sure that the program name matches the filename, or edit your program table to give the program name in the entry field. You can then start the CICS test session again.

Example abend message

Figure 49 on page 168 was produced by an abend. The severity code is S, and the CICS test session will terminate. If you press ENTER to continue, you will go straight back to panel EFH12.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/04/86 10:45:41

Program Name: NATALIE                                Line Number: NAT00050

Message EFH8635S has been generated

After this screen the CICS environment will TERMINATE

ABEND AEIV - An exception condition has arisen for which no EXEC CICS HANDLE
CONDITION command is active.
Condition raised      : LENGERR
Last executed command : READ




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Terminate   PA2=Escape
```

Figure 49. Sample abend error message

As the message says, the line in the program NATALIE with the sequence number `NAT00050` contains an `EXEC CICS READ` command. This has raised a LENGERR condition for which the program contains no `EXEC CICS HANDLE CONDITION` command. To solve this problem, you could either correct the `EXEC CICS READ` command to eliminate the condition, or include an `EXEC CICS HANDLE CONDITION` command, and some code, to handle the condition in the future.

The CICS/CMS error log

Every time that the error handler displays a message panel, it writes an entry to the CICS/CMS error log. This gives you a record of all the errors in a particular CICS/CMS session or CICS test session.

CICS/CMS writes the error log to a file with a filename, filetype, and filemode defined either in the EFHSETP EXEC or in EFHPROF EXEC. For example, to ask CICS/CMS to write the error log to the file SESSION ERRLOG on your Z-disk, you could include the following statements in your EFHPROF EXEC:

```
SETPARM 'LOGFN SESSION'
SETPARM 'LOGFT ERRLOG'
SETPARM 'LOGFM Z'
```

You can change these definitions immediately before you start a CICS test session, by pressing PF7 on panel EFH12 and using the `Additional Parameter` line on panel EFH121. You can stop CICS/CMS writing the error log at all by specifying a period (`.`) as the value for the "LOGxx" parameters.

The error log contains the start and end time of each CICS/CMS session, and, in between, entries from the error handler. Those entries give you:

- The program name and line number
- The message number

- The text of the message.

If you run CICS/CMS using the defaults supplied by IBM, it will append the error log to a file called CICSCMS ERRORLOG on your A-disk. This file will contain the error messages from all your CICS/CMS sessions, unless you erase or edit it.

Restricting the life of the error log

You can make your error log less permanent. You can restrict its life to:

- A terminal session
- A CICS/CMS session
- A CICS test session

Restricting the error log to a terminal session

If you are using CICS/CMS from a host-connected terminal, you can define the error log as being on your Z-disk, as we did in the example above. It will then be erased, along with everything else on the Z-disk, every time you log off.

You can do the same thing on a PC, but only if you haven't set up your Z-disk using the VM/PC configurator (VMPCCON). Disks defined by VMPCCON are not erased.

Restricting the error log to a CICS/CMS session

You can erase the error log automatically, at the start of every CICS/CMS session, by including a CMS `ERASE` command for the log file in your EFHPROF EXEC. From then on, every time you enter the `CICSCMS` command, CICS/CMS will erase the error log.

You can do the same thing explicitly, by selecting option 5 on panel EFH1 each time you start a CICS/CMS session. This erases a number of CICS/CMS files, including the error log.

Restricting the error log to a CICS test session

You can erase the error log before beginning a CICS test session, by executing an appropriate CMS `ERASE` command from the `CP/CMS Command` line of panel EFH12.

You can erase the error log in the middle of a CICS test session, by executing `ERASE` from the command line of the escape panel (EFH122).

Chapter 15. Debugging tools

How to use this chapter

CICS/CMS provides a wide range of debugging tools, from the familiar CICS/VS interactive tools, such as EDF, designed to help identify application program problems, to special forms of trace, designed to help systems people track down problems in CICS/CMS itself.

To help you find the information that you need on the debugging tools you're most likely to use, we've divided them into three categories, as follows:

- General.

  These are the tools that can be used by anyone involved in CICS programming; they can help you diagnose most application programming problems.

- Storage analysis.

  These are the extra tools you'll need when you can't identify a problem using the general tools. To use them, you'll need to know something about the structure of CICS/VS and its control blocks, and about programming in CMS.

- Systems.

  These are the tools that systems people might need to investigate problems in CICS/CMS itself, or problems in application programs that appear to cause problems in CICS/CMS.

You can use any of the tools available, if you think it will help you diagnose your problem. We'd advise, however, that you don't use the storage analysis or systems tools until you're sure you can't get what you want from the general tools.

General debugging tools

The general tools available with CICS/CMS are:

- The CICS/CMS error handler
- Program listings
- CICS/VS interactive tools

- CICS/CMS trace.

The CICS/CMS error handler

Your primary source of help from CICS/CMS is the error handler. It should give you enough information to identify immediately what's causing a problem, or at least to decide what other sources of information you might need.

"The CICS/CMS error handler display" on page 161 describes the error handler.

Program listings

For most application programming problems, you'll need to go back to your source program, and to the listings produced by your translations, compilations, and assemblies.

Some of the CICS/CMS debugging tools refer to program line numbers.

If you follow the recommendations about sequencing programs in the individual compiler descriptions in "CICS/CMS support for high-level languages" on page 84, you can relate the line numbers given in error message and EDF panels directly to your source program, without referring to listing files.

If you haven't sequenced your programs, or the CICS debugging tool refers to an application program storage address, you'll need to relate the information to your listing files to narrow down your search for the root of a problem. In particular, you will need to compare the line number that the error handler panels give you with your translator listing, to find the point in your program from which the error message was generated.

CICS/CMS writes all your listings to files on the Z-disk. The filetypes of these files are:

ASMLIST1 The listing from creating a BMS DSECT.

ASMLIST2 The listing from creating a physical map.

TRANLIST The listing from a translation. This is useful for COBOL, COBOL II, and PL/I programs. For assembler programs, the listing is written to the translator output file, filetype TRANOUT. However, this file is erased during the assembly phase, so you'll need to look in the assembly listing file.

lanLIST The listing from a compilation or assembly, where **lan** is COB (for COBOL or COBOL II), PLI (for PL/I), or ASM (for Assembler).

You can look at the listings using your editor. Alternatively, if you're used to dealing with printed listings, you can use the CMS `PRINT` command to print the files containing the listings.

CICS/VS interactive tools

CICS/CMS supports the CICS/VS interactive tools: EDF, CECI, and CEBR. "Testing tools" on page 125 tells you how to use them with CICS/CMS and Chapter 16, "How to diagnose common problems" on page 191 describes situations in which you'll find them useful.

CICS/CMS trace facilities

CICS/CMS provides trace facilities similar to those in CICS/VS. However, CICS/CMS trace is tailored to suit the requirements of people who are developing applications interactively.

All trace entries are written to a file, with a filename, filetype, and filemode defined in the EFHSETP EXEC, or in your EFHPROF EXEC. For example, if you want your trace written to the file MYTRAC TRACLOG on your Z-disk, you include the statements:

```
SETPARM 'TRACEFN MYTRAC'
SETPARM 'TRACEFT TRACLOG'
SETPARM 'TRACEFM Z'
```

in your EFHPROF EXEC.

CICS/CMS trace entries are formatted in a way that makes them easy to read on a screen. They contain the same information as in CICS/VS trace, but with some additional annotation, as follows:

- The function name and first option of every EXEC CICS command that an application issues.
- The line number in your application at which each EXEC CICS command is issued.
- The EIBRESP value for each EXEC CICS command executed.
- The characters >>> before each EXEC CICS command to help you identify the point at which your application issues the command.
- The characters < >, if an EXEC CICS command finishes with a return condition other than NORMAL. This makes it easy for you to scan the trace log for possible problems.
- A symbolic representation of the current nesting level of your CICS test session, written at the end of each entry. A slash (/) represents the level of transaction; an asterisk (*) represents the level of program. For example, when you start to run a transaction, a single slash will appear at the end of each trace entry. As soon as the first program associated with the transaction starts, /* will appear at the end of each entry. If you then go to the escape panel (perhaps to start EDF from within your program), you will move to another level of nested transaction, and /*/ will appear at the end of each entry.
- Optionally, output from the EFHUSTG utility, which gives the addresses of modules currently in storage. This is a storage analysis utility, described in "The EFHUSTG log" on page 177.

Figure 50 shows a typical piece of trace output, part of the trace of an execution of the Primer's sample application.

```
FC000500 TC   01000100 00092D34                    50139D8C
CC080200 TCAI C1C3C3E3 00010001 ACCT               70136538  /
EA000300 TMP  01000100 000910F5        5 ACCT      4013659A  /
EA000500 TMP  01000100 00092D34                    400C1178  /
CC1A0200 PCP  82000101 00000000 b        ACCT00    70133EAC  /
F2820400 PC   00000000 00000000          ACCT00    6013660E  /
F1CC0400 STG  000007E8 0108F04C    Y  0<           400BFC38  /
C8000400 ++++ 00095000 8C0007F8   &    8          500BE512  /
EA000300 TMP  01000300 000914D8         Q ACCT00   4013442E  /
EA000500 TMP  01000300 00092CE4         U          400C1178  /
CC1B0200 PCP  00092CE4 00000000    U     ACCT00    701343E0  /
F1C10400 STG  00000068 0108F04C       0<           40136626  /
C8000400 ++++ 00095800 81000070     a              500BE512  /
F1CA0400 STG  00200DE3 0108F04C    T  0<           50136648  /
C8000400 ++++ 00096000 8A040DE8   -    Y           500BE512  /
F3110300 IC   1100C120 00000000   A                5013670C  /
F3000500 IC   00000000 00000000                    40132CB4  /
CC0A0200 TCAI C1C3C3E3 00010001 ACCT     ACCT00    7013674C  /
F1CC0400 STG  00000510 0108F04C           0<           400BFC38  /
C8000400 ++++ 00096DF0 8C000518                   500BE512  /
CC1A0200 PCP  0100C120 00000000          ACCT00    70133EAC  /
F2010400 PC   00000000 00000000          ACCT00    5013678A  /
F1CC0400 STG  00000400 0108F04C       0<           400BFC38  /
C8000400 ++++ 00097310 8C000408                    500BE512  /
EA000300 TMP  01000300 000970B8          ACCT00    4013442E  /
EA000500 TMP  01000300 00092CE4        U           400C1178  /
CC1C0200 PCP  000201E8 00000001    Y     ACCT00    70134D74  /
F1930400 STG  0009053A 0108F04C           0<           50134ED8  /
C8000400 ++++ 00097800 93090540     1              500BE512  /
CC180200 PCP  FD096E00 04020532   >      ACCT00    70134616  /*
E1000400 EI++ 00008160 00001804   a-         14     500207A4  /*
>>>    14      EXEC CICS SEND         MAP
F1400400 STG  000908C0 0108F04C       0<           500D0B10  /*
C9000400 ---- 000908C0 850000E8     e  Y           500BE54E  /*
```

Figure 50. Sample CICS/CMS trace output

From the trace output you can see that, as the transaction ACCT starts, a single slash (/) appears at the end of the line. When program ACCT00 starts, an asterisk (*) appears beside it. The line immediately following the start of ACCT00, with >>> at the front, shows the start of the first command, `EXEC CICS SEND MAP`. The 14 in the command line is the line number in the program which issued the command.

Appendix E, "CICS/CMS trace entries" on page 253, describes the difference between the CICS/VS entries you'll find in a trace log, and the CICS/CMS entries. The appendix describes the CICS/CMS entries in detail. For more information on CICS/VS entries in trace output, and on interpreting trace output in general, look in the *CICS/VS Problem Determination Guide.*

There are two ways of asking CICS/CMS to produce trace output, as follows:

- Through your EFHPROF EXEC file
- Through the parameter definition panel (EFH121).

Requesting trace through your EFHPROF EXEC file

You can use your EFHPROF EXEC to ask for trace to be enabled by setting the TRACE parameter to YES, as follows:

```
SETPARM 'TRACE YES'
```

Trace will then be enabled for every CICS test session. Remember to clean up your trace log from time to time. The log can quickly use up your available disk space, so you should always get rid of unwanted trace, either by editing the log, or by erasing the entire file. You can erase the entire file automatically from EFHPROF by including a CMS ERASE command.

Since the trace file can take up a lot of space, you might like to consider putting it on your Z-disk. You can request this in your EFHPROF EXEC, as follows:

1. Use the TRACEFM parameter to define your trace log filemode as Z.

2. Unless you are using VM/PC, and have set up your Z-disk using VMPCCON, edit into the EXEC the command:

   ```
   EXEC EFHTSPAC
   ```

Whenever EFHPROF is executed, it will then run EFHTSPAC. If there isn't already a Z-disk, EFHTSPAC will set one up and format it. If there is one, it will do nothing. All trace output from then on will be written to the Z-disk.

Requesting trace through the EFH121 panel

If you want to be able to turn trace on only for selected CICS test sessions, set the TRACE parameter to NO in EFHPROF EXEC, as follows:

```
SETPARM 'TRACE NO'
```

You can then turn trace on for a particular test session by pressing PF7 on the execution panel (EFH12). This will display panel EFH121 (shown in "Changing your CICS/CMS environment within a session" on page 119). One of the lines on this panel looks like this:

```
Trace          ===> No          Yes,No
```

If you change the first No to yes, and press ENTER, you will turn trace on. You can then:

1. Return to EFH12 by pressing PF3

2. Erase any existing trace file from the CP/CMS Command line

3. Run your application.

When the application finishes, you can switch off trace from panel EFH121. You will then have a trace file that contains information only for that CICS test session.

The PF7 option is also available on the escape panel (EFH122). This means that you can display panel EFH1221, and turn trace on for ***part*** of a CICS test session. At some point before you want to start trace, when the application that you're testing is waiting for input, press PA2. This will display the escape panel. Press PF7 to display panel EFH1221. Turn trace on, by entering yes in the trace line.

Press PF3, and you will return to the escape panel. Pressing PF3 there returns control to your application, which will write trace entries from then on.

Storage analysis tools

The storage analysis tools available with CICS/CMS are:

- EFHUMAP
- EFHUSTG
- CP/CMS debugging tools

The EFHUMAP utility

EFHUMAP is a CICS/CMS utility that provides the sort of information you would get in a dump from a CICS/VS system. It gives you the addresses of CICS/CMS modules and control blocks. EFHUMAP is most useful when you get an error or abend message from CICS/CMS, and you need to find out where in storage to start investigating the problem. You can use EFHUMAP to get the address of the control block or module from which you want to start your investigations, and then use CP commands, such as DISPLAY, to display or dump the relevant storage.

Example output from EFHUMAP

Figure 51 shows a short piece of EFHUMAP output, which we produced during an execution of the Primer's sample application.

```
*DFHEDFDL  000F67A8  09/30    10.43  *PIDLVL*0110I
*DFHFEP    00102120  09/30    12.47  *PIDLVL*0110I
*DFHEDFCX  001037F0  09/30    10.39  *PIDLVL*0110I
*DFHEBF    001045D8  09/30    10.14  *PIDLVL*0110I
*DFHBFP    00104620  09/30    10.01  *PIDLVL*0110I
*DFHPHP    00105CA8  09/30    12.59  *PIDLVL*0110I
*DFHALP    001063E0  09/30    09.59  *PIDLVL*0110I

LOAD Module:
DFHTRP        000095F4

Addresses of component modules:


Global control areas
CSA           00008160
CSAOPFL       0000D200
PAM           0000C300
EFHCOMMA      00008C68



-------------  Transaction CICS -----------------
TCA           0013B5D8
```

Figure 51. Sample EFHUMAP output

Description of EFHUMAP output

Most of the output from EFHUMAP gives the addresses where the modules that make up CICS/CMS are stored. In Figure 51, we have shown the last few lines of that part of the output. These are the lines starting *DFH.

The lines following Global control areas give the addresses of CICS/CMS control blocks, such as the common system area (CSA).

The last lines of EFHUMAP output relate to the areas associated with the current application. In the example above, there's only one: the address of the task control area (TCA) for the current CICS test session.

Requesting EFHUMAP output

You ask for EFHUMAP output by entering EFHUMAP as a command in the CP/CMS command line on panel EFH12 or panel EFH122. If you enter EFHUMAP on its own, you get a list of all CICS/CMS module and control block addresses. You can, however, limit the output to just those addresses in which you're interested, by providing an appropriate string as a parameter to EFHUMAP. For example:

```
EFHUMAP *EFH
```

will list the addresses of only the CICS/CMS (EFH) modules.

```
EFHUMAP /*
```

will list the addresses of only the current dynamic storage areas.

EFHUMAP writes its output to two places:

1. Your screen. This gives you the chance to find something out immediately.
2. The file CICSCMS MAPLOG on your A-disk. This gives you a more permanent record of the information, which you can look at with your editor, or print if you want hard copy.

You will get information on dynamic area addresses from EFHUMAP only if you use it within the CICS environment. If you run EFHUMAP after you have returned to the execution panel (EFH12), you will only get module addresses.

The EFHUSTG log

EFHUSTG is an addition to the CICS/CMS trace facilities. It's a way of finding out how an application is using storage. It does this by writing a log of storage allocated, with a reference to the line in the trace table that tells you what allocated that storage. Each time CICS/CMS allocates a piece of storage, it makes an entry in the storage log. When it frees that piece of storage, it deletes the entry.

Requesting EFHUSTG output

You can list the current contents of the EFHUSTG log by typing:

```
EFHUSTG
```

in the CP/CMS command line on panel EFH12 or panel EFH122. The storage log is then written to your trace log file.

Note: You ***must*** have trace running to use EFHUSTG. If you run an application without trace, CICS/CMS won't create the storage log.

Controlling the number of entries in the EFHUSTG log

You set the maximum number of entries to be put in the EFHUSTG log in the STGTABLE parameter in EFHSETP or EFHPROF, or from panel EFH121. If that number proves to be inadequate, you will get a message from CICS/CMS like the one in Figure 52.

```
EFH125                    ERROR HANDLER FUNCTIONS    03/07/86 12:42:32

Program Name: Unknown                                Line Number: Unknown

Message EFH1484E has been generated

After this screen the CICS environment will CONTINUE

Insufficient storage to record more EFHUSTG data
EFHUSTG utility output frozen for rest of run

If more output required increase STGTABLE parameter and re-run




Press ENTER to Resume

PF1=Help          PF3=End          PF5=Suppress-Msg      PF6=Redisplay-Screen
PF9=Reset-Msgs    PF12=Quit        PA2=Escape
```

Figure 52. Error handler display for frozen EFHUSTG log

If you decide to continue with the run, you won't get any more output from any EFHUSTG commands that you issue. As the error handler message says, the EFHUSTG log is frozen at the point where it ran out of space.

Example output from EFHUSTG

Figure 53 on page 179 shows some EFHUSTG output, produced during execution of the Primer's sample application.

```
CC390200 CCMS 00130DB8 02404040          RCV CONS 5013077A  /*
CC390200 CCMS 000A380C 6E404000     >    RCV CONS 5013077A  /*
F1CC0400 STG  00000870 0108104C        <          400C3178  /*
C8000400 ++++ 000A6000 8C000878   -               400C3178  /*
CC240200 CCMS 00000000 00000000          EFHESCAP 601308E8  /*

======      Outstanding allocated Storage          ======
              Address  Record#  Length           Call addr
Storage       00089850    00040 8C800518          400C3178  /*
Storage       00088B10    00058 8C800108          600D4948  /*
Storage       00086750    00049 93800050          500C5180  /*
Storage       0009B800    00093 8C800518          400C3178  /*
Storage       0009C000    00067 93801340          5012DDB6  /*
Storage       0009D800    00097 8C800518          400C3178  /*
Storage       0009DD20    00101 8C800408          400C3178  /*
Storage       0009E130    00110 8C800518          400C3178  /*
Storage       0009E800    00128 8C800518          400C3178  /*
Storage       00087070    00119 81800070          4012F4E4  /*
Storage       0009F000    00121 8A800DE8          5012F506  /*
Storage       000A1E00    00143 8C800518          400C3178  /*
Storage       000A0200    00137 8C801BF8          600D4082  /*
Storage       000A2800    00180 93800D90          400C15C4  /*
Storage       000867A0    00169 98800020          400DA066  /*
Storage       000A4800    00267 8C8007A8          400C15C4  /*
Storage       00089580    00198 93800080          400C15C4  /*
Storage       000894E0    00189 978000A0          400D8768  /*
Storage       000A3800    00310 85800F18          5013077A  /*
Storage       0009FDF0    00304 8C800408          400C3178  /*
Storage       000A6000    00314 8C800878          400C3178  /*
Storage       000A5800    00250 978007B0          400D8768  /*
CC380200 CCMS 000A380B 00C2001D      B   SND CONS 50130938  /*
CC390200 CCMS 0009FFFA 06404040          RCV CONS 50130938  /*
CC390200 CCMS 000A380C 7D404040     '    RCV CONS 50130938  /*
```

Figure 53. Sample EFHUSTG output

Description of EFHUSTG output

You can find output from EFHUSTG in your trace log by looking for a line that says `Outstanding allocated Storage`. This denotes the start of the EFHUSTG output, which is written into the trace table at the point at which you issued the `EFHUSTG` command. In Figure 53 the line immediately above the EFHUSTG output contains `EFHESCAP`, indicating that the `EFHUSTG` command was executed from the escape panel.

The entries in each line of EFHUSTG output are:

1. The `Address` field.

   The address of the start of a storage allocation.

2. The `Record#` field.

   The line number in the trace table that shows what it was that allocated the storage. If the trace log contains information from a previous CICS test session, or if you print the trace log file, you will need to refer to the trace log line numbers to relate the `Record#` field to a line in the trace log. These line numbers start in column 105 in the log (not shown in Figure 53). If the trace log contains information from your current test session only, and you're checking it with your editor, you should be able to use the editor's line numbers to interpret `Record#`.

3. The `Length` field.

The first byte tells you the class of storage allocated. You can ignore the second byte. The last two bytes give you the size of the storage allocation.

The possible values of the first byte, and the classes of storage they represent are as follows:

1-byte code	Storage class
81	DCA
83	ISC TIOA
85	TIOA
8A	TCA
8C	User
8D	TRANSDATA
8E	TEMPSTRG
8F	File
93	Shared
97	TSMAIN
98	TSTABLE
9B	JCA

You can find out more about storage classes in the *CICS/VS Problem Determination Guide.*

4. The `Call addr` field.

   The last three bytes contain what is probably the most important information. They give you the address from which the call to storage control was made. You can use EFHUMAP, as described in "The EFHUMAP utility" on page 176, to associate this address with a CICS/CMS module.

CP/CMS debugging tools

The main strength of CICS/CMS as a debugging tool is the way that it lets you combine CICS/VS facilities with CP and CMS facilities to identify problems.

Whenever you hit a problem that you can't solve using the CICS/VS tools, you can use the full range of CMS and CP debugging commands and techniques to examine your transaction closely while it is running.

You can interrupt the execution to examine and change general registers, storage areas, or objects such as the program status word (PSW), and then continue execution. You can use trace facilities beyond those of CICS/CMS trace, so you can see where branches are being taken and when supervisor calls or I/O interruptions occur. By interrupting a test with the CICS/CMS escape facility, you can use these CMS and CP facilities exactly where and when they are needed.

The principal debugging facilities available are:

- DEBUG
- PER
- CP debugging commands.

CMS DEBUG

When a transaction abends, you will get the CICS/CMS error handler display. Often, this will provide enough information for you to identify the cause of the abend without further information. Sometimes, however, you will need to inspect the transaction's storage areas and registers to help identify the problem.

CMS DEBUG lets you look at selected storage areas, the contents of registers and the PSW.

You can invoke DEBUG in one of the following ways:

1. If you want to use DEBUG for all your programs, you can put:

   ```
   SETPARM 'USER DEBUG'
   ```

   in your EFHPROF EXEC. DEBUG will then be invoked:

   a. When CICS is initialized at the start of each CICS test session

   b. When an application program is loaded

   c. When a BMS map is first loaded.

2. If you want to use DEBUG for a particular CICS/CMS session, you can set the line that says `DEBUG program` in panel EFH122 to `DEBUG`. DEBUG will be invoked at the times described above.

3. If you want to use DEBUG to help diagnose the cause of a problem (such as an abend) when it occurs, you can press PA2 for the escape panel (EFH122) when you get the abend message. You then enter the `DEBUG` command on the panel's `CP/CMS Command` line to invoke DEBUG.

4. If you are running with the CICS/CMS SPIE option turned off, you'll get a CMS message if a program check occurs. At that point, you can use the `DEBUG` command.

Every time that DEBUG is invoked, you can use its subcommands to examine storage areas. If you use the `DEBUG` command after a program check (as described in 4 above), you can also inspect registers and the contents of the PSW.

If you're testing a program that you know to have problems, you can ask for DEBUG using either of the first two methods above. You can then enter DEBUG subcommands before the program starts, to stop execution at specified instructions. When the program stops at the specified points, you can use DEBUG subcommands to examine storage areas.

You'll find a general description of DEBUG in the *CMS User's Guide*, and detailed information on how to use it in the *VM/SP System Programmer's Guide*.

PER

`PER` is a CP command that lets you monitor what happens while you are running a program. You can monitor:

- The execution of specific instructions
- The execution of successful branch instructions

- The execution of all instructions that change the contents of specified registers
- The execution of any instruction that changes a specified piece of virtual storage.

You can find out more about PER, and see some examples of using it, in the *CP Command Reference for General Users.*

CP debugging commands

The CP DISPLAY command lets you check the contents of registers and individual storage locations. You can ask for EBCDIC translations of any display.

Other CP debugging commands let you set an instruction address in your program at which it will stop (ADSTOP), and change the contents of registers or storage locations (STORE).

You'll find examples of using these commands in the *CMS User's Guide* and in the *CP Command Reference for General Users.*

Systems debugging tools

The debugging tools that are available with CICS/CMS for systems people are:

- CP/CMS trace facilities
- CP/CMS dump facilities
- CICS/CMS terminal trace
- Field engineering trace
- Remote server trace.

CP/CMS trace facilities

As we explained earlier, CICS/CMS trace is a good way of diagnosing errors in a transaction. Sometimes, however, CICS/CMS trace is not enough. You need to be able to check the activity of the program at a more detailed level.

There are two forms of trace to help you do this:

1. CP trace
2. SVC trace.

CP trace

The CP trace facilities trace the following:

- Instructions
- Branches
- Interrupts (including program, external, I/O, and supervisor call (SVC) interrupts)
- I/O and channel activity.

Using options on the CP TRACE command, you can direct the trace output to a virtual printer (rather than to the screen), trace particular kinds of instruction, and trace without interrupting execution. Take care, however, when you use CP trace. It slows execution, and can produce a lot of output. It's best to use trace options to restrict the output to what interests you most. For example, you could trace branches only.

You can use the TRACE command from the CP/CMS Command line of either panel EFH12 (to get trace for a complete CICS test session), or panel EFH122 (to get trace for part of a test session).

You'll find guidelines for using CP trace in the *CP Command Reference for General Users*.

SVC trace

The CMS SVCTRACE command provides detailed information about every supervisor call (SVC) executed, either by your program, or by CICS/CMS on behalf of your program. The information includes:

- Register contents before and after the SVC
- The name of the called routine and the location from where it was called
- The contents of the parameter list passed to the SVC.

You can use the SVCTRACE command from the CP/CMS Command line of either panel EFH12 (to get trace for a complete CICS test session), or panel EFH122 (to get trace for part of a test session).

You'll find guidelines for using the SVCTRACE command in the *CMS User's Guide*.

CP/CMS dump facilities

Using a combination of CICS/CMS and CP/CMS facilities, you should be able to diagnose any problem interactively. As a last resort, you may want to dump all or part of your virtual memory.

There are two ways of getting a dump: the CP DUMP command, and the DUMP subcommand of CMS DEBUG. Both enable you to dump all or part of your virtual storage. The CP DUMP command has the added facility of letting you ask for an EBCDIC translation of your hexadecimal dump.

You'll find descriptions of the CP DUMP command and the CMS DUMP subcommand of DEBUG in the *CP Command Reference for General Users* and the *CMS Command and Macro Reference* manual respectively.

Note: You may also be able to use VMDUMP, as described in the *VM/SP System Programmer's Guide*. However, you can only do so if your VM system has the appropriate system extension installed.

CICS/CMS terminal trace

If there appears to be a problem in the way data is getting to and from a terminal or PC, you might find the CICS/CMS terminal trace feature useful.

You turn it on by setting the TERMTRAC keyword to YES in EFHSETP or EFHPROF, as described in "Terminal control parameters" on page 244. This will give you trace information on the terminal data stream.

The trace output is written to a CMS file, which you can define in EFHPROF or EFHSETP using the TCLOGFN, TCLOGFT, and TCLOGFM parameters.

Figure 54 shows some typical terminal trace output. We produced it by running the sample application, and attempting to display two nonexistent records.

```
******** CICS/CMS Terminal Control Log Started at 11:16:47 on 03/04/86 *********
CONS  B
CONS  B E&     0ACCOUNT FILE: MENU BT 0TO SEARCH BY NAME, ENTER: C; 0ONLY SURNAM
CONS  B
CONS
CONS '¢3 ¢ND ¢S79998
CONS  C ¢S  ¢/ H K1NO RECORD OF THIS ACCOUNT NUMBER
CONS  B
CONS
CONS '¢3 ¢S79999
CONS  C ¢S  ¢/ H K1NO RECORD OF THIS ACCOUNT NUMBER
```

Figure 54. Sample output from terminal trace

There are limitations on the usefulness of the terminal trace log in its "raw" form, as follows:

- We've shown the sample output in EBCDIC, for simplicity. However, you can only see most of the data in a "raw" terminal trace log if you display it in hexadecimal mode. In XEDIT, you can display data in hexadecimal by issuing the HEXTYPE macro, or using the SET VERIFY ON HEX subcommand with suitable parameters to identify the columns you want to see in hexadecimal.

- Records in the terminal trace log can be up to 1992 characters long. It's therefore impractical to try to print the log in its raw form.

To help you get more out of the trace log, and, in particular, to enable you to print it, we've provided the EFHTLOGT EXEC, and the EFHTLOGX XEDIT macro, both of which translate the terminal trace log into a more useful form.

Using the EFHTLOGT EXEC

The EFHTLOGT EXEC converts records from a terminal trace log and copies the converted records to a CMS file. It will give you EBCDIC translations of the hexadecimal records, and try to translate the terminal trace log data into BMS macros.

You can execute EFHTLOGT from within a CICS/CMS session, on the CP/CMS command line of panels EFH12 and EFH122, or outside CICS/CMS as a CMS command. If you type just:

```
EFHTLOGT
```

EFHTLOGT will convert the whole of the file defined by the TCLOGFx parameters using the default options defined below.

The complete form of the EFHTLOGT command is:

```
EFHTLOGT fn ft fm ( options ! targets
```

where:

fn is the filename of the file to be processed. The default is CICSCMS.

ft is the filetype of the file to be processed. The default is TERMLOG.

fm is the filemode of the file to be processed. The default is *.

Note: To use the full default fileid (CICSCMS TERMLOG *), simply invoke EFHTLOGT with no parameters before the bracket that separates the options.

The options are as described below:

Note: In the option descriptions, we've shown the minimum entry needed for each option in upper-case.

Option	Meaning
Bms/NOBms	If you specify BMS (or B), EFHTLOGT will try to translate the data into a set of BMS macro statements. This will be useful when you're trying to analyze a 3270 data stream involving BMS. If you omit the option, or specify NOBMS (or NOB), EFHTLOGT won't try to translate the data into BMS statements.
File/Display	Defines the destination of the EFHTLOGT output. If you omit the option, or specify File (or F), EFHTLOGT will write its output to a file, with the same filename and filemode as the input file, and with a filetype consisting of the filetype of the input file, prefixed with a $ (dollar sign). If, for example, you run EFHTLOGT using the default terminal trace log file, it will produce the output file, CICSCMS $TERMLOG A1. If you specify DISPLAY (or D), EFHTLOGT will write the output to your screen.

Innnn Gives the length of data that EFHTLOGT will process. The majority of records in terminal trace log output are up to 2000 bytes long, so the default, I2048, will cope with the longest records. You'd only use this option, therefore, to limit the scope of EFHTLOGT. For example, you might specify I80, to process only those parts of the log file that you can see on a terminal screen.

Split/NOSplit If you include the SPLIT option, EFHTLOGT will divide the results of the hexadecimal and character display so that character data will appear on a separate line from the hexadecimal (3270 controls) data.

If you leave the option out, or specify NOSPLIT (or NOS), EFHTLOGT will put character and decimal data together in the output lines, separated by the characters defined by the L and R options (see below).

Onnn Defines the length of data written to the output file. The default, O80, is most suitable when you use the DISPLAY option to write EFHTLOGT output to your screen. If you use the FILE option, you may want something different. For example, if you ask EFHTLOGT to write its output to a file that you intend to print, O132 might be a suitable option.

Lx Defines the character that EFHTLOGT is to put on the left-hand side of hexadecimal data to separate it from the character data when you run with the NOSPLIT option. `x` can be any character except `!` (exclamation mark), and defaults to <.

Rx Defines the character that EFHTLOGT is to put on the right-hand side of hexadecimal data to separate it from the character data when you run with the NOSPLIT option. `x` can be any character except `!` (exclamation mark), and defaults to >.

Note: The default options are set in a file called EFHTLOGX PROFILE. If you want to use defaults of your own choice, you must edit that file.

The `targets` define the number of lines from the terminal trace log that you want EFHTLOGT to process, as described below.

Note: You must separate the `targets` from the `options` with an exclamation mark (`!`).

Target **Meaning**

from identifies the first line to be processed. The default is the first line.

The `from` value can be:

- An absolute line number (:n). Conversion will start at the line number you give as `n`.
- A relative line number (+n). In this context, this is exactly the same as using :n.
- A `*`, to process the file from the first line up to and including the line you specify in the `to` target.

to identifies the number of lines, or the last line to be processed.

The `to` value can be:

- An absolute line number (:n). The line number you give will be the last converted.
- A relative line number (+n). Conversion will start at the line number you give for the `from` target, and continue through `n` lines.
- A `*`, to process the file from the line you give in the `from` target through the rest of the file.

If you use the `to` target, you must also use the `from` target.

Processing terminal trace logs from within XEDIT

You can use the EFHTLOGX XEDIT macro to interpret the CICSCMS terminal log file from within XEDIT. You type it on the XEDIT command line.

The format of `EFHTLOGX` is:

```
EFHTLOGX from to ( options
```

where the target values (`from` and `to`), and the options are mostly the same as we described above for EFHTLOGT. However, EFHTLOGX uses the target values in a different way from EFHTLOGT.

Once you've used the the XEDIT `/` line command to set the first line that you want to convert as the current line, you can use the following targets:

- If you type `EFHTLOGX` with a single target value in the form `:n`, it will convert `n` records, starting with the current line.
- If you type `EFHTLOGX` with a single target value in the form `+n`, it will convert all records from the current line up to and including the `n`th line.
- If you type `EFHTLOGX` with no target values, it will convert the current line only.

Note: When you use EFHTLOGX, it writes the output to your screen by default. If you want it to write to a file, you have to specify the FILE option.

Field engineering trace

The CSFE transaction provides trace information to help system programmers and IBM field engineers to diagnose both hardware and software problems.

CICS/CMS runs with CSFE trace enabled, so that, any time that you want to produce information using its facilities, you can do so. There are three CSFE functions that apply to CICS/CMS, each requested through CSFE's `DEBUG` option:

1. Storage freeze
2. Storage violation

3. Global trap/trace exit.

Storage freeze

As a transaction runs, CICS/CMS allocates storage from various subpools. To make the best use of the storage available, it will reuse storage if it can. By requesting a storage freeze, you can stop CICS/CMS reusing storage. If your application abends, you'll then have a complete record, in the trace log, of all the storage the transaction used. You can then escape from the error handler abend panel, and use EFHUSTG from panel EFH122 to check the storage. All storage is freed at the end of the transaction, whether you've asked for storage freeze or not. You'll therefore always need to check the storage before returning to panel EFH12.

Note: You request a storage freeze by using `CSFE DEBUG` with the `STGFRZ=ON` option. In CICS/VS, you can choose a particular transaction for which to freeze storage, using the TRANID option. CICS/CMS is a single-user, single-thread system, so this option is not relevant. However, CICS/CMS does not treat it as an error if you include it. As long as you specify a valid transaction ID, CICS/CMS simply ignores the option.

Storage violation

CICS/CMS chains together segments of unused storage in a free area queue element (FAQE) chain. By using `CSFE DEBUG` with the option, `FAQE=ON`, you can force CICS/CMS to scan the chains, and check the contents of storage accounting areas, every time that it enters the trace program. If it finds an error, it reports it.

In CICS/VS, the storage violation trap produces a dump when it finds an error. In CICS/CMS, the error handler displays message EFH1587T. This gives you the chance to escape to panel EFH122, and check storage, before the CICS test session terminates.

Global trap/trace exit

The global trap/trace exit lets you activate a trap exit routine. In CICS/VS, you use `CSFE DEBUG` with the `TRAP=ON` option, which activates the module DFHTRAP.

CICS/CMS introduces more flexibility, letting you set up a module of your own, as follows:

1. Write a program to be your global trap/trace exit module. The CSECT or ENTRY name must match the module name. The *CICS/VS Problem Determination Guide* defines the form this program must take.

2. Compile or assemble the program to produce a TEXT file.

3. Specify the name of the TEXT file in the TRACTRAP parameter in EFHSETP or EFHPROF.

4. Enable CICS/CMS trace.

Note: You shouldn't try to use this exit except under the guidance of IBM support personnel.

Where to find more information

The CSFE transaction is fully described in the *CICS/OS/VS CICS-Supplied Transactions* manual.

Remote server trace

You can trace the communication activity involved in using the remote server. This trace is intended mainly for IBM support personnel, and is unlikely to be very meaningful to someone who hasn't a comprehensive understanding of the material in the *CICS/VS Remote Server Diagnosis* manual.

You ask for remote server trace output by setting CPIOTRAC to YES in EFHSETP or EFHPROF. CICS/CMS then writes the trace output to a special temporary storage queue, CPIOTRAC. There are two points to remember about this queue:

1. Like all temporary storage queues in CICS/CMS, CPIOTRAC is erased at the end of each CICS test session. You must, therefore, always check it from the escape panel (EFH122), before ending the CICS test session by returning to panel EFH12.

2. You can check the contents of CPIOTRAC using CEBR. However, much of the information in the trace is in hexadecimal, so you should always press PF2 on the CEBR panel to switch to hexadecimal display.

Chapter 16. How to diagnose common problems

Program execution problems are usually grouped into four general types: loops, waits, incorrect output, and abends. This section deals with each of these types in turn, giving some suggestions about how you can use the debugging aids we have been discussing to help you track down the cause of a particular problem. We also deal separately with program checks, though these are really just a type of abend.

Loops

There are two kinds of loop: those involving calls to CMS and those that don't. In either case, once you suspect that your program is looping, you need to stop it.

First, try pressing RESET, followed by PA2. If this displays panel EFH122, your application must have been waiting for you to enter something. From the escape panel (EFH122), you can try to find out what was happening. (For example, you could try turning on EDF, and continuing the test.) You can stop the test from panel EFH122, by pressing PF12 to return to panel EFH12.

If PA2 doesn't display panel EFH122, try pressing RESET, followed by ENTER. One of two things will happen:

1. `VM READ` will appear in the bottom right-hand corner of the screen.

   If you are running with TRAPEXT set to YES in EFHSETP (the default), you can get into the CMS subset, by entering:

   ```
   #CP EXTERNAL 33
   ```

   or by pressing PA1 and entering:

   ```
   EXTERNAL 33
   ```

   `EXTERNAL` is a CP command that simulates an external interrupt to the virtual machine. The result of entering it in this situation, in this form, is to enter the CMS subset, from which you can use CMS debugging aids to get more information. You can then type `RETURN`, to return to CICS/CMS and stop the current CICS test session.

2. Your keyboard will lock.

   In this case, your program is probably in a loop that doesn't contain any calls to CMS. You can't interrupt the virtual machine with the `EXTERNAL` command. Instead, you get into CP by pressing RESET, then press ENTER or PA1. Once you're in CP, you'll see `CP READ` in the bottom right-hand corner of the screen. You can continue in one of two ways, as follows:

a. You can debug using CP commands such as `DISPLAY PSW`, `DISPLAY G`, and so on. When you've finished, you enter `BEGIN`. This returns you to the loop.

b. You can create an artificial program check, by entering the CP command:

```
BEGIN 1
```

CICS/CMS will handle the program check as described in "Program checks" on page 194. It will display an error handler panel, from which you can escape to panel EFH122, and use CMS debugging functions and the CICS/CMS utilities EFHUMAP and EFHUSTG. When you've finished your investigations, you can return to the execution panel (EFH12), by pressing PF12 on panel EFH122.

The first thing to do after you've stopped your looping application, and restarted CICS/CMS (if necessary), is to rerun the application with EDF turned on.

You'll soon know if you have a loop containing EXEC CICS commands; you'll see several commands repeat themselves. Look in your source program for the last command in the loop, and you should find a statement that makes the program go back to the first command in the loop. You might find it helpful to change the source program(s), to include `EXEC CICS RECEIVE` commands near where you suspect the loop is starting. This will make the application stop at those points, letting you look at the values of suspect variables. You can easily remove the `RECEIVE` commands once you've identified the problem.

If you have a loop containing no EXEC CICS commands, your program will "freeze" after a particular EXEC CICS command. You know that the loop must start after the last command that EDF displays, and end before your program reaches any other EXEC CICS command. This tells you where to start looking for the problem. If you can't find it simply by studying the source, you could try running the program again, and using CP/CMS instruction and branch traces to isolate the problem more exactly.

Waits

You should not have any wait problems in CICS/CMS, since none of the typical causes of waits can arise. The `EXEC CICS WAIT` command is ignored by CICS/CMS, as are the `ENQ` and `DEQ` commands. Problems with lack of storage are dealt with immediately by CMS, and will stop CICS/CMS completely, rather than hold it up. The other main causes of waits in CICS/VS are connected with your task's interaction with other users' tasks. Because CICS/CMS is a single-user system, these cannot arise.

Incorrect output

If your program is producing output that you don't expect, or that you know to be wrong, there are some things you can do to try to find out why. Under the next few headings you'll find some suggestions for things you can try. However, they are not intended to be a step-by-step process. Use whichever is most appropriate to the symptoms of your problem.

Check any messages that appeared

You should investigate all messages, including any that you don't believe are causing the problem. Check the error log, which will contain a record of all the messages that CICS/CMS issued while your program was running.

All CICS/CMS messages are described in the *CICS/CMS Messages and Codes* manual. If you get a message that is not documented there, and that is not a VM message, you should talk to your systems people. This may indicate a problem in CICS/CMS itself, and they will want to investigate it.

Check the instruction flow through your program

First, use EDF to check that the logic path of EXEC CICS commands is what you expect. If your symptom was that the data written to a file was incorrect in some way, check the appropriate write commands. If you suspect the logic, but can find no problems with EDF, you will need to check the flow "below" the EXEC CICS level. The EFHUSTG command, used at appropriate points, might show problems in storage allocation. You could also use CP or CMS facilities, such as PER, to trap storage alterations.

Check the data flow through your program

If the symptom of your problem is bad or missing data, it could be that there's something wrong with the input data. Check all sources of input that your program uses.

If the data coming into your program is what you expect, your program must be doing something wrong in manipulating that data.

The problem could be happening when you read the data into the program. Compare each field of the record description in your program with the data in the file.

If the data appears to be reaching the program correctly, you will need to check what happens to it as it goes through the program. If you run the program with EDF, it will stop at each EXEC CICS command. From the EDF panel, you can examine working storage, and check the value of suspect variables. You could also use the CICS/CMS escape feature. For example, from the escape panel (EFH122), you could use CECI to write the current value of a variable to temporary storage with an `EXEC CICS WRITEQ TS` command. You could then check the value with CEBR, before returning to your program. You should be able to get the addresses of variables you want to check from your compiler listing.

If the data appears to be valid throughout your program, the problem is likely to be in writing that data to the screen. The main source of all such problems is faulty definition, either of BMS or file structures.

Look at a map or file structure in an appropriate listing file. Record the name, length and data type of suspect fields. Make sure that you have initialized all data structures to null values. (This is a common cause of problems.)

Use EDF to check the contents of each field as your program writes it. Another common cause of problems is that you are accidentally using an old level of the map or file structure. Look at the `FROM` option in any relevant EXEC CICS commands, and compare the value it contains with your map data structure.

Abends

As we explained earlier, CICS/CMS makes it as easy as possible for you to diagnose the causes of abends. The error handler distinguishes between each type of abend, and gives you a full screen display. This usually includes a description of the particular problem that CICS/CMS has found, together with suggestions about what the cause of the problem might be, and information about storage and register contents if they will be helpful in finding the problem.

Once you've read and understood the message, possibly with the help of the *CICS/CMS Messages and Codes* manual, you may need to use EFHUMAP to find the address of the module or control block from which you want to start your investigations.

If you aren't sure how the problem has arisen, try running the program again and using some of the debugging aids.

You should certainly run the program with EDF, which will let you check the contents of working storage through your program. If you are familiar with trace output, you can turn trace on, and perhaps use EFHUSTG to put additional information into your trace log. You can escape at any stage and use CP/CMS aids, such as PER, to trap alterations in storage.

Program checks

Program checks are really a special type of abend, generated by CMS when it detects that your program has caused a machine interrupt. We need to consider them here because CICS/CMS has its own way of reporting them.

Whenever you get a program check in CICS/VS, your program stops with an ASRA abend, and CICS/VS produces one or more storage dumps (transaction dumps, formatted dumps, snap dumps, or language dumps). Unless you know something about the internal structure of CICS/VS, it can be quite difficult to get the information you want from these dumps.

CICS/CMS doesn't produce a dump in the event of a program check. All the CMS storage is available online, so you can look at it on your screen, using CMS facilities, as explained in Chapter 15, "Debugging tools" on page 171. If you need a printed copy of any of the storage, you can use the CP `DUMP` command.

To help you get exactly the information you want after a program check, CICS/CMS provides a SPIE (specify program interruption exit) option.

Debugging program checks with SPIE

You can turn on the SPIE option by setting the SPIE parameter to YES in EFHSETP or EFHPROF, or by specifying `Yes` for `Trap program checks` in the parameter panel (EFH121). (The SPIE parameter is set to YES in the IBM-supplied EFHSETP.)

With the SPIE option turned on, CICS/CMS does the following when a program check occurs:

1. A SPIE routine constructs a message containing the following information:

a. The type of exception that has caused the program check (operation, execute, protection, and so on).

b. The address and contents of the program status word (PSW) at the time of the check.

c. The address and contents of register 14 at the time of the check.

In the case of both the PSW and register 14, the address comes in one of two forms, depending on where the program check occurred, as follows:

- If the program check was within an application program, the address is a single hexadecimal string giving the absolute address within the program.
- If the program check was in CICS/CMS module code, the address is the absolute address, followed by the hexadecimal offset and name of the module involved. This information is important if you have to report a problem to IBM.

d. The contents of the 16 general purpose registers at the time of the check.

e. A storage address to help you diagnose the common error of branching to zero. This address is the instruction counter in the PSW if it is greater than X'5000'; otherwise, it is the address of register 14.

f. 28 bytes of storage just before and just after the PSW or register 14 address.

2. CICS/CMS then displays the message using the error handler, and writes its contents to the error log.

3. CICS/CMS then issues the ASRA abend. It starts EDF, which displays its ASRA abend panel.

4. CICS/CMS then returns to the execution panel (EFH12).

How to use SPIE information

The best way of using the SPIE information to find an application program problem is as follows:

1. On the EFH125 panel, check the address of the PSW. If that address includes an offset and CICS/CMS module name, the program check is within CICS/CMS or a library. This may result from your program's last EXEC CICS command passing bad addresses.

2. If the PSW address is in the form that indicates that the program check was within an application program, look in the LOAD MAP file to find the start address of your program. Subtract it from the abend (PSW) address. This gives you the offset of the statement that raised the program check.

3. If necessary, recompile your program with an option that produces an object code listing for each statement. For example, you could use the PMAP option for a COBOL program, the MAP option for COBOL II, or the LIST option for PL/I.

4. In the object listing, find the offset that matches the one you found in step 2. You've then isolated the "problem" statement.

From both the error handler panel and the EDF panel, you can display the CICS/CMS escape panel, EFH122. From there, you will find it useful to run EFHUMAP. This will tell you the location of all CICS/CMS modules in storage, and the addresses of control blocks such as the TCA and CSA.

You could also use CMS and CP facilities to dump portions of storage, and to display registers. However, you must be aware that, by the time CICS/CMS has displayed both the error handler and abend panels, you have moved some way from the source of the original problem. PSW and register values displayed in this way are unlikely to be of much help in diagnosing the problem.

Debugging program checks without SPIE

If you get a program check, and you are not running with SPIE turned on, CICS/CMS itself will fail, and you will get a message from CMS. You can try entering the CMS DEBUG command immediately after the program check. In this case, the DEBUG subcommands might give you some useful information about the problem. Unfortunately, in this situation, you can only use the limited DEBUG facilities, and you can't use CMS commands or run utilities such as EFHUMAP.

We therefore advise most strongly that, if a program fails with a program check, you ***always*** rerun it with SPIE active before you try to diagnose the problem.

Debugging program checks with CP

If you would prefer to use CP facilities to debug program checks, you can issue the CP command TRACE PROGRAM before starting the CICS test session. If a program check occurs, you will then enter CP, and you can use facilities such as CP DISPLAY and DUMP to diagnose the problem.

Program checks in the translator

Program checks in the translator are handled in the same way as in CICS/OS/VS: you get a translator diagnostic and a return code of 16.

Where to find more information

If you're new to CICS programming, you'll probably want some information on the best approach to diagnosing and solving problems. The best place to start is the *CICS/VS Application Programming Primer*, which contains a general introduction to debugging, with examples using the sample application (the same application used in this guide).

The main reference books you will need for debugging are:

- The *CICS/VS Application Programmer's Reference Manual (Command Level).*

  This tells you about the error conditions that can result from all EXEC CICS commands, and describes the exec interface block (EIB).

- The *CICS/CMS Messages and Codes* manual.

This describes all the messages and abend codes that can come from CICS/CMS. Its main purpose is to give the extra information that systems people need when they are investigating problems that application programmers have not been able to solve themselves.

Application programmers may find it useful for occasional reference, but should usually find enough information in the messages themselves to diagnose most application problems.

You'll find a more detailed list of useful manuals for CICS/CMS in the bibliography at the front of this book.

The thing to remember when you use any of the CICS/VS books, such as the *CICS/VS Application Programming Primer* and the *CICS/VS Application Programmer's Reference Manual*, is that some of the guidance they give is either irrelevant to CICS/CMS users, or needs to be interpreted for CICS/CMS to be useful.

The main example of information irrelevant to CICS/CMS is anything concerned with the relationship between you and other users of the same CICS system. As a CICS/CMS user, you have your own CICS system, and you cannot be affected by, or affect, other users, unless the number of people using VM causes the real machine to become saturated. That is not a CICS/CMS problem anyway.

You will have to modify the descriptions of techniques for getting debugging information. When the CICS/VS books describe ways of getting transaction dumps, storage contents, and so on, they are describing procedures that you, as a CICS/CMS user, don't need to use (and often can't use).

As we explained earlier, apart from CICS/VS transactions, such as CEDF, your main debugging tools are CP and CMS facilities. Some of the information you will need for debugging, therefore, will not come from CICS/VS documentation at all, but from the VM/SP and VM/PC libraries. The *CMS User's Guide* is the best place to start. It gives a good introduction to the CP and CMS facilities available, and to the way they can be used together. You'll find more detailed information, and guidance on using the CP and CMS debugging tools, in the *VM/SP System Programmer's Guide.*

There's a complete list of all useful documentation for CICS/CMS in the bibliography at the front of this book.

Chapter 17. What if you find an error in CICS/CMS?

If you think you have found an error in CICS/CMS, you first need to report it to the person at your installation who is responsible for reporting problems to IBM.

They will then check that the problem really is in CICS/CMS, and, if so, report it to your IBM Support Center. They will need to complete an authorized program analysis report (APAR) and submit it to IBM, together with enough documentation to enable IBM to re-create the problem, confirm that it is a genuine problem with the product, and then, if necessary, provide a problem fix.

This chapter tells you how to go about reporting errors in CICS/CMS. The first part applies to both the application programmers and the system person to whom they report the problem. It tells you how to make sure that you have a CICS/CMS problem rather than an application program problem. The second part is only for those responsible for reporting problems to IBM. It tells them what they need to include in the problem report to make it possible for IBM to diagnose and solve the problem.

Remember

IBM will only accept problem reports on problems found in CICS/CMS running under VM/SP. If you think you've found a CICS/CMS problem while using a PC, therefore, you must re-create the problem using CICS/CMS on your host VM system before reporting it to IBM.

Making sure you've found a CICS/CMS problem

Sometimes it's easy to spot a CICS/CMS problem, since CICS/CMS spots it for you, and uses the error handler to report it. If you ever get a CICS/CMS error panel containing the sentence:

```
Please note this information
and contact your support center.
```

and you can find nothing wrong with an application program, you've probably got a CICS/CMS problem.

CICS/CMS won't always be so helpful, however. Sometimes you will have to guess that you've found a CICS/CMS problem from the fact that something odd is happening (or maybe not happening), and you can't find a reasonable explanation for it in anything you're doing.

The kind of odd occurrences you might look for are:

- CICS/CMS reporting errors in your use of something you know you're not using. For example, if you ran a transaction that consistently failed with an ABNC abend, indicating a problem with temporary storage, and you were sure that the transaction didn't use temporary storage, you could probably conclude that there was something wrong with CICS/CMS.

- Messages that are not documented in the *CICS/CMS Messages and Codes* manual, and that are not CP or CMS messages.

  As we've seen, with very few exceptions, CICS/CMS displays its own messages, and also traps any CICS/VS messages or abend codes, and displays them using the error handler. If you ever see a CICS/VS message or abend code that is not described in the *CICS/CMS Messages and Codes* manual, you can be pretty certain that there is something wrong in CICS/CMS.

What we need to know to help

This section is for those responsible for reporting problems to IBM. It tells you what information IBM needs to diagnose and solve the problem.

If the problem arose in the first place from running CICS/CMS on a PC, you first need to go through the process that led to the problem, using CICS/CMS on your host VM/SP system.

To report a problem to IBM, via your IBM Support Center, you will need to provide the following:

- The APAR error description
- Information needed to re-create the problem
- A description of the circumstances leading to the problem

The APAR error description

The APAR error description provides an overview of the problem, and of the circumstances leading to it. It should include at least:

- A brief description of the problem as it appears to you.
- The general circumstances in which the problem occurs. For example, if you have a transaction that fails only when you run it with EDF, you should say so.
- The contents of any messages displayed as part of the problem. Sometimes, the message itself will have told you to note some information and report it to IBM. In these cases, make sure you have all that information.
- The name of the module in which the problem occurs. If CICS/CMS has displayed a message, or messages, as part of the problem's symptoms, you can find the module that issued the message from the *CICS/CMS Messages and Codes* manual.

  If you know the location in storage where the problem is occurring, you can use EFHUMAP to find the failing module.

- The version of CICS/VS running on your remote system (CICS/OS/VS Versions 1.6.1 or 1.7; CICS/DOS/VS Version 1.6), if CICS/CMS was using any resources on that system.
- The service level of the CICS/CMS system you are using. The output from EFHUMAP shows you the service level of each component of your CICS/CMS system.
- A list of all the other information you are providing, and the form in which you are providing it.

Information needed to re-create the problem

Depending on the exact nature of the problem, this can include:

- Source listings of all relevant programs.
- Source listings of all relevant BMS map files.
- Listings of the files containing any relevant CICS/CMS tables. For example, for a pseudoconversational program using remote VSAM files and remote temporary storage queues, you should provide your program, file, and temporary storage tables.
- At least a description of the exact format and structure of all data files used, including any remote VSAM or DL/I files. If possible, it would be most helpful to have copies of the files themselves. When this is impossible, however, we need enough information to create files of exactly the same structure.
- Printed copies of the CICS/CMS trace and terminal trace logs. Before printing the terminal trace log, you'll need to convert it using the EFHTLOGT EXEC, as explained in "Using the EFHTLOGT EXEC" on page 185.

Describing the circumstances leading to the problem

If the problem arose while you were executing a transaction, these are the questions you should ask yourself:

- Does the transaction use any special local features, such as macro libraries, modified EXECs, and so on?

  If the transaction does use any local features, you will need to provide copies, listings or descriptions of them.
- Did the problem arise while executing the transaction with EDF, without EDF, or in both circumstances?
- Was the remote server transaction (CEHS) running when the problem arose? Was the user transaction actually using the remote server when the problem arose?
- Which CICS/CMS parameters were set when the transaction ran. The parameters may have been set in EFHSETP EXEC, in the programmer's EFHPROF EXEC, or using either of the parameter panels, EFH121 or EFH1221.
- Which version of which compiler did you use to create the TEXT file?

If the problem arose in circumstances other than when you were executing a transaction, we need a description of those circumstances. Possibilities are:

- While you were using one of the CICS-supplied transactions, CECI, CEDF, or CEBR, or the CICS/CMS file utility transaction, CCU2.

  If you were using CECI, did the problem arise from a particular command?

  If you were using CEBR, you should provide details of the queue you were using, its contents, and how you created it.

- While you were using one of the CICS/CMS EXECs.

  Were you using an EXEC directly, by entering it as a command, or indirectly, via the panels?

- While you were translating a program.

What's next?

This is the end of the book's main text. The rest of the book consists of appendixes, giving reference information on topics described in general terms in the main text, and some extra guidance that you may want to refer to from time to time.

Appendix A. System administration: installation, customization and service

Take note

As its title suggests, this appendix is for system administrators.

We assume that those reading it are fully experienced in using VM, particularly in installing VM. It also assumes some experience in CICS/VS system programming.

If you need to learn more about VM, it's best to start with the *CMS Primer* to get acquainted with using a VM/SP system. You'll find a list of suggested VM/SP books in "VM/SP manuals" on page vi.

Before application programmers can use CICS/CMS (either from a CMS session at a terminal or from a PC), someone has to install the product onto the host VM system from the tapes on which IBM distributes it.

Before making CICS/CMS available to the application programmers, there are some steps that can be taken to customize CICS/CMS to an installation's particular requirements. This customization can include changing the IBM-supplied EXECs that drive CICS/CMS, adding macro libraries, and so on.

From time to time, IBM supplies fixes; that is, new versions of CICS/CMS modules or EXECs containing corrections or improvements to the original product. Someone has to apply these changes to the host copy of CICS/CMS.

These activities (installation, customization, and service) are usually the responsibility of a system programmer (or possibly senior operator), who is the administrator of the CICS/CMS system for the installation. This appendix is therefore addressed to those *system administrators*.

Once CICS/CMS is installed on the VM host system, application programmers who use the product from a terminal connected to the host system can start using it. They just link to the CMS disk on which it resides. Those who want to use CICS/CMS locally on a PC, however, have to download CICS/CMS from the host system onto their PC before they can use it. Since this is the job of the application programmer, not the system administrator, it is not described here, but in "Copying CICS/CMS to a PC" on page 42.

However, if all or some of your application programmers are going to use PCs, you will want to ensure, as part of your installation process, that they will be able to copy the host CICS/CMS system to their PC fixed disks. Furthermore, when you come to apply service to the host CICS/CMS system, you may want to use a PC yourself to help your PC users copy the changed components of CICS/CMS down to their PC copies.

For these reasons, we strongly recommend that, if anyone in your installation is going to use CICS/CMS on a PC, you have access to a PC for system administration use only.

Installing CICS/CMS on a VM system

Installing CICS/CMS on the host is straightforward. You use the VM/SP product installation tool, INSTFPP EXEC. This reduces installation to a simple, conversational procedure for copying the product from the distribution tape to a CMS disk on the host system.

The contents of the distribution tape

The tape on which CICS/CMS is distributed contains five logical files. The contents of each are described in the paragraphs that follow.

Logical Tape File 1 contains two CMS files, as follows:

1. I5668795 011005

   This file contains the product name, and its release and service levels.

2. I5668795 EXEC

   This file is an EXEC (written in REXX), that is used by the installation tool, INSTFPP EXEC, to load the rest of the files from the distribution tape. You can find out more about this file in the *VM/SP Installation Guide.*

   When you execute this EXEC, it transfers the other logical files from the tape to a CMS disk.

Logical Tape File 2 contains a single CMS file, I5668795 MEMO. This is the *Memo to Users*, which is printed automatically by INSTFPP. This memo contains information on:

- Prerequisites you need to run CICS/CMS.
- Where and how the installation EXEC loads CICS/CMS.
- Step-by-step instructions for checking that CICS/CMS has installed correctly. We describe this procedure in "Checking for correct installation" on page 206.
- A list of the errors that might occur during the installation. This includes all errors except those that relate only to the INSTFPP EXEC. For each possible error you will see the return code and the message produced, together with advice on what to do if either the installation or the verification don't work.
- How service is supplied and applied.

Logical Tape File 3 contains:

- TEXT files for the elements that make up the CICS nucleus, the CICS translators, and the CICS command-level interpreter. These are used only when the system needs servicing.

- TEXT files containing sample applications and CICS/CMS utility programs.

All CICS/CMS TEXT files are CMS mode 5.

Logical Tape File 4 contains a dummy file.

Logical Tape File 5 contains the product files. These include:

1. EXECs, all written in REXX, for executing the many elements of CICS/CMS, and the XEDIT files associated with those EXECs.

2. Relocatable modules that make up the run-time CICS/CMS system. These are individual modules for the PC, but are collected into a LOADLIB file for VM/SP. In fact, you could do without these files, since you can re-create them at any time from their TEXT equivalents, using the CICS/CMS EFH15 panel. We have included them to make it easier for you to install CICS/CMS the first time.

3. Non-relocatable load modules for the CICS command translator.

4. The source of the sample application.

All CICS/CMS files other than TEXT files are CMS mode 2 files.

You'll find a complete list of the product files in the *CICS/CMS Program Directory* that IBM supplies with your CICS/CMS tape.

Installing CICS/CMS

You install CICS/CMS in the same way that you install any other CMS product. Your basic reference for the installation is, therefore, the *VM/SP Installation Guide.*

To install CICS/CMS on your host VM system, you run the INSTFPP EXEC. You must have this EXEC set up on the 191 CMS disk of the userid MAINT, and you must run it on that userid.

Before you execute INSTFPP, make sure:

- That you have read the *CICS/CMS Program Directory.* It contains the latest information about CICS/CMS.

- That you have mounted the IBM-supplied tape containing CICS/CMS on the correct tape drive. You must have it on one of the virtual addresses 181 to 184, allocated to the userid MAINT.

- That you have set aside enough virtual storage for the installation. You'll need at least 12 megabytes.

When you execute INSTFPP, it goes through the following eight steps to copy all the elements of CICS/CMS from the distribution tape to a CMS disk of your choice:

1. It asks for the address of the tape drive on which you have mounted the installation tape. You must give it the ***real***, not the virtual, address.

2. It creates a temporary CMS C-disk.

3. It loads the first two tape files onto this disk.

4. It prompts you to load CICS/CMS.

5. It prints the *Memo to Users*.

   Make sure that you read the memo carefully before continuing with the installation. If there are any special instructions that could not be included in this guide, they will be in the memo.

6. It executes the I5668795 EXEC. This EXEC asks you where you want CICS/CMS to be copied (that is, it asks for the filemode (letter) of the CMS disk you have chosen). Having established what you want, it loads CICS/CMS from the tape to your chosen CMS disk.

7. It copies the *Memo to Users* from the temporary disk to the MAINT 319 disk.

8. It checks the return codes, and writes the results of each stage of the installation to the file PROD LEVEL on the CICS/CMS system disk. This file records the results of every program product installation (successful or not).

If anything goes wrong at any stage of the installation, don't try to restart the installation from the point at which it failed. You'll get a clear report of the problem. Correct it, and start the installation again from step 1.

Checking for correct installation

Once you've installed CICS/CMS on your chosen CMS disk, you need to check that it is working properly. You can do this by starting CICS/CMS, and trying the list of EXEC CICS commands given below, using CECI. Unless we specify otherwise in the description, all commands should return a response of `NORMAL`.

Notes:

1. In the descriptions below, we assume a working knowledge of using CECI. If you're not sure of what to do at any point, look in the *CICS/VS Application Programmer's Reference Manual* for guidance.

2. You must enter some of the commands or command options described below in uppercase. To avoid confusion, we recommend that you enter them all in uppercase, as shown. If you don't want to hold the shift key down for every command, you can get CICS/CMS to translate all your entries into uppercase by setting the UCTRAN parameter to YES, using PF7 on panel EFH12, and entering `UCTRAN YES` in the `Additional Parameter` line of panel EFH121. This is explained in more detail in "Changing your CICS/CMS environment within a session" on page 119.

The installation verification process is as follows:

1. IPL CMS, and type `CICSCMS`.

2. On the first CICS/CMS panel, EFH1, select option 2 to display panel EFH12.

3. On panel EFH12, press PF4 to run the command-level interpreter transaction (CECI). Once you have CECI running, you can test selected parts of CICS/CMS by executing EXEC CICS commands as follows:

a. Test the CICS/CMS system. Execute:

```
ASKTIME ABSTIME
```

If you press PF4 to see the exec interface block (EIB), you should see the current time and data in the EIBTIME and EIBDATE fields.

b. Test terminal control. Execute:

```
SEND FROM(&DFHC) ERASE
```

Execute:

```
RECEIVE INTO
```

The `Into` and `LENgth` fields in the command display should contain `THIS IS A SAMPLE` and `00016` respectively.

c. Test basic mapping support (BMS). Execute:

```
SEND TEXT FROM(&DFHC) FREEKB ERASE
```

to display the contents of the CECI variable &DFHC, using BMS. The command should display a screen containing `THIS IS A SAMPLE`.

d. Test file control.

Browse the sample application file ACCTFIL. Before you can do so, you'll need to set up a new CECI variable, &A, using PF5. If you're not sure how to set up CECI variables, look in the *CICS/VS Application Programmer's Reference Manual.*

Give &A a length of 5, and a value of 11111 (the key of the single record in the sample application file). Then execute the following series of browse commands:

1) `STARTBR DATASET(ACCTFIL) RIDFLD(&A)`

   to start the browse.

2) `READNEXT DATASET(ACCTFIL) RIDFLD(&A)`

   to read the first record in ACCTFIL. In the `Into` field you should see the contents of that record (the name "LOCKS GOLDIE" is probably its most notable element).

3) `READNEXT DATASET(ACCTFIL) RIDFLD(&A)`

   to try to read the next record in ACCTFIL. ACCTFIL only contains a single record, so this command should give a response of `ENDFILE`.

4) `READPREV DATASET(ACCTFIL) RIDFLD(&A)`

   to read the previous record in ACCTFIL. The `Into` field should again contain the "LOCKS GOLDIE" record.

5) `ENDBR DATASET(ACCTFIL)`

   to end the browse.

e. Test temporary storage.

First, press PF5 to check the contents of the standard CECI variables: &DFHC, &DFHW, and &DFHR. Press ENTER, then execute the commands:

```
WRITEQ TS Q(CECI) FROM(&DFHC)
WRITEQ TS Q(CECI) FROM(&DFHW)
WRITEQ TS Q(CECI) FROM(&DFHR)
```

to write the contents of the CECI variables to the temporary storage queue, CECI.

Execute:

```
READQ TS Q(CECI) ITEM(2)
```

to read the second entry in the temporary storage queue CECI. The INto field in the command display should contain the value of CECI variable &DFHW.

Use the CICS/CMS escape feature to check the contents of the queue CECI. Press PA2, and CICS/CMS should display panel EFH122. On that panel, press PF2 to run the temporary storage browse transaction, CEBR. On the ENTER COMMAND line in the CEBR panel, enter Q CECI, and press ENTER. The queue CECI should contain the values of &DFHC, &DFHW, and &DFHR, in that order.

When you've finished browsing temporary storage, press PF3 once to end CEBR, again to return to the EFH122 panel, and again to return to CECI. You should return to the panel from which you escaped.

Execute:

```
DELETEQ TS Q(CECI)
```

to delete the temporary storage queue. Then escape to panel EFH122, as described above, to check that it has been deleted using CEBR, and return to CECI.

f. Test transient data.

Execute the commands:

```
WRITEQ TD Q(CECI) FROM(&DFHC)
WRITEQ TD Q(CECI) FROM(&DFHW)
```

to write the values of two of the CECI variables to the transient data queue CECI. Execute:

```
READQ TD Q(CECI)
```

to read the first record in CECI. You should see the value of &DFHC in the INto field in the command display.

Execute:

```
DELETEQ TD Q(CECI)
```

to delete the transient data queue.

When you've finished the installation test, press PA2, to get to the escape panel (EFH122), then press PF12. This terminates the CICS test session, and returns you to panel EFH12.

That completes the installation test using CECI, but there are other things you might like to try.

If you have a COBOL compiler available, you could run through the "getting to know" process in Chapter 2 of this book.

If you're going to make CICS/CMS available to programmers using PCs, you need to make sure that they can copy the host CICS/CMS system down to their PCs. Using your own PC, follow the PC installation procedure described in "Copying CICS/CMS to a PC" on page 42.

Clearly, these installation checks will not use every feature of CICS/CMS. However, they will exercise its most important features, and should assure you that the product is up and running.

Getting the remote server ready to use

If your application programmers are going to use resources on your test CICS/VS system, you will need to make sure that they can use the remote server. This is a two-stage operation:

1. On the CICS/VS system that contains the remote server, make the necessary table entries.

2. On your VM system, set up the connection between CICS/CMS and the CICS/VS system.

Preparing CICS/VS for remote server use

The remote server is a standard part of CICS/OS/VS Version 1.7. If you're using CICS/OS/VS Version 1.6.1, or CICS/DOS/VS Version 1.6, you have to apply the appropriate PTF to put the remote server on your system.

In all cases, you have to define the remote server transaction, CEHS, to your CICS/VS system, as follows:

- Using macros, you need to put an entry in the program control table (PCT) of the form:

```
DFHPCT TYPE=ENTRY
       TRANSID=CEHS
       PROGRAM=DFHCHS
```

  and an entry in the processing program table (PPT) of the form:

```
DFHPPT TYPE=ENTRY
       PROGRAM=DFHCHS
```

- Using resource definition online (RDO), you need two definitions, of the general form:

```
TRANSACTION(CEHS) GROUP(cicscms) PROGRAM(DFHCHS)
PROGRAM(DFHCHS) GROUP(cicscms) LANGUAGE(ASSEMBLER)
```

where `cicscms` is just a suggested group name. You can use what you like. Whatever you use, you must include this group in your GRPLIST list for system initialization, so that it's always installed when CICS/VS is active. If you don't want the remote server available all the time, you can install the group using `CEDA INSTALL`.

If you want to restrict use of the remote server, you can use appropriate TRANSEC or RSL options in the resource definitions above to override the default values of 1 (TRANSEC) and 0 (RSL).

For more information, see the *CICS/VS Resource Definition (Online)* manual.

The remote server also needs the intersystem communication transformer module, DFHXFP. If your system initialization table (SIT) is not already set up to activate it, you will need to add one of the following entries to the SIT:

XFP=YES For CICS/OS/VS 1.7 or CICS/DOS/VS

XFP=1$ For earlier releases of CICS/OS/VS.

On your CICS/VS system you will also need a 3270 entry in your terminal control table (TCT) for each possible CICS/CMS connection. This entry will be the same for all types of link between CICS/CMS and CICS/VS. The VTAM list on the CICS/VS system must associate the network name (NETNAME in the TCT entry) with a particular device address (for example, OC1). If the CICS/VS system is running as a guest of a VM system, users of the remote server may need to know the device address. It's the address they specify on the `DIAL` command when they make the connection to the CICS/VS system.

Setting up the connection between CICS/CMS and CICS/VS

The CICS/VS system can be running either as a guest of the same VM system as CICS/CMS, or on a system remote to the VM system, via PVM.

Remote server—local connection

If the CICS/VS system is a guest running under the VM system on which you have installed CICS/CMS, you need only define sufficient ports in the VM directory of the guest CICS/VS system and configure it appropriately. You will then have to tell your CICS/CMS users how to `DIAL` to the CICS/VS guest.

Remote server—remote connection

If your CICS/VS system is running on a different machine from the one on which you are running CICS/CMS, you will have to use pass-through VM (PVM) to access the CICS/VS system.

This section gives some general advice on using PVM, but for detailed guidance, you will need the *VM/SP Pass-Through Virtual Machine Guide and Reference Manual*, GC24-5206.

PVM allows CICS/CMS to use its remote server on a remote machine as if it is running on a local machine. CICS/CMS users access the remote CICS/VS system

by first dialling the PVM machine from the VM session that they can start using PF8 on panels EFH12 and EFH122.

There is a restriction in CP that can affect your use of PVM. The connection from CICS/CMS to its remote server is through a virtual or logical terminal operated by CICS/CMS, not from a real terminal operated by a real keyboard. CP does not allow users to dial PVM if PVM is already operating a logical terminal on behalf of another user. If a PVM machine connects two VM systems, it's a two-way process. Your CICS/CMS users will be able to use it to get to the remote VM system where CICS/VS is running as a guest; users of the remote VM system will be able to use it to get to the VM system where CICS/CMS is running. Because of the CP restriction, if a user on the remote VM/SP system has dialled the PVM machine, the CICS/CMS users will be "locked out". If they try to dial the PVM machine, they will get one of the CP messages:

```
DMKLOH206E CANNOT CONNECT TO HOST VIRTUAL MACHINE
```

or:

```
DMKDIA206E CANNOT CONNECT TO HOST VIRTUAL MACHINE
```

They will not be able to use PVM from CICS/CMS until all the remote users terminate their VM sessions on the VM system running CICS/CMS.

To avoid this CP restriction, we recommend that you set up two user IDs on the PVM machine. You reserve one of these (for example PVM1) for the exclusive use of those on the VM system running CICS/CMS. You reserve the other (for example, PVM2) for users of the remote VM system. You stop anyone using the CICS/CMS users' PVM user ID by installing an exit that prevents incoming calls. The exit is called DVMUE1, and you'll find a complete description of it in the *VM/SP Pass-Through Virtual Machine Guide and Reference Manual*, GC24-5206.

Having set up two PVM user IDs as described above, you have a system in which CICS/CMS users can dial a PVM machine which cannot itself create logical devices. The users pass through the normal PVM machine to the system with which they need to communicate. External users can still call the VM system from remote systems as they could before you installed CICS/CMS.

Customizing CICS/CMS

Once you have installed CICS/CMS, and ensured that it is complete and ready to be used, you can change it to suit your installation's particular requirements, before making it available to your application programmers.

This section gives some information on the parts of CICS/CMS you are most likely to want to customize.

Whatever changes you make, you should be careful of the following:

- If you change, or remove, any of the files associated with the sample *CICS/VS Application Programming Primer* application, your programmers may be unable to use it with this guide as an aid to learning about CICS/CMS.
- Make sure that you keep a master copy of the CICS/CMS system that IBM supplies, and a record of all the changes you've made. You'll need this when you get service changes from IBM to restore your changes to the changed

system. You'll also need to use the master copy of CICS/CMS to ensure that any apparent problems in CICS/CMS aren't the result of your customization.

Changing the EFHSETP EXEC

One part of CICS/CMS that you may want to change to suit your local requirements is the EFHSETP EXEC.

EFHSETP contains several parameters, each of which defines some aspect of the CICS/CMS working environment. Every time an application programmer starts CICS/CMS, EFHSETP runs, setting up the environment for that session. Application programmers can change their environment using the EFHPROF EXEC, to create something tailored to their own requirements. However, you will most likely want to try to create something using EFHSETP that will suit most of your installation's application programming activity.

The environment definitions (parameters) in the EFHSETP EXEC are the same as those in the EFHPROF EXEC that your application programmers use to set up their own development environment. For a complete description of the available parameters, see Appendix D, "CICS/CMS parameters" on page 241.

By and large, it's probably best to use the defaults that IBM supplies wherever possible, and leave your application programmers to use their EFHPROF EXECs to tailor CICS/CMS to their individual requirements. However, there are a few parameters that you might want to change for all your programmers, as follows:

- The UCTRAN parameter.

  This specifies whether user input is to be translated into uppercase. By default, it is set to NO. However, if your programmers use COBOL, and will therefore use Chapter 2 of this guide to get to know CICS/CMS, they will all use the sample application. This expects user entries to be in uppercase, and will reject them if they are not.

  It will help your programmers, therefore, if you change EFHSETP to set UCTRAN equal to YES.

- The CICSDSA parameter.

  The default that IBM supplies for the dynamic storage area (DSA) size is 256000. You can make this more or less, depending on your requirements. The description of CICSDSA in "General parameters" on page 242 gives some general guidelines on recommended CICSDSA sizes.

- The LANGUAGE parameter.

  You'll want to change the default language parameter, LANGUAGE, if your usual language for application programs is not COBOL.

  You may also want to change the parameters that define the default preload libraries for PL/I and COBOL II. For the reasons why you might change these, and the changes you can make, see "PL/I information for CICS/CMS" on page 86, and "COBOL II information for CICS/CMS" on page 88.

Changing CICS/CMS EXECs

There are several reasons why you might want to change the IBM-supplied EXECs, and/or their associated XEDIT files. The sections following describe the most common reasons, telling you which EXECs or XEDIT files to change, and how to change them.

Using the H Assembler under VM/SP

By default, CICS/CMS running under VM/SP uses the F Assembler. If you want your VM/SP users to use the H Assembler, you will have to change all the EXECs that refer to it.

The EXECs affected are EFHMAPCR and EFHTC. Both call a program called EFHUQVM to check whether the programmer is using a PC or a host-connected terminal. If the return code from EFHUQVM is 1, they assume that the programmer is using a PC, and set the assembler to HASM (the H Assembler). If the return code is 0, they assume that the programmer is using VM/SP and set the assembler to ASSEMBLE (the F Assembler).

To ensure that all your programmers use the H Assembler, find the call to EFHUQVM in EFHMAPCR and EFHTC, and put a statement immediately after it that sets the return code to 1.

Changing the CICS/CMS-supplied macro libraries

CICS/CMS uses three sets of macro libraries, as described in "CICS/CMS macro libraries" on page 99. The EFHTC EXEC contains `GLOBAL` and `FILEDEF` commands that define those libraries and their search order, before it starts to compile a program. If you want to add libraries of your own, or change the names of the CICS/CMS macro libraries, you change the relevant commands in EFHTC.

If you change the name of the EFHxUSER macro library, you'll also have to change the EFHMAPCR EXEC. EFHMAPCR writes user DSECTs to EFHxUSER, so, if you change the name of the user macro library, you will also have to change the statements in EFHMAPCR that refer to it.

Changing the default compiler options

When it invokes the high-level language compilers, the EFHTC EXEC uses a default set of options. The IBM-supplied defaults for these options are described in the following sections of this book:

PL/I "Default PL/I compiler options" on page 86

COBOL "Default COBOL compiler options" on page 86

COBOL II "Default COBOL II compiler options" on page 88

These defaults will override any options you specified when you generated your compilers.

If you want your programmers to use different defaults, you must change the relevant compiler call in EFHTC.

Changing the default language from COBOL

When application programmers run the EFHTC EXEC as a command, without specifying the filetype of the CMS file containing the program, EFHTC assumes that the program is written in COBOL. If your main compiler isn't COBOL, you can change EFHTC to invoke your chosen compiler by default.

Changing the PF key definitions

You can change the PF key definitions in XEDIT files to conform to your installation standards. For example, if you use a CMS editor other than XEDIT, you'll need to change the function definition for PF6 in the files EFH11PRO XEDIT and EFH13PRO XEDIT, both of filetype XEDIT.

Ensuring application programmers have enough virtual storage

You can help your application programmers to get the most out of CICS/CMS by taking steps to ensure that they have enough virtual storage. The things you can do are:

- Set up the programmers' VM userids with at least 2 megabytes of virtual machine storage. CICS/CMS will run in 1.5 megabyte machines, but programmers will quickly run short of virtual storage if they take advantage of all the facilities available.

- Ensure that your programmers only link to the virtual disks that they need when using CICS/CMS.

- If all your programmers use CECI, you can avoid storage fragmentation by making sure that CECI is loaded immediately after the other CICS/CMS modules. You do this by adding the following commands to your EFHSETP EXEC:

```
FILEDEF EFHLIB DISK EFHLIB LOADLIB DISK
GLOBAL LOADLIB EFHLIB
'NUCXLOAD DFHECID DFHECID EFHLIB ( SYSTEM'
```

  If the CECI module (DFHECID) is already loaded, CICS/CMS will ignore the NUCXLOAD, and give a return code of 1.

Note: CICS/CMS does not support VM shared segmented storage. You can't therefore use this method of reducing individual programmers' virtual storage requirements.

Applying service to CICS/CMS

As they use a product, people find errors, or think of improvements, and report these to IBM. If we get to know about problems that are seriously affecting our users' ability to use a product properly, or if we want to introduce new features that will help them get the most out of a product, we provide service updates to that product.

IBM supplies two types of service: *preventative* and *corrective*. The sections following tell you how to apply both types of service to your master CICS/CMS system on VM/SP. You don't apply service directly to programmers' CICS/CMS

systems on PCs. You copy the changed parts of CICS/CMS from the master system to the PC systems.

Preventative service

Preventative service comes as part of the normal service procedure for VM. IBM supplies a VM Program Update Tape (PUT). You then apply this tape to your system using the VMSERV EXEC, as described in the *VM/SP Planning and System Generation Guide* (SC19-6201). To ensure that any service on the VM PUT for CICS/CMS is applied, you will need to add a record for CICS/CMS to your VMPUT PRODUCTS file. The form of the record is:

```
5668795 xxx NORESP
```

where xxx is the virtual address of your CICS/CMS installation disk. NORESP is optional. Usually, when something goes wrong with part of the service process, you get a prompt, asking you what you want to do about it. If you want to run the VMSERV EXEC as a batch job, you'll need to suppress these prompts, which you do by including NORESP.

Corrective service

To correct a problem in CICS/CMS, IBM will send you a tape, containing a fix, which you will then have to apply to your master version of CICS/CMS.

The form of the fix will usually be one or more CMS files, which you can use to replace the existing versions of those files on your master copy of CICS/CMS.

Applying corrective service to CICS/CMS on your VM system is usually a three-stage operation:

1. Copy the files to a CMS disk, as described in "Copying service files to your VM system";
2. Rebuild your CICS/CMS system, incorporating the replacement files, as explained in "Rebuilding your CICS/CMS system" on page 216.
3. Make the new system available to your application programmers, as explained in "Incorporating service changes into your master CICS/CMS system" on page 220.

Figure 57 on page 222 shows diagrammatically a typical application of corrective service, including the steps you need to take to ensure that all your application programmers gain access to the changed system, whether they are using CICS/CMS from VM/SP or VM/PC.

The rest of this chapter describes each stage in that process in detail.

Copying service files to your VM system

The first step in applying corrective service to your CICS/CMS system is to copy the service files to a CMS disk. You must use a different disk from the one on which you have the CICS/CMS system, and you must make sure that its filemode is earlier in the CMS search order than the CICS/CMS system disk. For example, if your CICS/CMS system disk is the T-disk, you could have the replacement files on the R-disk.

You then link both disks from the userid you use for your system administration, and access them.

Whether you then need to go on to the second stage, rebuilding your CICS/CMS system, depends on what kind of fixes you have, as follows:

- If there are any files with a filetype of TEXT among the fix files, you will have to rebuild CICS/CMS. TEXT files contain replacement CICS/CMS system modules, and you can only incorporate these by building them into your master copy, as described in "Rebuilding your CICS/CMS system" below.

- If the replacement files do not contain any of filetype TEXT, you can just put them on a disk earlier in the CMS search order than your existing CICS/CMS system.

  If any of the replacement files are macros or copybooks you have to use the CMS `MACLIB` command, with the `REPLACE` option, to replace the existing versions in one of the macro libraries that CICS/CMS provides. "CICS/CMS macro libraries" on page 99 tells you how to do this. You can then put the new version of the macro library on the same disk as the rest of the service files. You can identify replacement macros or copybooks by their filetype, which will be either MACRO or COPY.

  Having put all your replacement files on an appropriate CMS disk, you can make the new system available to your application programmers, as described in "Incorporating service changes into your master CICS/CMS system" on page 220.

Rebuilding your CICS/CMS system

So that you can rebuild your CICS/CMS system quickly and easily, CICS/CMS includes an EXEC that does the rebuilding for you. It builds new load modules from the replacement TEXT files, and gives you the opportunity to save time by rebuilding only those parts of CICS/CMS affected by the changes.

Start CICS/CMS, and ask for option *7* on the selection panel (EFH1). This produces the panel shown in Figure 55.

```
EFH15                          APPLY SERVICE

 Select one of the following:

                  1   Build Relocatable CICS/CMS LOAD Modules
                  2   Build CICS/CMS COBOL/COBOL2 Translator EFHCTRAN
                  3   Build CICS/CMS Assembler Translator EFHATRAN
                  4   Build CICS/CMS PL/I Translator EFHPTRAN
                  5   Build CICS/CMS Utility Modules for Transient Area
                  6   Build CICS/CMS Other Utilities
                  7   Build CICS/CMS Run-time TXTLIB
                  8   All of the above


          Selection       ===>

          Input Filemode  ===> *        Default - *

          Output Filemode ===> A        Default - A

          LOADLIB name    ===> EFHPRIV  Default - EFHPRIV (Option 8 only)

PF1=Help          PF3=End          PA2=CMS Subset
```

Figure 55. The CICS/CMS service panel (EFH15)

The service EXEC divides CICS/CMS into seven distinct parts, enabling you to rebuild only those parts of CICS/CMS that have changed. If you know what you need to rebuild, you can take advantage of one of the first seven options. If you're not sure what you need to rebuild, you can soon find out.

The CICS/CMS system contains a file called EFHBUILD CNTRL. Each entry in this file contains three fields, as follows:

1. The right-hand field contains the name of an individual TEXT file.

2. The middle field contains the type of the file. This can be:

   REL For a relocatable module

   NRM For a non-relocatable module

   TRN For a transient area module

   MEM For a member of the run-time TXTLIB

   LIB For the EFHXLIB library.

   Where there are a series of modules, all of the same type, only the first is identified by the type above. The rest are shown as a period (.) to indicate the same type.

3. The left-hand field contains the name of the CICS/CMS component to which each TEXT file belongs.

To find out which option of panel EFH15 you need to use:

1. Find the TEXT filename in the right-hand column.

2. Check the filetype:

- In nearly every case, if the filetype is REL or NRM, the left-hand column tells you which component contains that module, and therefore which EFH15 (and possibly EFH151) option you need. The options involved are 1, 2, 3, and 4.

  There are two exceptions. EFHBOOT and EFHUCMS1 are both of filetype NRM. These are the modules classed by CICS/CMS as "Other Utilities", and you build them with option 6.

- If the filetype is TRN, the module is a transient area module, and you build it using option 5.

- If the filetype is MEM, the module is a member of TXTLIB, and you build it using option 7.

The most likely option you'll need on EFH15 is option 1. This lets you rebuild one or more of the relocatable load modules that are the heart of the CICS/CMS system.

If you want option 1, you just enter 1 on the selection line, and press ENTER. You don't enter anything in any of the other fields on panel EFH15.

If you want any other option, however, you may need to change one or more of the other fields on the panel, as follows:

- Input Filemode

  This defines the mode of the disk on which you've stored the TEXT files that CICS/CMS will use for the rebuild. If you've taken the advice we gave earlier, you will have put these on a disk earlier in the search order than those on your CICS/CMS system disk. If for some reason you haven't, you will need to change the default (*) to the letter identifying the disk on which you've put them.

  Note that, if you specify a particular disk, CICS/CMS will look for files to build on that disk only. You'll therefore have to ensure that ***all*** unchanged components of CICS/CMS are on that disk, as well as the new files.

- Output Filemode

  By default, CICS/CMS puts the rebuilt modules on your CMS A-disk. This lets you check that the rebuild has worked successfully before replacing the changed modules on your CICS/CMS system disk. This option lets you put the rebuilt modules on a disk of your choice.

  Note that, wherever you ask CICS/CMS to put the rebuilt modules, it will always write the SYSPRINT output from the rebuild to your A-disk.

- LOADLIB name

  The four CICS/CMS load modules are built into a single LOADLIB type file. By default, CICS/CMS calls this EFHPRIV. If you have changed this name on your system, you will need to change the default name on the panel to your own LOADLIB name.

  This option applies only to options 1 and 8 on panel EFH15.

If you select option 1 on panel EFH15, CICS/CMS will display a new panel, as shown in Figure 56.

```
EFH151              BUILD CICS/CMS RELOCATABLE LOAD MODULES

 Select one of the following:

                  1   Build CICS/CMS Load module DFHECID
                  2   Build CICS/CMS Load module EFHDFH
                  3   Build CICS/CMS Load module EFHMOD
                  4   Build CICS/CMS Load module DFHTRP
                  5   All of the Above


          Selection       ===>

          Input Filemode  ===> *              Default - *

          Output Filemode ===> A              Default - A

          LOADLIB name    ===> EFHPRIV        Default - EFHPRIV


PF1=Help          PF3=End          PA2=CMS Subset
```

Figure 56. Service panel for CICS/CMS load modules (EFH151)

EFH151 has the same "additional" lines as EFH15. If you want to change any of them, you should do so now, as explained on the previous page.

Having established which of the modules you need to rebuild, using EFHBUILD CNTRL, you can then select the appropriate option, and press ENTER. CICS/CMS will then rebuild the selected module (or all the modules if you select option 5).

The remaining options on panel EFH15 don't lead on to further panels. They rebuild the CICS/CMS modules shown in EFH15 immediately, as follows:

- Option 2 builds EFHCTRAN, which contains the modules for the COBOL and COBOL2 translator.
- Option 3 builds EFHATRAN, which contains the modules for the assembler translator.
- Option 4 builds EFHPTRAN, which contains the modules for the PL/I translator.
- Option 5 builds the two storage utilities, EFHUSTG and EFHUMAP, and the EFHUQVM utility, which CICS/CMS uses in various places to see whether the programmer is using CICS/CMS from a host-connected terminal or a PC.
- Option 6 builds the EFHUCMS1 utility (which converts local CMS files to the CICS/CMS format), and the EFHBOOT module (which "bootstraps" EFHMOD, and handles program checks).
- Option 7 builds the CICS/CMS run-time text library, EFHXLIB TXTLIB.

The final option, 8, builds a complete CICS/CMS system. The less you rebuild, the quicker you'll get it done. However, if you're at all unsure of what you need to rebuild, you should select option 8.

Whichever option you select, CICS/CMS rebuilds the parts requested, and stores them on the disk you've specified (by default, your A-disk). Before going any further, you should make sure that you've corrected the problem for which you received the fixes. Stop CICS/CMS, and then start it again. As long as the rebuilt modules are on a disk earlier in the CMS search sequence than your CICS/CMS system disk, CICS/CMS will use them instead of the versions on the system disk.

To complete the service procedure, you need to update your CICS/CMS system using the new modules, as explained below.

Incorporating service changes into your master CICS/CMS system

The last stage of applying service to CICS/CMS is to update the master version of CICS/CMS on your VM system, and alert your application programmers to the change.

If you have not had to rebuild any CICS/CMS modules to apply the service changes, all you need to do is replace the files on the CICS/CMS system disk with the new versions that you copied onto another CMS disk, as described in "Copying service files to your VM system" on page 215. If you have rebuilt some CICS/CMS modules, the new modules will be on your A-disk. Replace the existing CICS/CMS modules with these new versions.

If you have rebuilt the entire CICS/CMS system (option 8 on panel EFH15), you will need to remove all existing CICS/CMS modules from the CICS/CMS system disk, and replace them with the new ones from your A-disk.

Finally, you need to make sure that all your application programmers are using the new version.

For those using CICS/CMS from host-connected terminals, you need to make sure that they know there's a new version of the system. They'll pick it up automatically from the system disk.

Those using CICS/CMS from PCs, however, will need to change their version of the product on their local fixed disks. As long as the changed modules are ***not*** the ones in the LOADLIB file, it's easy for you to help them do this. Tell them which modules you have changed. They can then use VMPCSERV to connect to the host system, link to the disk where you've put the changed modules, and copy those modules down to their local disks. There they can use them to replace their existing copies of the modules.

If the changed modules are part of the LOADLIB, however, your PC users will need more help.

VM/PC does not support LOADLIB type files. When your PC users first install CICS/CMS on their local fixed disks, they automatically pick up the four separate modules from the system you've installed on the host. When you run the service EXEC, however, it doesn't rebuild those four modules; it rebuilds the LOADLIB file. The only way of changing the four CICS/CMS LOAD modules separately is through an option on the PC user's version of the EFH151 panel.

The procedure you need to go through is as follows:

1. Using your PC, connect to your host system using VMPCSERV, and start CICS/CMS. From the CICS/CMS selection panel (EFH1), select option 7 (`Download from Host`). This process is fully described in the first few pages of "Copying CICS/CMS to a PC" on page 42.

You will then see the panel shown in Figure 24 on page 44.

2. Select option 8 (`Build CICS/CMS Relocatable Load Modules`).

   You will then get a panel similar to the one you get from selecting option 1 on panel EFH15 under VM/SP, as shown in Figure 56 on page 219. The only difference is that the `LOADLIB` line is missing; it has no meaning for a PC. The purpose of this panel when you're using a PC is to rebuild the CICS/CMS load modules and, instead of putting them into the LOADLIB file, to copy them down to the PC user's A-disk. You can rebuild only the module(s) needed, or all the modules.

   Note: This operation can take some time. It depends on how much you're rebuilding, and on how fast the connection is between the PC and your host VM system. For a complete rebuild, over a direct connection, it should take about 45 minutes.

   Once CICS/CMS has finished rebuilding the module(s), you'll get the message:

   ```
   EFH9110I Download completed successfully
   ```

3. At this stage, you have the rebuilt modules on the PC's A-disk. To make them available to all your PC users, you need to put them up on your master CICS/CMS system on VM/SP. Use VMPCSERV again to copy the modules from your local PC A-disk, to your A-disk on the host. From there, you can use them to replace the existing copies of the modules on your CICS/CMS system disk.

Once you've completed this procedure, you can make the changed load module(s) available to your PC users in the same way as any other changed modules. You tell the users which modules are affected, and let them use VMPCSERV to copy them down to their local fixed disks.

Figure 57 on page 222 shows the complete corrective service process diagrammatically. The diagram shows what you'd need to do to incorporate service changes to the CICS/CMS relocatable load modules, EFHDFH and EFHMOD.

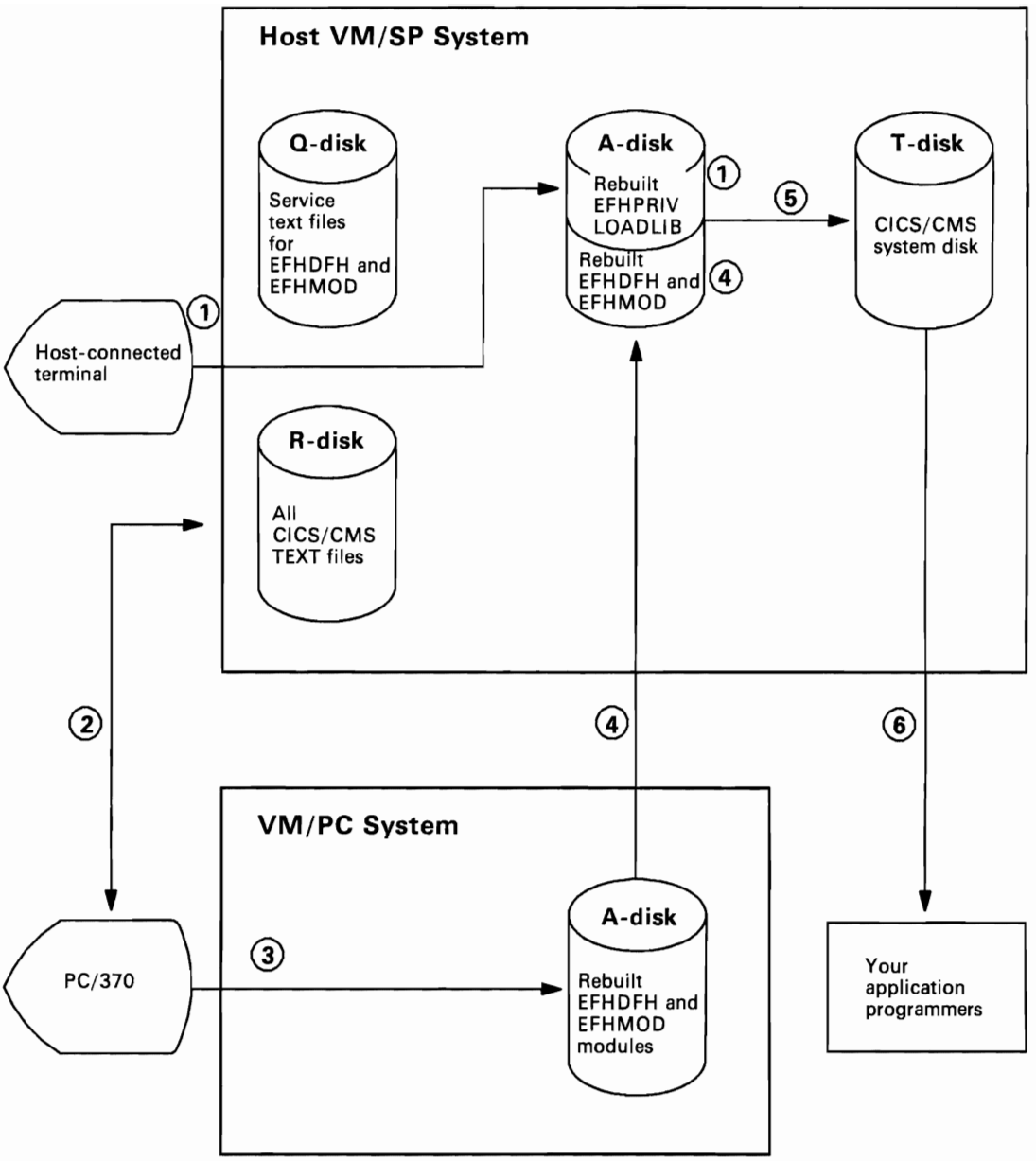


Figure 57. The corrective service process

The steps indicated by the numbers in the diagram are as follows:

1. On VM/SP, use options 2 and 3 on panel EFH151 to rebuild the EFHPRIV LOADLIB module on the host system A-disk.

2. On VM/PC, start VMPCSERV to connect to the host VM system.

3. On VM/PC, use options 2 and 3 on panel EFH151 to rebuild the EFHDFH and EFHMOD modules.

4. From VM/PC, copy the rebuilt modules from the PC's A-disk to the host VM system's A-disk.

5. On VM/SP, copy the rebuilt EFHPRIV LOADLIB, EFHDFH, and EFHMOD modules from the host system A-disk to the CICS/CMS system disk, replacing any existing copies.

6. Tell all application programmers using CICS/CMS under VM/SP that you've changed the CICS/CMS system.

   Tell all application programmers using CICS/CMS under VM/PC to replace the copies of EFHMOD and EFHDFH they currently have on their fixed disks with the new copies on the CICS/CMS system disk.

Appendix B. Differences between CICS/CMS and CICS/VS

CICS/CMS does not give you the whole of CICS, as you would expect to find it on an MVS or VSE system. This appendix gives a general account of the main differences you'll find between CICS/CMS and CICS/VS, and, where appropriate, the effect those differences have on application programming.

Notes:

1. The points made in this appendix about resources such as files don't apply if you use the remote server to access them on a remote CICS/VS system.

2. This appendix does ***not*** tell you exactly what subset of the application programming interface (API) CICS/CMS supports. It highlights general areas, for example, interval control, or BMS routing, but does not give details of commands and command options supported. The *CICS/CMS Application Programmer's Reference Summary* gives you all the detailed information you need on the API.

General points

- CICS/CMS does not support applications using the macro-level application programming interface (API), or CALL level DL/I requests.

- CICS/CMS does not support programs written in RPG II.

- CICS/CMS does not support the recovery features of CICS/VS. There is no local support for journals, logging, dynamic backout, or automatic transaction restart. The only recovery is that provided by CMS. However, when a CICS/CMS application uses the remote server to access resources on a remote CICS/VS system, those resources have all the recovery and integrity protection you have defined for your CICS/VS system.

File control

- CICS/CMS does not support VSAM files locally. Instead, it uses CMS files as pseudo-VSAM files, letting you access data with the CICS/VS file control API.

  For further information, see "How CICS/CMS supports VSAM files" on page 60.

- CICS/CMS does not support BDAM or ISAM files, either locally, or on a remote CICS/VS system.
- CICS/CMS issues a diagnostic message if you access a file for both browse and update, but does not reject the request.
- Other restrictions on ***local*** file use are as follows:
  - For CICS/CMS direct access storage, only extended disk format files, blocked at 512, 1K, 2K, or 4K bytes, are supported. CICS/CMS does not support the alternative, 800-byte block, CMS disk format.
  - CICS/CMS supports the RBA access method by treating the RBA value as an RRN value. CICS/CMS does not, therefore, support applications that rely on precise RBA values.
  - CICS/CMS writes all new records to the end of relative record data sets (RRDS), and returns a new value in `RIDFLD`. This is the way that CICS/VS treats entry sequential data sets (ESDS).
  - Under CICS/CMS, you must specify the `EQUAL` option on all `READ`, `STARTBR`, and `RESETBR` commands that use RRDS files.
  - CICS/CMS supports the `EXEC CICS DELETE` command for keyed files only.
  - CICS/CMS does not require the records in a data set named in an `EXEC CICS WRITE MASSINSERT` command to be in strict ascending order.
  - You can only have one update outstanding on a file at any one time. (CICS/VS allows multiple update requests on files if each request is issued by a different task.)
  - CICS/CMS does not support alternate indexes for its pseudo-VSAM files.

Terminal control

- CICS/CMS supports only those 3270 terminals that VM/SP supports. These are defined in the *VM/SP Terminal Reference* manual.

  CICS/CMS testing is confined to the functions of 3270 screens and 3270 printer terminals. It doesn't support functions specific to other terminal types, such as batch data interchange terminals.
- Since CICS/CMS supports only one interactive terminal, you can only run EDF in "two-terminal" mode to get EDF output on your terminal screen from a running printer transaction.
- CICS/CMS does not support partitions.
- Due to a CP restriction, you will get unpredictable results if you use successive `EXEC CICS SEND FROM (text)` commands, and:
  1. There are no other reads between commands
  2. The `text` does not contain any set buffer address orders

3. You don't use the ERASE option.

- CICS/CMS doesn't support BMS routing.

DL/I support

- CICS/CMS supports EXEC DL/I requests only; it does not support the CALL DL/I interface.
- CICS/CMS does not support DL/I data bases locally. All EXEC DL/I requests must be shipped to a remote CICS/VS system through the remote server.

  For more information, see "Starting and using the remote server" on page 105.

- CICS/CMS doesn't support the EXEC DL/I STAT command.

Temporary storage

All temporary storage in CICS/CMS is set up in CMS virtual storage. There is no equivalent of CICS/VS auxiliary temporary storage. This means that all temporary storage queues you create in a CICS test session are erased when you end that session by returning to panel EFH12.

Program control

- Different application programs must have different external entry names. For COBOL programs, this means that programs that will run in the same CICS/CMS session must have different names (specified as PROGRAM-ID in the IDENTIFICATION DIVISION).
- CICS/CMS does not support recursive COBOL programs. You can, however, execute programs recursively in all the other languages that CICS/CMS supports, including COBOL II.
- In CICS/VS, if a program uses the EXEC CICS XCTL command to transfer control to another program, and that program doesn't exist, CICS/VS raises a PGMIDERR condition in the program that issues the XCTL. In the same situation, CICS/CMS displays an error handler panel, and raises the PGMIDERR condition in the outermost program in the transaction, if there is one.

  For example, if program A issues a LINK to program B, and program B issues an XCTL to a nonexistent program:

  - CICS/VS raises the PGMIDERR condition in program B.
  - CICS/CMS terminates program B, issues an error handler message, and raises the PGMIDERR condition in program A.

  If program A is the first level program in a transaction, and it issues an XCTL to a nonexistent program:

- CICS/VS raises PGMIDERR in program A.
- CICS/CMS terminates program A, issues an error handler message, and waits for you to enter another transaction ID.

Standard transactions

- CICS/CMS does not support CECS (all its functions are available through CECI).
- CICS/CMS does not support CEDA, CEMT, CEST, or CEOT. Either it does not recognize their functions, or it controls them by different methods.
- CICS/CMS does not let you invoke master terminal (CEMT) functions from application programs.

New system programmer commands in CICS/OS/VS 1.7

CICS/OS/VS Version 1 Release 7 introduced the `INQUIRE` and `SET` commands of the command level application programming interface, together with the spool commands of the CICS/VS interface to the job entry subsystem (JES). CICS/CMS support for these new features is as follows:

- You should use the `INQUIRE` and `SET` commands only in application programs destined for a CICS/OS/VS 1.7 system. The CICS/CMS translator supports them, and you can check their syntax using CECI under CICS/CMS. However, the only option on either command that CICS/CMS supports in an executing program is `DATASET`. If an application issues an `INQUIRE` or `SET` command with any other option, CICS/CMS will either ignore the command and continue, diagnose the command and continue, or terminate the CICS test session.

  You'll find detailed information on using `INQUIRE` and `SET` in the *CICS/OS/VS Customization Guide*.

- CICS/CMS does not support any of the spooler commands: `SPOOLOPEN`, `SPOOLWRITE`, and so on.

Appendix C. CICS/CMS panels

This appendix gives reference information on the principal CICS/CMS panels.

"How CICS/CMS panels connect" shows, in a diagram, the flow of control between the panels.

"The principal CICS/CMS panels" on page 231 displays the principal CICS/CMS panels, tells you what the available options mean, and describes the functions of the PF and PA keys. Where a particular option or key function is explained in more detail in the main body of the manual, we refer you to the section containing that explanation.

How CICS/CMS panels connect

The diagram overleaf shows how the principal CICS/CMS panels connect with each other, and what path you have to follow to work your way through from one step in developing an application to any other step.

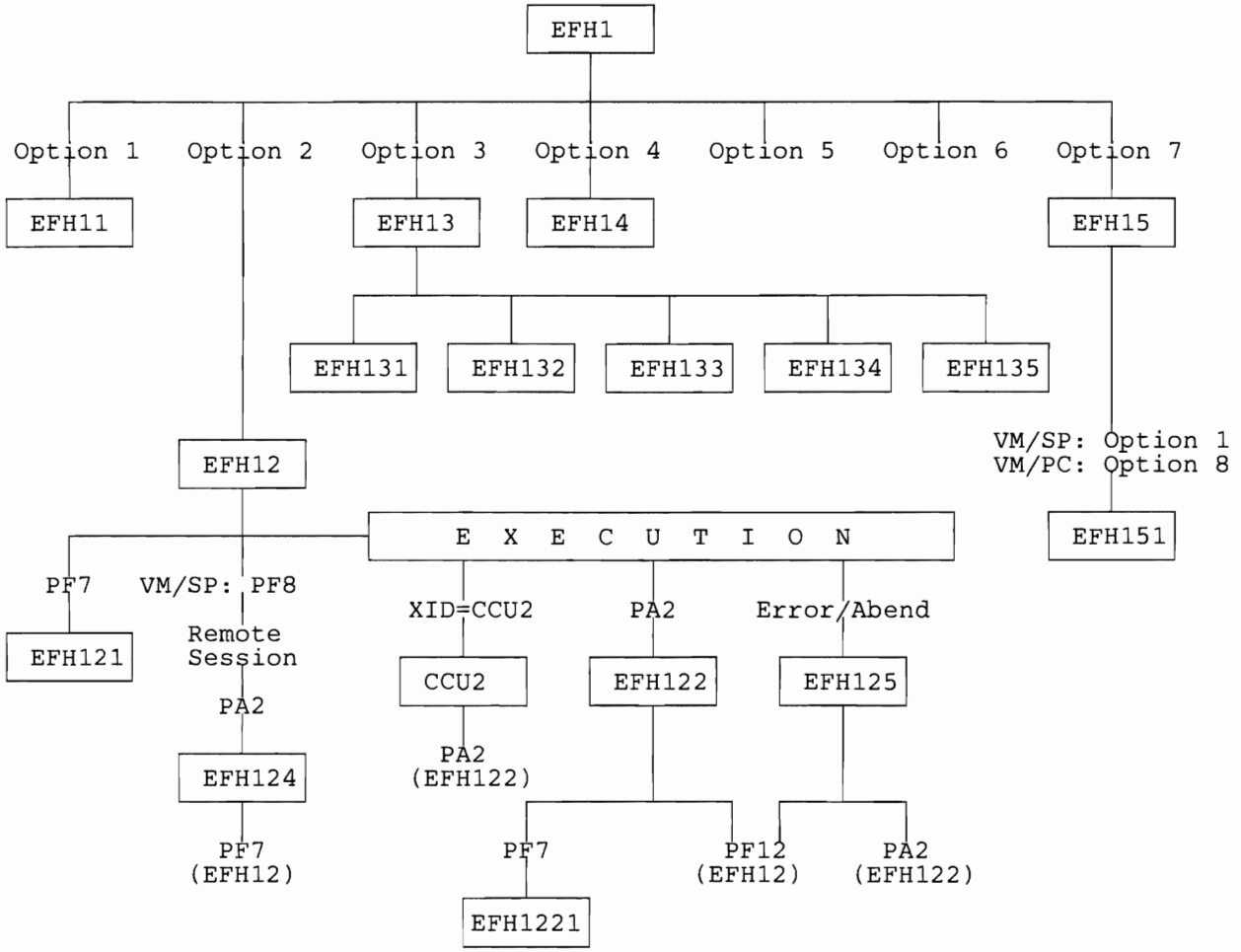


Figure 58. Panel flow diagram

The principal CICS/CMS panels

The panels described in this section are:

The program development selection panel (EFH1)

The application objects listing panel (EFH11)

The start execution panel (EFH12)

The escape functions panel (EFH122)

The purpose of describing these panels here is to give you a single reference source for each panel, and for the functions of its PF and PA keys. Most of this information is provided in the main body of the book, but not always in one place.

Notes:

1. The panel displays define only PF keys in the range PF1 to PF12. If you're using a PC or terminal with 24 PF keys, you can use PF keys 13 to 24 for the same functions as PF keys 1 to 12.

2. None of the descriptions include the PA1 key. Pressing PA1 always takes you to the control program (CP).

The program development selection panel (EFH1)

Panel display

```
EFH1                CICS/CMS PROGRAM DEVELOPMENT SELECTION

 Select one of the following:

                  1   List Application Files
                  2   Execute Program/Transaction
                  3   List Resource Tables
                  4   Convert CMS file to CICS/CMS file
                  5   Erase Temporary Files
                  6   Release CICS/CMS Nucleus Extensions
                  7   (Apply Service)(Download from Host)

          Selection ===> 1

 For application filelist specify the optional list criteria required:

          Criteria  ===> ACCT* * *


             (c) Copyright IBM Corp 1985

PF1=Help          PF3=End          PA2=Enter CMS Subset
```

Option descriptions

The list below describes the facilities available from the panel, by option number.

1. Displays a new panel, EFH11, containing a list of application files. You specify the filename, filetype, and filemode of the file(s) that you want listed by changing the default options in the line beginning:

   ```
   Criteria  ===>
   ```

   If you omit any of the criteria, CICS/CMS assumes *. Note that CICS/CMS remembers these criteria from one CICS/CMS session to the next. From this list (panel EFH11), you can assemble maps, and translate and compile source programs.

   For more information on EFH11, see Chapter 7, "Preparing your application for testing" on page 93, and "The application objects listing panel (EFH11)" on page 234.

2. Displays another panel (EFH12), providing various facilities for executing a CICS program or transaction.

   For more information on EFH12, see Chapter 9, "Testing an application" on page 123, and "The execution panel (EFH12)" on page 236.

3. Displays a panel (EFH13) that lets you list the CICS/CMS tables that you can currently access.

For more information on EFH13, see "Updating tables" on page 22.

4. Executes the CICS/CMS utility that converts local CMS files into keyed pseudo-VSAM files that you can use in CICS/CMS.

   For more information, see "Converting local CMS files to CICS/CMS keyed files" on page 110.

5. Erases the trace table, the error log, the EFHUMAP log, the terminal trace log, and the load map. This can be useful for "cleaning up" during a test session. You may, for example, want to test a program with trace, print the trace for that test session, then erase the trace and start again.

   As described in Chapter 4, "Setting up your CICS/CMS environment" on page 47, you can use the EFHPROF EXEC to erase these features automatically at the start of every CICS/CMS session.

6. Releases the extensions to the CMS nucleus that are retained between CICS/CMS sessions to improve performance. Unless you need the storage they occupy for some purpose unconnected with CICS/CMS, you should never use this option. If you do, there will be a delay the next time you execute a transaction.

   Note: This option will release only CICS/CMS modules loaded in the CMS nucleus. It won't release any application programs that you've loaded into the nucleus by specifying NUCXLOAD as their load method. The only way of releasing application programs loaded in the CMS nucleus is to re-IPL CMS, or to use the CMS `NUCXDROP` command to release programs by name.

7. Offers different facilities, depending on the way that you are using CICS/CMS, as follows:

   - If you are using a terminal attached to a host VM system, this option says:

     ```
     Apply Service
     ```

     It is the way that the system administrator incorporates changes, supplied by IBM, into the master CICS/CMS copy on the host.

   - If you are using a PC, this option says:

     ```
     Download from Host
     ```

     This is the way that you copy CICS/CMS from the host to your fixed disk (see "Copying CICS/CMS to a PC" on page 42).

PF and PA key functions

Key	Function
PF1	Displays a panel of information to help you use EFH1.
PF3	Stops the CICS/CMS session, and returns you to CMS.
PA2	Puts you into the CMS subset. To get back to EFH1 from the subset, type `RETURN`.

The application objects listing panel (EFH11)

Panel display

```
 EFH11              FILELIST:ACCT* * *                    Line 1 of 10

Cmd   Filename Filetype Fm Format LRECL    Records     Blocks   Date     Time
      ACCT03   COBOL    B2 F         80         70          2 03/07/85 10:56:51
      ACCT04   COBOL    B2 F         80        144          3 03/07/85 10:56:51
      ACCT02   COBOL    B2 F         80        378          8 03/07/85 10:56:50
      ACCT00   COBOL    B2 F         80         17          1 03/07/85 10:56:49
      ACCT01   COBOL    B2 F         80        371          8 03/07/85 10:56:49
      ACCTSET  MAP      B2 F         80        171          4 03/07/85 10:56:48
      ACCTFIL  EFHVDATA B2 F        383         14          2 03/07/85 10:56:48
      ACCTFIL  EFHVINDX B2 F        512          2          1 03/07/85 10:56:47
      ACCTIX   EFHVDATA B2 F         63         14          1 03/07/85 10:56:47
      ACCTIX   EFHVINDX B2 F        512          2          1 03/07/85 10:56:47









PF1=Help     2=Refresh 3=End       4=All this Name 5=Translate/Compile 6=Edit
PF7=Backward 8=Forward 9=Install  10=MAP-TEXT&ADS 11=Execute          12=
====>
```

Note: The files listed in the example above are those that make up the sample application. The actual files that you will see in an EFH11 display will depend on the criteria you have set in the selection panel (EFH1). The files are sorted by date.

PF and PA key functions

Key	Function
PF1	Displays a panel of information to help you use EFH11.
PF2	Refreshes panel EFH11, to include all files that currently satisfy the criteria you set in panel EFH1.
PF3	Returns you to the panel from which you entered this one. Usually, this returns you to panel EFH1. However, if you displayed the current panel by pressing PF4 on a previous EFH11 panel, you will return to that version of EFH11.
PF4	Displays a new EFH11 panel showing all files on all your currently-accessed disks with the same filename as the file the cursor was beside when you pressed PF4. The principal use of PF4 is to see the files created when you assemble a file containing maps, or translate and compile a source program file.

PF5 Translates and compiles the source program file indicated by the cursor.

For more information, see "Translating and compiling/assembling programs" on page 95.

PF6 Executes your editor, so that you can review or change the file indicated by the cursor.

PF7 Scrolls the screen display backward.

PF8 Scrolls the screen display forward.

PF9 Tells CICS/CMS that you want the file indicated by the cursor to be the table for a particular resource during this CICS/CMS session. CICS/CMS will know which kind of table it is by its filetype, which must be one of those defined in your installation EFHSETP EXEC, or your own EFHPROF EXEC. The filename can be anything you like.

For more information on this method of defining tables to CICS/CMS, see "CICS/CMS tables" on page 66.

PF10 Assembles the BMS map(s) in the file indicated by the cursor. The output from the assembly is:

1. The symbolic map, which CICS/CMS writes to a file with the same filename as the source file, and with a filetype of TEXT
2. The DSECT, which CICS/CMS adds to the appropriate EFHxUSER MACLIB file.

For more information on this process, see "Assembling maps" on page 94.

PF11 Displays the execution panel (EFH12), with the filename of the file indicated by the cursor in its `Program` line. You must therefore ensure that you have a file with that filename, and with a filetype of TEXT, containing a translated and compiled CICS program.

The line immediately beneath the key definitions is the command line. From here you can execute any valid CMS command.

The execution panel (EFH12)

Panel display

```
EFH12                          EXECUTE PROGRAM/TRANSACTION

 Type the name of the transaction or program and any optional parameters:

    Transaction    ===>
    Program        ===>

 Status:

    EDF is OFF









PF1=Help         2=CEBR          3=End        4=CECI       5=          6=Start-Clear
PF7=Set-parms    8=VM-Session    9=EDF ON    10=          11=         12=Terminate
PA2=CMS Subset

CP/CMS Command ===>
```

Functions available

There are three function lines in this panel, as follows:

Transaction

To execute a transaction from this panel, type the transaction name on the line beginning:

```
Transaction ===>
```

and press ENTER.

If the transaction has any parameters, you can give them after the transaction ID, just as in CICS/VS. However, the total input is limited to 52 characters. If you need more, press PF6 (see below) to start the transaction from a blank screen.

Program

To run a program from this panel, type the program name on the line beginning:

```
Program ===>
```

and press ENTER.

The program name you give must be the filename of a file of filetype TEXT that you have previously created using the EFHTC EXEC.

Note: You can't give parameters with the program name as you can with a transaction name.

CP/CMS Command You can execute most valid CP or CMS commands from this panel, on the line beginning:

```
CP/CMS Command ===>
```

The only exceptions are `#CP` commands (which you can execute only when your terminal is in CP read mode) and the `HX` command.

PF and PA key functions

PF1 Displays a help screen for this panel.

PF2 Executes the CICS/VS transaction, CEBR, for browsing temporary storage. When you finish using CEBR, you press PF3 to display its termination screen. On the top line of that screen you can type the CICS/CMS command:

```
CCMS QUIT
```

to get back to EFH12.

PF3 Quits EFH12, and returns you to:

- Panel EFH1, if you started CICS/CMS with the `CICSCMS` command, and then displayed EFH12 by selecting option 2 on EFH1.
- Panel EFH11, if you displayed panel EFH12 by putting the cursor on EFH11 alongside a program file and pressing PF11 (`Execute`).
- CMS, if you started CICS/CMS with the `EFH12` command, which bypasses EFH1 and displays EFH12 immediately.

PF4 Executes the CICS/VS transaction, CECI, for executing EXEC CICS commands interactively. When you finish using CECI, you press PF3 to display its termination screen. On the top line of that screen, you type:

```
CCMS QUIT
```

to return to EFH12.

PF6 This starts CICS and clears panel EFH12, giving you a clear screen. This gives you the chance to type a transaction ID, and any associated parameters, under the same conditions that a user might experience in CICS/VS.

PF7 Displays panel EFH121, the parameter panel, which lets you change some of the parameters that define the CICS/CMS environment in which you are working. We describe this panel fully in "Changing your CICS/CMS environment within a session" on page 119.

When you finish setting parameters on EFH121, you can return to EFH12 in one of two ways:

- By pressing PF3 to save your new parameter values and return to EFH12
- By pressing PF12 to drop the changes you've made since you last pressed ENTER, and return to EFH12.

PF8 This key applies to using CICS/CMS under VM/SP only. If you are using CICS/CMS under VM/PC, the PF key function definition will be blank instead of `VM-Session`.

PF8 displays the logo panel of another VM session. The main purpose of this key is to start the process that lets you connect to a remote CICS/VS system and run the remote server transaction (CEHS).

You'll find a full account of this in "Transferring to a remote CICS/VS system from a terminal" on page 106.

PF9 The switch key for the execution diagnostic facility (EDF). If the key definition says `EDF ON` when you press PF9, it switches EDF on, changes its own definition to `EDF OFF`, and changes the Status definition in the top part of the panel to `EDF is ON`. If you press PF9 when the definition is `EDF OFF` it turns EDF off and reverses the status and PF key definitions.

PF12 Does exactly the same as PF3.

PA2 Switches you to the CMS subset, from which you can issue several CMS commands. To return to EFH12 from the CMS subset, type the command:

```
RETURN
```

The escape panel (EFH122)

Panel display

```
EFH122                          ESCAPE FUNCTIONS

  Type the name of the transaction or program and any optional parameters:

      Transaction    ===>
      Program        ===>


  Status:

     EDF is OFF

     Nest level




PF1=Help       2=CEBR        3=End      4=CECI        5=Appln-PA1   6=Appln-PA2

PF7=Set-parms  8=VM-session  9=EDF ON  10=Reset-msgs 11=           12=Terminate
PA2=CMS-subset

CP/CMS Command ===>
```

Functions available

The functions available from the escape panel are exactly the same as defined for panel EFH12. The difference is that, because you display the escape panel from within a running transaction, anything you do is nested at a certain level within your CICS/CMS session. This nest level is shown graphically in the `Nest level` field by a series of slashes (/) and asterisks (*), as described in "Using the CICS/CMS escape feature" on page 127.

You should also note that when you issue a CP or CMS command from panel EFH122, CICS/CMS executes that command in the CMS subset. This limits the number of commands you can issue to those described as valid for the CMS subset in the *CMS User's Guide*. Furthermore, because CICS/CMS doesn't let you run any of its EXECs within the CMS subset, you can't do so from the escape panel.

PF and PA key functions

PF1 Displays a help panel for EFH122.

PF2 Starts CEBR, as defined for EFH12.

PF3 Quits the escape panel, and returns you to the CICS test session, at the point from which you escaped.

PF4 Starts CECI, as defined for EFH12.

PF5 Passes the PA1 key to an application that needs it.

If you press PA1 in CICS/CMS, it passes control to CP. If you have an application which needs you to enter PA1 for it to continue, you need to find a way of bypassing CICS/CMS, and getting the PA1 through to your application. You can do this by escaping from your transaction to EFH122 at the point where the transaction is waiting for PA1. You can then press PF5 on EFH122. This passes PA1 to the application, and resumes the application's execution immediately.

PF6 Serves the same purpose as PF5, but for the PA2 key. Pressing PA2 in a CICS test session displays the escape panel, EFH122. If you want to pass a PA2 to your application, you press PF6 on that panel display.

PF7 Displays panel EFH1221, which serves the same purpose as the parameter panel you get from EFH12 (EFH121), but lets you change fewer parameters.

For more information, see "Changing your CICS/CMS environment within a session" on page 119.

PF8 Displays up a new VM logo, as described for EFH12. This function is only available in CICS/CMS under VM/SP.

PF9 Switches EDF on and off, as described for EFH12.

PF10 Restarts the display of all error handler messages you have previously suppressed.

The CICS/CMS error handler panel (EFH125) has a PF key (PF5) that lets you stop an unwanted message. You usually need this when you have a loop in a program that repeatedly executes an EXEC CICS command that uses a CICS/VS feature that CICS/CMS does not support. You will get a series of identical messages. Pressing PF5 on one of the EFH125 panels containing the message will suppress all future occurrences.

PF12 Quits EFH122, and returns you to the execution panel (EFH12). This is always true. It doesn't matter how many levels of nesting you've created to get to an escape panel. If you press PF12, it will drop all the levels and go back to EFH12.

PA2 Enters the CMS subset, as described for EFH12.

Appendix D. CICS/CMS parameters

CICS/CMS provides two EXECs for setting up the parameters that define its working environment, as follows:

- EFHSETP is part of the master CICS/CMS system. It lets the system administrator establish a default environment for all programmers using CICS/CMS.
- EFHPROF resides on application programmers' A-disks. It lets them change some of the default settings established in EFHSETP, thereby creating their own working environment.

The parameters that EFHSETP and EFHPROF use to set up the CICS/CMS environment are identical. The "EFHSETP/EFHPROF parameters" section contains reference information for both application programmers and system administrators.

EFHSETP/EFHPROF parameters

We've divided the parameter descriptions into functional subsections. Each description gives the keyword, followed by a description of its function, including the range of values associated with it.

The defaults mentioned in the descriptions are the values that IBM supplies in the copy of EFHSETP shipped with the product.

Note: In the EFHSETP EXEC itself, there are some parameters that we don't describe here. These parameters are intended for CICS/CMS internal use only, and you should not change them in EFHSETP, or include them in an EFHPROF EXEC.

General rules for CICS/CMS parameters

There are three general types of parameters in CICS/CMS, as follows:

Parameter	Valid values
Numeric	A series of decimal digits, with or without a sign, and possibly with surrounding blanks.
Boolean	One of the values `YES` or `NO`, in uppercase or lowercase. CICS/CMS only reads the first character, so you can abbreviate them to `Y` and `N`.

Character A series of letters and numbers, starting with a letter. Character values are usually limited to 8 characters.

You can also give a character parameter a value of period (.), as shown in some of the individual parameter descriptions in this appendix. This is a general method in CICS/CMS of giving a character parameter a null or blank value. When you want to give such a parameter a value, you can do so within CICS/CMS, using the EFH121 panel, as described in "Changing your CICS/CMS environment within a session" on page 119.

Debugging parameters

Keyword	Description
USER	Specifies the debugging program you want to use. You can set this to DEBUG, which selects the CMS DEBUG program, or to the name of your own debugging program. (Default: no debugging program) The name of the debugging program you choose will appear in the `DEBUG program` line in panel EFH122. CICS/CMS will call the program each time a CICS test session starts, and each time a program is loaded within a CICS test session.
TRACE	Specifies whether trace output is required, as follows: **YES** Write trace entries to the file defined in the trace table definition parameters (see "Parameters for the CICS/CMS trace file" on page 251). **NO** Do not create a trace file. (Default)
EDF	Specifies whether the execution diagnostic facility (EDF) is to be enabled or disabled by default when you display panel EFH12. **YES** Enable EDF **NO** Disable EDF (Default)

General parameters

Keyword	Description
CWASIZE	An integer between 0 and 2048, defining the size of the CICS common work area (CWA) in bytes. (Default: 512)
TWASIZE	An integer between 1024 and 32767, defining the size of the CICS transaction work area (TWA) in bytes. (Default: 1024)

SPIE

Specifies how program checks will be handled.

YES CICS/CMS will report program checks using its error handler panel (EFH125). (Default)

NO Program checks will be reported by a CMS message.

You can find out more about the way that CICS/CMS handles program checks in "Program checks" on page 194.

CICSDSA

An integer between 50000 and 400000, defining the size of the CICS dynamic storage area (DSA). (Default: 256000)

Note: You have to be careful with the value you give CICSDSA. It defines the size of your CICS working space. CICS/CMS uses it for I/O buffers, storage and terminal buffers, and so on. What is left after allocating the DSA is your CMS working storage, in which your programs will run.

You may need to increase your DSA size if you use large temporary storage data sets, or large CICS/CMS keyed pseudo-VSAM files. Your DSA size can also depend on the size of your virtual machine. If you run in a virtual machine of over 2 megabytes, you may want to increase the DSA size. If you run in a virtual machine smaller than the 2 megabytes we advise as the minimum, you may need to reduce the DSA size to, say, 160000 bytes. However, if you make the DSA too large, you will not leave enough storage for your programs to run. On the other hand, if you make it too small, you will put CICS/CMS under stress.

We recommend that you start with the default supplied by IBM.

STGTABLE

Specifies the maximum number of entries that CICS/CMS can write to the EFHUSTG log. (Default: 500)

Each entry in the EFHUSTG log is 20 bytes long. If you set the STGTABLE value too high, you might run out of virtual storage. You should find 500 adequate for most purposes.

You'll find a complete description of EFHUSTG, and the rules that govern its use, in "The EFHUSTG log" on page 177.

DATEFORM

A combination of the characters DD (day), MM (month), and YY (year), defining the form in which CICS will display the date in the `DATE` option of the `EXEC CICS FORMATTIME` command. It doesn't affect the form of the date in the exec interface block (EIB). (Default: DDMMYY)

TRAPEXT Lets you handle external interrupts with the EXTERNAL 33 command. (Default: YES)

"Loops" on page 191 explains why you would need the EXTERNAL 33 command.

CPIOTRAC If you specify YES, you will get trace output for the communication activity involved in using the remote server.

You'll find an introduction to remote server trace in "Remote server trace" on page 189. (Default: NO)

Terminal control parameters

Many of the parameters listed here apply to particular terminal types only. If you are an application programmer, using EFHPROF to set up your own terminal, and you are not sure whether your terminal supports a particular option, look in the user's guide for your terminal before using it.

Most of these parameters serve the same purpose as the operands of the same name used to define terminal control table (TCT) entries in CICS/VS for VTAM 3270 terminals. If you want to find out more about them, you can look in the *VTAM 3270 Devices* section of the *Terminal Control Table* chapter of the *CICS/VS Resource Definition (Macro)* book.

There are two general points to note about the parameters for screens with special features: EXTDS, COLOR, PS, HILIGHT, and SETATTR:

1. The parameter has no effect if your application doesn't use the feature. That is why the default value for all these parameters is YES.

2. If you change the value to NO, and your application uses the feature in a BMS map, BMS will ignore the feature in your map and DSECT, even if your screen supports it.

Keyword **Description**

SCRNHT The maximum number of lines on the screen. (Default: 24)

This is the value returned by an `EXEC CICS ASSIGN SCRNHT` command.

SCRNWD The maximum column width of the screen. (Default: 80)

This is the value returned by an `EXEC CICS ASSIGN SCRNWD` command.

Together, SCRNHT and SCRNWD define the ***physical*** limits of the screen you're using. They correspond to the CICS/VS TCT entries DEFSCRN and ALTSCREN. If your terminal doesn't support an alternate screen size, you should never change SCRNHT or SCRNWD. If your terminal does support an alternate sceen size, you may have to change the values, as explained in the description of ALTSCREN below.

BMSHT The maximum number of lines on a BMS page. (Default: 24)

BMSWD The maximum number of characters in each line of a BMS page. (Default: 80)

Together, these parameters correspond to the CICS/VS TCT entry PGESIZE.

UCTRAN Specifies whether lowercase characters will be translated to uppercase on input.

Note: This applies only to input handled by CICS terminal control. It has no effect on the input fields in CICS/CMS panels. Note also that transaction IDs that you enter on panels EFH12 and EFH122 are always converted into uppercase.

YES Translate all user entries to uppercase.

NO Do not translate user entries. (Default)

ALTSCREN Specifies whether the alternate screen size will be used. This parameter affects all input and output; it applies to CICS terminal control ***and*** to CICS/CMS panels.

YES Use the alternate screen size.

NO Use the terminal's default screen size. (Default)

Note: If your terminal doesn't support an alternate screen size, you should never change ALTSCREN from NO. If your terminal does support an alternate screen size, and you want to use it in a particular CICS/CMS session, you must change three parameters: ALTSCREN, SCRNHT, and SCRNWD. You set ALTSCREN to YES, and SCRNHT and SCRNWD to the height and depth respectively of the alternate screen size. When you want to change back to your default screen size, you must remember to change not only the ALTSCREN parameter (to NO), but also the SCRNHT and SCRNWD parameters (to your default screen size).

EXTDS Specifies whether the screen supports any of the extended facilities.

YES The screen can support one or more of the extended color (COLOR), character set (PS), and highlighting (HILIGHT) features. (Default)

NO The screen supports no extended features.

COLOR Specifies whether extra color facilities are available.

YES The screen has the extended color feature, which lets you choose colors for each field or character. (Default)

NO The screen has no extended color feature.

PS Specifies whether extra character sets are available.

YES The screen supports the programmed symbol facility. This lets you define up to six different character sets that your programs can use. (Default)

NO The screen supports only the standard character set.

HILIGHT Specifies whether extra highlighting is available.

YES The screen supports the extended highlight facility. This lets you display fields or characters using reverse video, underlining, or blinking. (Default)

NO The screen supports no highlight facilities except the bright feature.

SETATTR Specifies whether the screen supports the set attribute order.

YES The screen supports the set attribute order. (Default)

NO The screen does not support the set attribute order.

PRTBMSHT Defines the number of lines in a BMS page for a printer terminal. (Default: 24)

PRTBMSWD Defines the number of characters in each line of a BMS page for a printer terminal. (Default: 80)

Together, these parameters define the shape of the BMS page buffer. They are equivalent to the TCT PGESIZE entry for the printer. For a detailed description of the rules governing the use of PRTBMSHT and PRTBMSWD, see "EFHPROF definitions for printing" on page 136.

PRTFRMHT Defines the maximum number of lines on each printer page. (Default: 66)

PRTFRMWD Defines the maximum width of lines on a printer page. (Default: 132)

These parameters correspond to controls on a 3270 printer, and are fully described in "EFHPROF definitions for printing" on page 136.

PRTBUFF Defines the printer buffer size. (Default: 1920) For a description of the rules governing its use, see "EFHPROF definitions for printing" on page 136.

TERMTRAC Specifies whether you want terminal trace.

YES CICS/CMS will write trace entries for your terminal data stream to the file defined by the terminal control trace log parameters (see "Parameters for terminal control trace" on page 249).

NO CICS/CMS will not create a terminal trace log. (Default)

You'll find a description of the terminal trace log in "CICS/CMS terminal trace" on page 184.

LOCSYSID The 4-character identification by which you refer to your local system. If an `EXEC CICS` command specifies this in the `SYSID` option, CICS/CMS will search only the local system for the requested resource. (Default: LOCL)

REMSYSID The 4-character identification by which you refer to your remote CICS/VS system. (Default: REMT)

If an application accesses remote resources using the `SYSID` option on `EXEC CICS` commands, the value given in `SYSID` must match the value in this parameter. If an application accesses resources defined as remote in a CICS/CMS table, the value of REMSYSID is irrelevant.

For guidance on using LOCSYSID and REMSYSID, see "Using the SYSID option" on page 66.

Parameters for program control

Keyword **Description**

LANGUAGE Defines the default language for your source programs. CICS/CMS will assume that a program is written in this language, unless you have specified otherwise in a CICS/CMS program table file (see "Program tables" on page 70).

Valid values for LANGUAGE are COBOL, COBOL2, PLI, and ASSEMBLE. (Default: COBOL)

LOADMETH Defines the method by which programs will be loaded. CICS/CMS will assume this method for a program, unless you specify something different in a CICS/CMS program table file (see "Program tables" on page 70).

Valid values for LOADMETH are:

INCLUDE	Load in the user area. (Default)
NUCXLOAD	Load in the nucleus extension.

PROGFN The CMS filename of the file to be used as the CICS/CMS program table. (Default: CICSCMS)

PROGFT The CMS filetype of the file to be used as the CICS/CMS program table. (Default: EFHTPROG)

PROGFM The CMS filemode of the file to be used as the CICS/CMS program table. The default, *, tells CICS/CMS to look for a file with the defined filename and filetype on any currently accessed CMS disk, starting with the A-disk.

PLILOAD Specifies the CMS filename of a library of PL/I routines that CICS/CMS loads each time you run a PL/I program. (Default: EFHPLIXX)

For more information, see "PL/I information for CICS/CMS" on page 86.

CO2LOAD Specifies the CMS filename of a library of COBOL II routines that CICS/CMS loads each time you run a COBOL II program. (Default: EFHCO2XX)

For more information, see "COBOL II information for CICS/CMS" on page 88.

Parameters for the error handler log file

You'll find a description of the error handler log file in "The CICS/CMS error log" on page 168.

Keyword	Description
LOGFN	Filename of the CMS file to contain the error handler log. (Default: CICSCMS)
LOGFT	Filetype of the CMS file to contain the error handler log. (Default: ERRORLOG)
LOGFM	Filemode of the CMS file to contain the error handler log. (Default: A1)

Parameters for terminal control trace

You'll find a general description of terminal control trace in "CICS/CMS terminal trace" on page 184.

Keyword	Description
TCLOGFN	Filename of the CMS file to contain the terminal trace output. (Default: CICSCMS)
TCLOGFT	Filetype of the CMS file to contain the terminal trace output. (Default: TERMLOG)
TCLOGFM	Filemode of the CMS file to contain the terminal trace output. (Default: A1)

Parameters for destination control table

Keyword	Description
DCTFN	The CMS filename of the file to be used as the CICS/CMS destination control table. The default, period (.), means that you start with no table filename defined. Before you can run applications that need to refer to the table, you must tell CICS/CMS which file to use, as described in "CICS/CMS tables" on page 66.
DCTFT	The CMS filetype of files to be used as CICS/CMS destination control tables. (Default: EFHTTD)
DCTFM	The CMS filemode of files to be used as CICS/CMS destination control tables. The default, *, tells CICS/CMS to look for a file of the defined filename and filetype on any currently accessed CMS disk, starting with the A-disk.

Intrapartition transient data parameters

You'll find a complete account of using these two parameters in "Intrapartition transient data" on page 56.

Keyword	Description
INTRAMRL	Defines the maximum record length of all intrapartition queues, in bytes. (Default: 32763)
INTRAFT	Defines the filetype of the CMS files used for intrapartition queues. (Default: CICSTDI)

Parameters for PSB directory

Keyword	Description
PDIRFN	The CMS filename of the file to be used as the CICS/CMS PSB directory. The default, period (.), means that you start with no directory filename defined. Before you can run applications that need to refer to the directory, you must tell CICS/CMS which file to use, as described in "CICS/CMS tables" on page 66.
PDIRFT	The CMS filetype of files to be used as CICS/CMS PSB directories. (Default: EFHTPDIR)
PDIRFM	The CMS filemode of files to be used as CICS/CMS PSB directories. The default, *, tells CICS/CMS to look for a file of the defined filename and filetype on any currently accessed CMS disk, starting with the A-disk.

Parameters for temporary storage table

Keyword	Description
TSTFN	The CMS filename of the file to be used as the CICS/CMS temporary storage table. The default, period (.), means that you start with no table filename defined. Before you can run applications that need to refer to the table, you must tell CICS/CMS which file to use, as described in "CICS/CMS tables" on page 66.
TSTFT	The CMS filetype of files to be used as CICS/CMS temporary storage tables. (Default: EFHTTS)
TSTFM	The CMS filemode of files to be used as CICS/CMS temporary storage tables. The default, *, tells CICS/CMS to look for a file of the defined filename and filetype on any currently-accessed CMS disk, starting with the A-disk.

Parameters for file control table

Keyword	Description
FCTFN	The CMS filename of the file to be used as the CICS/CMS file table. The default, period (.), means that you start with no table filename defined. Before you can run applications that need to refer to the table, you must tell CICS/CMS which file to use, as described in "CICS/CMS tables" on page 66.
FCTFT	The CMS filetype of files to be used as CICS/CMS file tables. (Default: EFHTFILE)
FCTFM	The CMS filemode of files to be used as CICS/CMS file tables. The default, *, tells CICS/CMS to look for a file of the defined filename and filetype on any currently accessed CMS disk, starting with the A-disk.

Parameters for the CICS/CMS trace file

You'll find a detailed description of CICS/CMS trace in "CICS/CMS trace facilities" on page 173.

Keyword	Description
TRACEFN	Filename of the CMS file to contain the CICS/CMS trace output. (Default: CICSCMS)
TRACEFT	Filetype of the CMS file to contain the CICS/CMS trace output. (Default: TRACELOG)
TRACEFM	Filemode of the CMS file to contain the CICS/CMS trace output. (Default: A1)

Other parameters

Keyword	Description
TRACTRAP	Names the user global trace/trap exit program. The default, period (.), specifies that there is no such program available. This is mainly intended for IBM support personnel, but is introduced in "Field engineering trace" on page 187.
LOADLIB	Specifies the name of the CICS/CMS load library. (Default: EFHLIB) *Note:* This parameter applies to VM/SP only. It has no meaning for PC users. "Rebuilding your CICS/CMS system" on page 216 tells you how CICS/CMS uses this library.

Appendix E. CICS/CMS trace entries

If you look at a few lines of output from a CICS/CMS trace log, you'll see that the entries can be divided into two basic types:

```
CC0A0200 TCAI C1C3C3E3 00010001 ACCT     ACCT00   7013674C  /
F1CC0400 STG  00000510 0108F04C           0<          400BFC38  /
C8000400 ++++ 00096DF0 8C000518                     500BE512  /
CC1A0200 PCP  0100C120 0000000C          ACCT00   70133EAC  /
F2010400 PC   00000000 00000000          ACCT00   5013678A  /
F1CC0400 STG  00000400 0108F04C       0<          400BFC38  /
C8000400 ++++ 00097310 8C000408                     500BE512  /
EA000300 TMP  01000300 000970B8          ACCT00   4013442E  /
EA000500 TMP  01000300 00092CE4        U          400C1178  /
CC1C0200 PCP  000201E8 00000001    Y     ACCT00   70134D74  /
F1930400 STG  0009053A 0108F04C           0<          50134ED8  /
C8000400 ++++ 00097800 93090540     1               500BE512  /
CC180200 PCP  FD096E00 04020532   >      ACCT00   70134616  /*
```

The two types are:

- Those beginning with CC
- All other entries.

The entries beginning CC are unique CICS/CMS entries, generated by its EFH modules. The purpose of this appendix is to give you enough information to interpret the CICS/CMS entries. If you want more general information about the contents of the CICS/CMS trace log, see "General debugging tools" on page 171.

The "all other entries" are standard CICS/VS trace entries, issued by DFH modules. In other words, they convey the same information, about the same events, as they do on a CICS/VS system, and are fully described in the *CICS/VS Problem Determination Guide*. The *CICS/OS/VS Program Debugging Reference Summary*, SX33-6048 gives a quick reference to CICS/OS/VS trace entries.

The relevant fields in each CICS/CMS entry are:

1. The hexadecimal code of the entry. This is the byte immediately following the characters CC.
2. A 3- or 4-character string identifying the CICS/CMS module that wrote the trace entry. This is the second stand-alone field in each trace entry.
3. The contents of the first information field (field A). These are the 4 bytes immediately following the module identifier.

4. The contents of the second information field (field B). These are the 4 bytes following Field A.

5. The contents of field A and field B, translated into EBCDIC. These are the 8 bytes in the fifth stand-alone field in each trace entry.

6. The name of the resource involved. This is the string that precedes the last hexadecimal field in each entry.

7. The address of the code from where the trace entry was requested. This is the last hexadecimal field in each entry.

The last characters in the display above (/ and *) show the nesting level of transactions. For more information on this, see "Using the CICS/CMS escape feature" on page 127.

Each trace entry also includes the date and time at which the entry was made, and the line number of the entry within the trace file. These are on the extreme right-hand side of the file, and are not shown in the sample trace output.

In this appendix we concentrate on five of the fields. For example, in the first CC entry in the sample output above, the relevant fields are:

Code 0A

Module TCAI (the CICS/CMS module that creates a new task control area)

Field A C1C3C3E3 (EBCDIC translation ACCT)

Field B 00010001 (EBCDIC translation unprintable)

Resource ACCT00

The rest of this appendix describes these fields, in numeric (hexadecimal) order of the codes. In each case, we give the code, with an explanation of the event it represents, followed by descriptions of the contents of fields A and B and the resource field. The descriptions are subdivided to show which CICS/CMS modules generate which trace entries.

EFHMAIN entries

These entries are generated by CICS/CMS starting and ending transactions.

00 Start of a transaction.

Field A All zeroes.

Field B All zeroes.

Resource The string `Starts...`

01 End of a transaction.

Field A All zeroes.

Field B All zeroes.

Resource The string `Ends.....`

EFHTCAI entries

These entries are generated by the CICS/CMS module that creates a new TCA.

08 Increment the task level on entry to a new task.

Field A The transaction ID.

Field B The new task level (first 2 bytes), and old program level (second 2 bytes).

Resource The name of the program to which a `LINK` is about to be made.

09 Just before decrementing the the task level and returning to the caller.

Field A The transaction ID.

Field B The new task level (first 2 bytes), and old program level (second 2 bytes).

Resource The name of the program to which control is about to return.

0A About to execute the first program in a transaction.

Field A The transaction ID.

Field B The current task level (bytes 1 and 2) and program level (bytes 3 and 4).

Resource The name of the program about to start.

0B Return to the main program of a transaction.

Field A The transaction ID.

Field B Only the first byte is relevant. It's the program control request code (usually `01`, for an `EXEC CICS LINK`).

Resource The name of the main program.

EFHTRANI entries

These entries are generated by transaction initialization.

0E On entry to the module.

Field A The transaction ID.

Field B The first byte is the start code, indicating whether the transaction was started by a `RETURN TRANSID`, a `CCMS START` command, or by other terminal input. The remaining 3 bytes are not relevant.

Resource The name of the program.

0F Start a task.

Field A The transaction ID.

Field B A 1-***bit*** flag, indicating that the task is terminating if it is on. The rest of the field is not relevant.

Resource The name of the program.

EFHPCP entries

These entries are generated by the CICS/CMS program control program.

18 Increment program level (as the result of a `LINK`).

Field A The current LIFO storage pointer (TCALCDSA).

Field B The first byte indicates the language of the program specified in the `LINK`, as follows:

01 PL/I
03 assembler
04 COBOL
05 COBOL II

The remaining 3 bytes are not relevant.

Resource The name of the program specified in the `LINK`.

19 Decrease link level on a `RETURN`.

Field A The current program level.

Field B The first byte indicates the language of the program to which execution is returning, as follows:

01 PL/I

03 assembler

04 COBOL

05 COBOL II

The remaining three bytes are not relevant.

Resource The name of the program to which execution is returning.

1A Start of PCP executable code.

Field A The first 2 bytes are the program control request byte, indicating the type of request (LINK, XCTL, and so on). You'll find a description of this field in the *CICS/VS Problem Determination Guide*. The last 2 bytes are not relevant.

Field B The address of the most recent entry in the PCP LINK/RETURN stack.

Resource The name of the program in the request.

1B On return from PCP to its caller.

Field A The first byte is the program control return code. The other 3 bytes are not relevant.

Field B The address of the current entry in the PCP LINK/RETURN stack.

Resource The name of the program in the request.

1C Loading a program with the INCLUDE command.

Field A The address where the program is loaded, if field B is all zeroes.

Field B The response code from the load.

Resource The name of the loaded program.

1D Loading a program with the NUCXLOAD command.

Field A The address where the program is loaded, if field B is all zeroes.

Field B The response code from the load.

Resource The name of the loaded program.

EFHPCPI entry

This entry is generated by program control initialization.

1E On entry to the module.

Field A All zeroes.

Field B All zeroes.

Resource The string `EFHPCPI`

EFHPCIER entry

This entry is generated by the program control module that ensures that a failing `INCLUDE` command doesn't affect subsequent `INCLUDE`s.

1F On entry to the module.

Field A All zeroes.

Field B All zeroes.

Resource Name of the module for which the attempt to load using `INCLUDE` has failed.

EFHTCPI entry

This entry is generated by terminal control initialization.

20 On entry to the module.

Field A All zeroes.

Field B All zeroes.

Resource The string `EFHTCPI`.

EFHERR entries

These entries are generated by the CICS/CMS error handler.

21 The CICS test session is terminating after the message.

Field A All zeroes.

Field B All zeroes.

Resource The string `TERMCALL`.

22 A message has been displayed.

Field A The address from where EFHERR was called.

Field B All zeroes.

Resource The number of the CICS/CMS message issued.

EFHESCAP entry

This entry is generated by the CICS/CMS module that displays the escape panel (EFH122).

24 Escape panel was displayed.

Field A All zeroes.

Field B All zeroes.

Resource The string `EFHESCAP`.

EFHEDFX entries

This entry is generated by the CICS/CMS EDF program.

28 On entry to EDF.

Field A The contents of the EDF request byte, in hexadecimal. You can find out the meaning of this value from the description of the EISEDFRB field in the *CICS/VS Data Areas* manual.

Field B The address of the task control area (TCA).

Resource The contents of the request byte, in EBCDIC.

EFHTCS entry

This entry is generated by the terminal control module used to send data to a 3270 display acting as the virtual console.

38 On entry to the module.

Field A The address of the data sent to the terminal.

Field B The first four characters of the terminal data.

Resource The string `SND`, followed by the terminal ID.

EFHTCR entry

This entry is generated by the terminal control module used to receive data from a 3270-type display acting as the virtual console.

39 On entry to the module

Field A The address of the data received from the terminal.

Field B The first 4 characters of the terminal data.

Resource The string RCV, followed by the terminal ID.

EFHFCP entries

These entries are generated by the part of CICS/CMS file control that handles all non-browse requests.

40 Read, or read for update, request. The request byte in the previous file control trace entry will tell you which of the two request types it is.

Field A Address of read buffer area within either the VSWA or the FWA, depending on the request options used.

Field B Number of the record being read.

Resource The name of the file.

41 Rewrite

Field A Address of write buffer area within the FWA.

Field B Number of record being updated.

Resource The name of the file.

42 Write an index record.

Field A Address of write buffer area within the FWA.

Field B The first byte is the request byte (write, rewrite, or delete). The other three bytes are irrelevant.

Resource The name of the file.

43 Write new record.

Field A Address of write buffer area within the FWA.

Field B Number of record being written.

Resource The name of the file.

44 Compare time stamps in EFHVDATA and EFHVINDX files of a CICS/CMS keyed file.

Field A The first 4 bytes of the EFHVINDX file time stamp buffer contents.

Field B The first 4 bytes of the EFHVDATA file time stamp buffer contents.

Resource The name of the file.

45 Write a new time stamp.

Field A The address of the time stamp to be written to the EFHVINDX file.

Field B The address of the time stamp to be written to the EFHVDATA file.

Resource The name of the file.

EFHTDINP entry

This entry is generated by the CICS/CMS module that processes requests for intrapartition transient data.

48 Entry to EFHTDINP.

Field A The intrapartition transient data destination ID.

Field B Only the first byte is relevant. It's the type of transient data request issued.

Resource The string `EFHTDINP`.

EFHTDEXP entry

This entry is generated by the CICS/CMS module that processes requests for extrapartition transient data.

49 Entry to EFHTDEXP.

Field A The extrapartition transient data destination ID.

Field B Only the first byte is relevant. It's the type of transient data request issued.

Resource The string `EFHTDEXP`.

EFHISP entries

These entries are generated by the CICS/CMS module that handles communications between CICS/CMS and a remote CICS/VS system.

50 Entry to EFHISP.

Field A The contents of ISCRQTR, which gives the type of request passed to EFHISP. This will normally be 4, for a converse request.

Field B All zeroes.

Resource The string `EFHISPen`.

51 Return because the link specified in `SYSID` does not exist.

Field A The contents of TCATPLRC (that is, the return code from the terminal control locate function).

Field B All zeroes.

Resource The string `EFHISPnl`

52 Return because the link is not a system entry.

Field A The contents of TCTTETT (that is, the terminal type).

Field B All zeroes.

Resource The string `EFHISPns`.

53 Return on no reply.

Field A All zeroes.

Field B All zeroes.

Resource The string `EFHISPnr`.

54 Normal return.

Field A All zeroes.

Field B All zeroes.

Resource The string `EFHISPrt`.

EFHXVCOM entry

This entry is generated by the CICS/CMS module that transmits data between a PC or host-connected terminal and a CICS/VS system.

55 On entry to the module.

Field A The first bit indicates whether the terminal is a PC (1) or a host-connected terminal (0). The rest of the field is irrelevant.

Field B All zeroes.

Resource The string `EFHXVCOM`.

EFHCPIO entry

This entry is generated by the CICS/CMS module that emulates the CPIO component of VM/PC for the remote server.

5A On entry to the module.

Field A The function.

Only the first bit is relevant. This is 0 (if the logical device module (EFHLDSF) has called EFHCPIO for initialization or termination of the remote server), or 1 (if the communications module (EFHXVCOM) has called EFHCPIO to send something to the remote system).

Field B All zeroes.

Resource The string `EFHCPIO`.

EFHICP entries

These entries are generated by the CICS/CMS interval control program.

60 Start transaction.

Field A The transaction ID.

Field B The terminal ID.

Resource The `REQID`.

61 `RETRIEVE/RETRY`.

Field A The transaction ID.

Field B The terminal ID.

Resource The `REQID` value in the interval control command issued.

62 Cancel START request (with data).

Field A Transaction ID.

Field B The address of the retrieve data element for this request (mapped by the control block, EFHICRDE).

Resource The REQID value in the interval control command issued.

63 Cancel START request (without data).

Field A The transaction ID.

Field B The terminal ID.

Resource The REQID value in the interval control command issued.

EFHSPP entries

These entries are generated by the CICS/CMS syncpoint program.

65 SPP entry.

Field A The current task level (bytes 1 and 2), and program level (bytes 3 and 4).

Field B The address of the current main task control area (TCA).

Resource The string SPPENTRY.

66 SPP exit.

Field A The current task level (bytes 1 and 2), and program level (bytes 3 and 4).

Field B The address of the current main task control area (TCA).

Resource The string SPP EXIT.

EFHFCPT entries

These entries are generated by the CICS/CMS file control program termination module.

68 Entry to file control task termination.

Field A All zeroes.

Field B All zeroes.

Resource The string FCPTENTR.

69 Unlock data set (that is, the task is ending without issuing an UNLOCK, either explicitly or implicitly).

Field A Address of the FCT entry.

Field B Address of either the VSWA or the FWA active for the update request outstanding. Whether it's the VSWA or FWA depends on the request options used.

Resource The data set name.

6A Exit from file control task termination.

Field A All zeroes.

Field B All zeroes.

Resource The string FCPTEXIT.

EFHFCPI entry

This entry is generated by the CICS/CMS program that initializes file control.

6C Entry to EFHFCPI.

Field A All zeroes.

Field B All zeroes.

Resource The string FCP INIT.

EFHTSPI entry

This entry is generated by the CICS/CMS program that initializes temporary storage.

6D Entry to EFHTSPI.

Field A All zeroes.

Field B All zeroes.

Resource The string TS INIT.

EFHTDPI entry

This entry is generated by the CICS/CMS program that initializes transient data.

6E Entry to EFHTDPI.

Field A All zeroes.

Field B All zeroes.

Resource The string `TDP INIT`.

EFHDLII entry

This entry is generated by the CICS/CMS program that initializes the DL/I interface.

6F Entry to EFHDLII.

Field A All zeroes.

Field B All zeroes.

Resource The string `DLI INIT`.

EFHFCP2 entries

These entries are generated by the CICS/CMS module that handles all file control browse requests.

70 Entry to EFHFCP2.

Field A The first byte is the request value; the other 3 bytes are not relevant. You'll find a description of this field in the *CICS/VS Problem Determination Guide.*

Field B The address of the file control table entry (FCTE).

Resource The string `FCP2ENTR`.

71 An `EXEC CICS READPREV` on a nonkeyed file.

Field A The address of the read buffer area within the FWA.

Field B The number of the record being read.

Resource The name of the file being browsed.

72 An `EXEC CICS READNEXT` on a nonkeyed file.

Field A The address of the read buffer area within the FWA.

Field B The number of the record being read.

Resource The name of the file being browsed.

73 An `EXEC CICS READPREV` on a keyed file.

Field A The address of the read buffer area within the FWA.

Field B The number of the record being read from the EFHVDATA file.

Resource The name of the file being browsed.

74 An `EXEC CICS READNEXT` on a keyed file.

Field A The address of the read buffer area within the FWA.

Field B The number of the record being read from the EFHVDATA file.

Resource The name of the file being browsed.

75 An invalid record has been read in a `READNEXT` or `READPREV` on a keyed file.

Field A The address of the read buffer area within the FWA.

Field B The number of the invalid record.

Resource The name of the file being browsed.

76 The FWA and VSWA in use in a browse operation.

Field A The address of the FWA.

Field B The address of the VSWA.

Resource The name of the file being browsed.

77 Invalid browse request.

Field A The first byte is the request value; the other 3 bytes are not relevant. You'll find a description of this field in the *CICS/VS Problem Determination Guide.*

Field B The first byte is the request modifier; the other 3 bytes are not relevant. You'll find a description of this field in the *CICS/VS Problem Determination Guide.*

Resource The name of the file being browsed.

EFHCCMS entries

These entries are generated by the CICS/CMS control transaction, CCMS.

90 Entry to the module.

Field A All zeroes.

Field B All zeroes.

Resource The string `CMND BEG`.

91 Exit from the module.

Field A All zeroes.

Field B All zeroes.

Resource The string `CMND END`.

EFHTCSP entry

This entry is generated by the terminal control module that sends data to a terminal printer.

9A Entry to the module.

Field A The 3270 command issued.

Field B All zeroes.

Resource The string `EFHTCSP`.

EFHTCPW entry

This entry is generated by the terminal control printer support module for processing write control characters (WCCs).

9C Entry to the module.

Field A The WCC character.

Field B All zeroes.

Resource The string `EFHTCPW`.

Index

Special Characters

A

B

C

Q

R

U

V

W

X

Z

Numerics

Customer Information Control System/
Conversational Monitor System
(CICS/CMS)
Release 1
User's Guide

READER'S COMMENT FORM

Order No. SC33-0285-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication . . .

Note: *Staples can cause problems with automated mail-sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.*

If you want an acknowledgement, give your name and address below.

Name . . . . . .

Job Title . . . . . Company . . . . . .

Address. . . . . .

. . . . . Zip . . . . . .

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC33-0285-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

Customer Information Control System/
Conversational Monitor System
(CICS/CMS)
Release 1
User's Guide

READER'S COMMENT FORM

Order No. SC33-0285-0

This manual is part of a library that serves as a reference source for systems analysts, programmers, and operators of IBM systems. You may use this form to communicate your comments about this publication, its organization, or subject matter, with the understanding that IBM may use or distribute whatever information you supply in any way it believes appropriate without incurring any obligation to you. Your comments will be sent to the author's department for whatever review and action, if any, are deemed appropriate.

Note: *Copies of IBM publications are not stocked at the location to which this form is addressed. Please direct any requests for copies of publications, or for assistance in using your IBM system, to your IBM representative or to the IBM branch office serving your locality.*

Number of your latest Technical Newsletter for this publication . . .

Note: *Staples can cause problems with automated mail-sorting equipment. Please use pressure-sensitive or other gummed tape to seal this form.*

If you want an acknowledgement, give your name and address below.

Name . . . . . .

Job Title . . . . . Company . . . . . .

Address. . . . . .

. . . . . Zip . . . . . .

Thank you for your cooperation. No postage stamp necessary if mailed in the U.S.A. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.)

SC33-0285-0

Reader's Comment Form

Fold and tape | Please Do Not Staple | Fold and tape

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 40 ARMONK, N.Y.

POSTAGE WILL BE PAID BY ADDRESSEE:

International Business Machines Corporation
Department 6R1H,
180 Kost Road,
Mechanicsburg, PA 17055, USA

Fold and tape | Please Do Not Staple | Fold and tape

IBM®

SC33-0285-0

Printed in the USA.