

CICS for MVS/ESA



Customization Guide

Version 4 Release 1

CICS for MVS/ESA



Customization Guide

Version 4 Release 1

Note!

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

Second edition (April 1997)

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

This is the second edition of the Customization Guide for CICS/ESA 4.1. It is based on the first edition, SC33-1165-00, which is now obsolete. Changes from the first edition are marked by the '+' sign to the left of the changes. The vertical lines in the left-hand margins indicate changes made between the CICS/ESA 3.3 edition and the CICS/ESA 4.1 first edition.

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

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

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

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

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

© Copyright International Business Machines Corporation 1977, 1997. All rights reserved.

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Notices	ix
Programming interface information	ix
Trademarks and service marks	x
Preface	xi
What this book is about	xi
Who this book is for	xi
What you need to know to understand this book	xi
How to use this book	xi
Notes on terminology	xi
Syntax notation and conventions used in this book	xii
Determining if a publication is current	xii
Bibliography	xiv
CICS/ESA 4.1 library	xiv
Other CICS books	xv
Books from related libraries	xv
Summary of changes	xix
Changes for the CICS/ESA 4.1 second edition	xix
Changes for the CICS/ESA 4.1 first edition	xix
Changes for the CICS/ESA 3.3 edition	xx

Part 1. Customizing with user exit programs 1

Chapter 1. Global user exit programs	3
Overview — what is a global user exit?	3
Global user exit programs	4
The global user exit points	17
Chapter 2. Task-related user exit programs	201
Introduction to the task-related user exit mechanism (the adapter)	201
The stub program	202
The task-related user exit program	204
Adapter administration	229
Chapter 3. The user exit programming interface (XPI)	231
Overview	231
General form of an XPI call	234
Global user exit XPI examples, showing the use of storage	240
The XPI functions	247
Dispatcher functions	249
Dump control functions	263
Kernel domain functions	267
Loader functions	268
Monitoring functions	276
Program management functions	280
State data access functions	297
Storage control functions	305
Trace control function	313

	Transaction management functions	314
	User journaling function	330
<hr/>		
Part 2. Customizing with initialization and shutdown programs		333
	Chapter 4. Writing initialization and shutdown programs	335
	Initialization programs	335
	Shutdown programs	337
	General considerations	338
<hr/>		
Part 3. Customizing with user-replaceable programs		341
	Chapter 5. General notes about user-replaceable programs	343
	Chapter 6. Writing a program error program	351
	Chapter 7. Writing a transaction restart program	357
	Chapter 8. Writing a terminal error program	361
	Background to error handling for TCAM and sequential devices	361
	The sample terminal error program	363
	User-written terminal error programs	384
	Chapter 9. Writing a node error program	395
	Background to CICS-VTAM error handling	396
	When an abnormal condition occurs	403
	The sample node error program	411
	User-written node error programs	421
	Using the node error program with XRF or persistent sessions	427
	Using the node error program with VTAM generic resources	429
	Chapter 10. Writing a program to control autoinstall of terminals	431
	Preliminary considerations	431
	The autoinstall control program at INSTALL	433
	The autoinstall control program at DELETE	441
	Naming, testing, and debugging your autoinstall control program	442
	The sample programs and copy books	444
	Chapter 11. Writing a program to control autoinstall of APPC connections	451
	Preliminary considerations	451
	The autoinstall control program at INSTALL	453
	The autoinstall control program at DELETE	456
	The sample autoinstall control program for APPC connections	456
	Chapter 12. Writing a program to control autoinstall of shipped terminals	459
+	Installing shipped terminals and connections	459
+	The autoinstall control program at INSTALL	461
+	The autoinstall control program at DELETE	464
+	Default actions of the sample programs	465

Chapter 13. Writing a program to control autoinstall of programs	467
Preliminary considerations	467
Benefits of autoinstall	469
Requirements for autoinstall	469
The autoinstall control program at INSTALL	470
The sample autoinstall control program for programs, DFHPGADX	472
Chapter 14. Writing a dynamic transaction routing program	477
The dynamic transaction routing program, DFHDYP	477
Parameters passed to the dynamic transaction routing program	484
Naming your dynamic routing program	491
Testing your dynamic transaction routing program	491
Dynamic transaction routing sample programs	492
Chapter 15. The user-replaceable journaling programs	493
Chapter 16. Writing a CICS–DBCTL interface status program	497
<hr/>	
Part 4. Customizing the XRF overseer program	499
A general note about user-written programs	500
Chapter 17. The extended recovery facility overseer program	501
The sample overseer program	501
The DFHWOSM macros	506
Customizing the sample overseer program	516
<hr/>	
Part 5. CICS journaling, monitoring, and statistics	519
A general note about user-written programs	520
Chapter 18. CICS journaling	521
Formatting a journal before output	521
Opening, closing, and reading journals	522
The structure and contents of journal records	523
Notes on CICS journaling	537
Chapter 19. CICS monitoring	539
Introduction to CICS monitoring	539
CICS monitoring record formats	547
Data produced by CICS monitoring	557
Chapter 20. CICS statistics	579
Introduction to CICS statistics	579
CICS statistics record format	583
Global user exit in the CICS statistics domain	588
Processing the output from CICS statistics	588
<hr/>	
Part 6. Customizing CICS compatibility interfaces	589
A general note about user-written programs	590
Chapter 21. Using TCAM with CICS	591
CICS with TCAM SNA	592

	The TCAM application program interface	594
	The CICS–TCAM interface	596
	TCAM devices	604
	TCAM user exits	607
	Starting and terminating TCAM	607
	CICS and TCAM: program interrelationship	609
	 Chapter 22. The dynamic allocation sample program	613
	Installing the program and transaction definitions	614
	Terminal operation	614
	Help feature	614
	Values	615
	The flow of control when a DYNALLOC request is issued	616
<hr/>		
	Part 7. Customizing CICS security processing	619
	General notes	620
	 Chapter 23. Invoking a user-written external security manager	621
	An overview of the CICS–ESM interface	621
	The MVS router	621
	How ESM exit programs access CICS-related information	624
	CICS security control points	627
	Early verification processing	629
#	 Chapter 24. Changes to link security for MRO	633
	 Chapter 25. Writing a “good night” program	635
	The sample “good night” program, DFH0GNIT	637
<hr/>		
	Part 8. Examining and modifying resource attributes	641
	A general note about user-written programs	642
	 Chapter 26. User programs for the system definition utility program (DFHCSDUP)	643
	An overview of DFHCSDUP	643
	DFHCSDUP as a batch program	644
	Invoking DFHCSDUP from a user program	653
	 Chapter 27. The programmable interface to the RDO transaction, CEDA	663
	Using DFHEDAP in a DTP environment	664
<hr/>		
	Part 9. Appendixes	667
	 Appendix A. Coding entries in the VTAM LOGON mode table	669
	TYPETERM device types and pointers to related LOGON mode data	671
	VTAM MODEENT macro operands	674
	PSERVIC screen size values for LUTYPE0, LUTYPE2, and LUTYPE3 devices	679
	Matching models and LOGON mode entries	680
	LOGON mode definitions for CICS-supplied autoinstall models	692
	 Appendix B. Default actions of the node abnormal condition program	697

	Default actions for terminal error codes	697
	Default actions for system sense codes	709
	Action flag settings and meanings	711
	Appendix C. Transient data write-to-terminal program (DFH\$TDWT) . . .	713
	Appendix D. Uppercase translation of national characters	715
	Using the XZCIN exit	715
	Using DFHTCTDY	715
	Appendix E. The example program for the XTSERREQ global user exit,	
	DFH\$XTSE	717
	Index	733

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

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

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A..

This book contains sample programs. Permission is hereby granted to copy and store the sample programs into a data processing machine and to use the stored copies for study and instruction only. No permission is granted to use the sample programs for any other purpose.

Programming interface information

This book is intended to help you to customize your CICS/ESA 4.1 system. This book primarily documents Product-sensitive Programming Interface and Associated Guidance Information provided by CICS.

Product-sensitive programming interfaces allow the customer installation to perform tasks such as diagnosing, modifying, monitoring, repairing, tailoring, or tuning of CICS. Use of such interfaces creates dependencies on the detailed design or implementation of the IBM software product. Product-sensitive programming interfaces should be used only for these specialized purposes. Because of their dependencies on detailed design and implementation, it is to be expected that programs written to such interfaces may need to be changed in order to run with new product releases or versions, or as a result of service.

However, this book also documents General-use Programming Interface and Associated Guidance Information.

General-use programming interfaces allow the customer to write programs that request or receive the services of CICS.

General-use Programming Interface and Associated Guidance Information is identified where it occurs, either by an introductory statement to a chapter or section or by the following marking:

```
_____ General-use programming interface _____  
  
General-use Programming Interface and Associated Guidance Information...  
  
_____ End of General-use programming interface _____
```

Trademarks and service marks

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

ACF/VTAM	AIX
C/370	CICS
CICS/ESA	DB2
ESA/370	IBM
IBMLink	IMS/ESA
Language Environment	MVS/ESA
RACF	Resource Measurement Facility
RMF	SAA
System/370	VTAM

Preface

What this book is about

This book provides the information needed to extend and modify an IBM CICS/ESA 4.1 system to match your requirements. It describes how you can tailor your system by coding exit programs, by replacing specific CICS-supplied default programs with versions that you write yourself, and by adapting sample programs.

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

Who this book is for

This book is for those responsible for extending and enhancing a CICS system to meet the special processing needs of an installation.

What you need to know to understand this book

To use the information in this book, you need to be familiar with some of the architecture of CICS and the programming interface to CICS. General-use programming interface information is given in the *CICS/ESA Application Programming Reference* manual and the *CICS/ESA System Programming Reference* manual.

Resource definition information is in the *CICS/ESA Resource Definition Guide*.

To use the following chapters you need to be familiar with the telecommunications access methods (ACF/VTAM and TCAM):

- “Chapter 8, Writing a terminal error program”
- “Chapter 9, Writing a node error program”
- “Chapter 10, Writing a program to control autoinstall of terminals”
- “Chapter 21, Using TCAM with CICS”.

If your task involves error processing, you may need to consult the *CICS/ESA Messages and Codes* manual, the *CICS/ESA User's Handbook*, the *CICS/ESA Problem Determination Guide*, or the *CICS/ESA Diagnosis Reference* manual.

How to use this book

The parts and chapters of the book are self-contained. Use an individual part or chapter as a guide when performing the task described in it.

Notes on terminology

In this book, the term VTAM refers to ACF/VTAM. The term TCAM refers to the DCB interface of ACF/TCAM. The term APPC (advanced program-to-program communication) refers to the LUTYPE6.2 intersystem connection (ISC) protocol. The term CICS/ESA is an abbreviation for Customer Information Control System/Enterprise Systems Architecture.

CICS/ESA 4.1 supports CICS applications written in:

- Assembler language
- C
- COBOL
- PL/I.

In this book, the phrase “the languages supported by CICS” refers to the above languages.

1 KB equals 1024 bytes.

Syntax notation and conventions used in this book

The symbols { }, [], and | are used in the syntax descriptions of the EXEC CICS commands and macros referred to in this book. They are not part of the command and you should not include them in your code. Their meanings are as follows:

- Braces { } enclose two or more alternatives, one of which you **must** code.
- Square brackets [] tell you that the enclosed is optional.
- The “or” symbol | separates alternatives.

In addition to these symbols, the following conventions apply:

- Punctuation symbols and uppercase characters should be coded exactly as shown.
- Lowercase characters indicate that user text should be coded as required.
- Default values are shown like this: DEFAULT.
- Options that are enclosed neither in braces { } nor in square brackets [] are mandatory.
- The ellipsis ... means that the immediately preceding option can be coded one or more times.
- All EXEC CICS commands require a delimiter appropriate to the language of the application. For a COBOL program this is 'END-EXEC', for example. Delimiters are not included in the syntax descriptions of the commands.

Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager softcopy versions of a publication are in step, but subsequent updates will probably be available in softcopy before they are available in hardcopy.

For CICS Transaction Server books, these softcopy updates appear regularly on the *Transaction Processing and Data Collection Kit* CD-ROM, SK2T-0730-xx. Each reissue of the collection kit is indicated by an updated order number suffix (the -xx part). For example, collection kit SK2T-0730-06 is more up-to-date than SK2T-0730-05. The collection kit is also clearly dated on the cover.

Here's how to determine if you are looking at the most current copy of a publication:

- A publication with a higher suffix number is more recent than one with a lower suffix number. For example, the publication with order number SC33-0667-02

is more recent than the publication with order number SC33-0667-01. (Note that suffix numbers are updated as a product moves from release to release, as well as for hardcopy updates within a given release.)

- When the softcopy version of a publication is updated for a new collection kit the order number it shares with the hardcopy version does not change. Also, the date in the edition notice remains that of the original publication. To compare softcopy with hardcopy, and softcopy with softcopy (on two editions of the collection kit, for example), check the last two characters of the publication's filename. The higher the number, the more recent the publication. For example, DFHPF104 is more recent than DFHPF103. Next to the publication titles in the CD-ROM booklet and the readme files, asterisks indicate publications that are new or changed.
- Updates to the softcopy are clearly marked by revision codes (usually a “#” character) to the left of the changes.

Bibliography

CICS/ESA 4.1 library

Evaluation and planning		
<i>Release Guide</i>	GC33-1161	April 1997
<i>Migration Guide</i>	GC33-1162	April 1997
General		
<i>CICS Family: Library Guide</i>	GC33-1226	April 1995
<i>Master Index</i>	SC33-1187	October 1994
<i>User's Handbook</i>	SX33-1188	April 1997
<i>Glossary (softcopy only)</i>	GC33-1189	n/a
Administration		
<i>Installation Guide</i>	GC33-1163	April 1997
<i>System Definition Guide</i>	SC33-1164	April 1997
<i>Customization Guide</i>	SC33-1165	April 1997
<i>Resource Definition Guide</i>	SC33-1166	April 1997
<i>Operations and Utilities Guide</i>	SC33-1167	April 1997
<i>CICS-Supplied Transactions</i>	SC33-1168	April 1997
Programming		
<i>Application Programming Guide</i>	SC33-1169	October 1994
<i>Application Programming Reference</i>	SC33-1170	April 1997
<i>System Programming Reference</i>	SC33-1171	April 1997
<i>Sample Applications Guide</i>	SC33-1173	October 1994
<i>Distributed Transaction Programming Guide</i>	SC33-1174	October 1994
<i>Front End Programming Interface User's Guide</i>	SC33-1175	October 1994
Diagnosis		
<i>Problem Determination Guide</i>	SC33-1176	October 1994
<i>Messages and Codes</i>	GC33-1177	April 1997
<i>Diagnosis Handbook</i>	LX33-6093	October 1994
<i>Diagnosis Reference</i>	LY33-6082	April 1997
<i>Data Areas</i>	LY33-6083	April 1997
<i>Supplementary Data Areas</i>	LY33-6081	October 1994
<i>Closely-Connected Program Interface</i>	LY33-6084	February 1996
Communication		
<i>Intercommunication Guide</i>	SC33-1181	April 1997
<i>Server Support for CICS Clients</i>	SC33-1591	February 1996
<i>CICS Family: Inter-product Communication</i>	SC33-0824	October 1996
<i>CICS Family: Communicating from CICS on System/390</i>	SC33-1697	October 1996
Special topics		
<i>Recovery and Restart Guide</i>	SC33-1182	October 1994
<i>Performance Guide</i>	SC33-1183	October 1994
<i>CICS-IMS Database Control Guide</i>	SC33-1184	October 1994
<i>CICS-RACF Security Guide</i>	SC33-1185	October 1994
<i>Shared Data Tables Guide</i>	SC33-1186	October 1994
<i>External CICS Interface</i>	SC33-1390	April 1997
<i>CICS ONC RPC Feature for MVS/ESA Guide</i>	SC33-1119	February 1996
<i>CICS Web Interface Guide</i>	SC33-1892	November 1996

The book that you are reading was republished in hardcopy format in April 1997 to incorporate updated information previously available only in softcopy. The right-hand column in the above table indicates the latest hardcopy editions of the CICS/ESA books available in April 1997. A book with a date earlier than April 1997 remains the current edition for CICS/ESA 4.1. Note that it is possible that other books in the library will be updated after April 1997.

When a new order is placed for the CICS/ESA 4.1 product, the books shipped with that order will be the latest hardcopy editions.

The style of IBM covers changes periodically. Books in this library have more than one style of cover.

For information about the softcopy books, see “Determining if a publication is current” on page xii. The softcopy books are regularly updated to include the latest information.

Other CICS books

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

CICS Clients

- *CICS Clients: Administration*, SC33-1436
- *CICS Family: Client/Server Programming*, SC33-1435

Books from related libraries

This section lists the non-CICS books that are referred to in this manual.

ACF/TCAM books

- *ACF/TCAM Installation and Migration Guide*, SC30-3121
- *ACF/TCAM System Programmer's Guide*, SC30-3117
- *ACF/TCAM Version 3 Application Programming*, SC30-3233.

ACF/VTAM books

- *ACF/VTAM Network Implementation Guide*, SC31-6434
- *ACF/VTAM Version 3 Programming*, SC23-0115.

MVS books

Short Title	Version 3 book	Version 4 book
<i>Assembler Programming Guide</i>	<i>MVS/ESA Application Development Guide</i> , GC28-1821	<i>MVS/ESA Application Development Guide: Assembler Language Programs</i> , GC28-1644
<i>Authorized Assembler Programming Guide</i>	<i>MVS/ESA System Programming Library: Application Development Guide</i> , GC28-1852	<i>MVS/ESA Application Development Guide: Authorized Assembler Language Programs</i> , GC28-1645

Short Title	Version 3 book	Version 4 book
<i>Authorized Assembler Programming Reference</i>	<i>MVS/ESA System Programming Library: Application Development Macro Reference, GC28-1857</i>	<i>MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs Volume 1, GC28-1647</i>
<i>Authorized Assembler Programming Reference Volume 2</i>	—	<i>MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs Volume 2, GC28-1648</i>
<i>Authorized Assembler Programming Reference Volume 3</i>	—	<i>MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs Volume 3, GC28-1649</i>
<i>Authorized Assembler Programming Reference Volume 4</i>	—	<i>MVS/ESA Application Development Reference: Services for Authorized Assembler Language Programs Volume 4, GC28-1650</i>
<i>Diagnosis: Data Areas Volume 1</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 1 ABEP-DFTL, LY28-1043</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 1 (non-OCO for JES2 and JES3), LY28-1821</i>
<i>Diagnosis: Data Areas Volume 2</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 2 DIAG-IVT, LY28-1044</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 2 (non-OCO for JES2 and JES3), LY28-1822</i>
<i>Diagnosis: Data Areas Volume 3</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 3 JBT-QWB, LY28-1045</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 3 (non-OCO for JES2 and JES3), LY28-1823</i>
<i>Diagnosis: Data Areas Volume 4</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 4 RAB-SRRA, LY28-1046</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 4 (non-OCO for JES2 and JES3), LY28-1824</i>
<i>Diagnosis: Data Areas Volume 5</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 5 SSAG-XTLST, LY28-1047</i>	<i>MVS/ESA Diagnosis: Data Areas Volume 5 (non-OCO for JES2 and JES3), LY28-1825</i>
—	<i>MVS/ESA Resource Measurement Facility (RMF), Version 4.1.1—Monitor I & II Reference and User's Guide, LY28-1007</i>	—
<i>System Management Facilities</i>	<i>MVS/ESA System Programming Library: System Management Facilities (SMF), GC28-1819</i>	<i>MVS/ESA System Management Facilities (SMF), GC28-1628</i>

Other related books

IBM ESA/370 Principles of Operation, SA22-7200

IMS/ESA Application Programming: DL/I Calls, SC26-4274

+ *RACF External Security Interface (RACROUTE) Macro Reference for MVS and VM, GC28-1366*

Service Level Reporter Version 3 General Information, GH19-6529

SNA Formats, GA27-3136

SNA Sessions Between Logical Units, GC20-1868

System Programming Library: Resource Access Control Facility (RACF),
SC28-1343.

Summary of changes

Changes for the CICS/ESA 4.1 second edition

+ This book is the second edition of the *Customization Guide* for CICS/ESA 4.1.
+ Changes that were made for the first edition are still indicated by vertical bars to
+ the left of the changes. Changes made for this second edition are indicated by the
+ '+' symbol to the left of the changes. Users of the first edition can therefore see
+ what has changed since that first edition was published. Softcopy versions of this
+ book use both these revision indicators and use the '#' symbol to show further
+ changes since this second hardcopy edition of the book was published.

+ These are the most significant changes for this edition:

- + • The following new global user exits are described in "Chapter 1, Global user
+ exit programs:
 - + – XDUREQC, in the dump domain
 - + – XLDLOAD and XLDELETE, in the loader domain.
- + • Information about the XDTAD, XDTLC, and XDTRD global user exits, in the
+ shared data tables management program, has been updated.
- + • "Chapter 12, Writing a program to control autoinstall of shipped terminals" has
+ been added, to reflect the fact that the autoinstall user program is now invoked
+ for shipped terminals, as well as for local terminals and APPC connections.

Changes for the CICS/ESA 4.1 first edition

These were the most significant changes for this edition:

- The following new global user exits were described in "Chapter 1, Global user
exit programs":
 - XAKUSER, in the activity keypoint program
 - XALCAID, in the terminal allocation program
 - XFCNREQ, in the file control open/close program
 - XICEREQ and XICEREQC, in the interval control program
 - XRMIIN and XRMIOUT, in the resource manager interface program
 - XRSINDI, in the resource management modules
 - XSNON and XSNOFF, in the security manager domain
 - XTDEREQ and XTDEREQC, in the transient data program
 - XTSEREQ and XTSEREQC, in the temporary storage control program
 - XXMATT, in the transaction manager domain
 - XZIQUE, in the VTAM working-set module.

Changes to the following existing global user exits were also described:

- XEIIIN and XEIIOUT, in the EXEC interface program
- XKCREQ, in the task control program
- XZCATT, in the VTAM terminal management program.
- New function calls added to the exit programming interface (XPI) were
described in "Chapter 3, The user exit programming interface (XPI)". The calls
are in the following functional areas:
 - Access to CICS state-data
 - Kernel domain

- Program management
- Storage control
- Transaction management.
- “Chapter 11, Writing a program to control autoinstall of APPC connections” was added.
- “Chapter 13, Writing a program to control autoinstall of programs” was added.
- Changes that affect the user-replaceable dynamic transaction routing program, DFHDYP, were described in “Chapter 14, Writing a dynamic transaction routing program”.
- Information about response times and new performance class monitoring records for the Front End Programming Interface (FEPI) was added to “Chapter 19, CICS monitoring”.
- “Chapter 24, Changes to link security for MRO” was rewritten, because the DFHACEE user-replaceable program, previously used to obtain the CICS region’s userid, is withdrawn in CICS/ESA 4.1.
- “Chapter 25, Writing a “good night” program” was added.
- Three new appendixes were added:
 - “Appendix B, Default actions of the node abnormal condition program” contains information relevant to the node error program that was previously in the *CICS/ESA User’s Handbook*.
 - “Appendix D, Uppercase translation of national characters”.
 - “Appendix E, The example program for the XTSEREQ global user exit, DFH\$XTSE”.

Changes for the CICS/ESA 3.3 edition

These were the most significant changes for the CICS/ESA 3.3 edition:

- The following new global user exits were described:
 - XISCONA, in the intersystem communication program, used to control the queuing of function shipping requests
 - XPCREQ and XPCREQC, in the program control program, used with link requests
 - XSZARQ and XSZBRQ, for use with the CICS/ESA Front End Programming Interface (FEPI).

In addition, new function added to the XPCTA exit, in the program control program, was described, as well as new fields, UEPGIND and UEPTIND, in the standard DFHUEPAR parameter lists passed to global and task-related user exits, respectively.

- Changes to the following exit programming interface (XPI) functions were described:
 - The loader function, DEFINE_PROGRAM
 - The monitoring function, MONITOR
 - The storage control function, GETMAIN
 - The trace control function, TRACE_PUT.
- Other new function described included:

- New fields in the communication area passed to the program error program, DFHPEP
- Additional performance class monitoring fields in the system-defined DFHSTOR group.
- “Appendix C, Transient data write-to-terminal program (DFH\$TDWT)” was new for this release.

Part 1. Customizing with user exit programs

Chapter 1. Global user exit programs	3	General form of an XPI call	234
Overview — what is a global user exit?	3	Global user exit XPI examples, showing the use of storage	240
Global user exit programs	4	The XPI functions	247
The global user exit points	17	Dispatcher functions	249
Chapter 2. Task-related user exit programs	201	Dump control functions	263
Introduction to the task-related user exit mechanism (the adapter)	201	Kernel domain functions	267
The stub program	202	Loader functions	268
The task-related user exit program	204	Monitoring functions	276
Adapter administration	229	Program management functions	280
Chapter 3. The user exit programming interface (XPI)	231	State data access functions	297
Overview	231	Storage control functions	305
		Trace control function	313
		Transaction management functions	314
		User journaling function	330

user exit programs

Chapter 1. Global user exit programs

This chapter describes the **global user exit points** of CICS/ESA 4.1, and how you can use them, in conjunction with programs of a special type that you write yourself (**global user exit programs**), to customize your CICS system. The chapter is divided into three sections:

1. **“Overview — what is a global user exit?”** is an introduction to global user exits, describing their main features and what they can be used for.
2. **“Global user exit programs”** on page 4 covers topics that you need to consider when writing a global user exit program. It deals with the following:
 - “Register conventions” on page 4
 - “31-bit addressing implications” on page 5
 - “Using CICS services” on page 5
 - “Using EDF with global user exits” on page 6
 - “The global work area” on page 7
 - “Making trace entries” on page 7
 - “Parameters passed to the global user exit program” on page 7
 - “Returning values to CICS” on page 10
 - “Restrictions on the use of fields as programming interfaces” on page 11
 - “Exit programs and the CICS storage protection facility” on page 11
 - “Errors in user exit programs” on page 12
 - “Defining, enabling, and disabling an exit program” on page 13
 - “Invoking more than one exit program at a single exit” on page 13
 - “Invoking a single exit program at more than one exit” on page 14
 - “Sample global user exit programs” on page 14.
3. **“The global user exit points”** on page 17 contains detailed information about each global user exit point, including the place in the CICS code at which it occurs, and the specific (as distinct from the standard) parameters that are passed to an exit program.

Overview — what is a global user exit?

A global user exit *point* (sometimes referred to simply as a “global user exit”) is a place in a CICS module or domain¹ at which CICS can transfer control to a program that you have written (a global user exit *program*), and at which CICS can resume control when your exit program has finished its work. You do not have to use any of the global user exits, but you can use them to extend and customize the function of your CICS system according to your own requirements. For a complete list of the global user exit points, see pages 18 through 22.

Each global user exit point has a unique identifier, and is located at a point in the module or domain at which it could be useful to do some extra processing. For example, at exit point XSTOUT in the statistics domain, an exit program can be given control before each statistics record is written to the SMF data set, and can access the relevant statistics record. You might want to use an exit program at this

¹ A domain is an isolated functional unit of CICS/ESA 4.1 that communicates with the rest of CICS and with other programs using a set of strictly defined and controlled interfaces.

global user exit programs

exit point to examine the statistics record and suppress the writing of unwanted records.

Global user exit support is provided automatically by CICS. However, there are several conventions that govern how you write your exit program, which are described in “Global user exit programs.” Also in that section is a list of the standard parameters that the calling modules and domains pass to an exit program, and some information about returning values to the caller.

Because global user exit programs work as if they were part of the CICS module or domain, there are limits on the use you can make of CICS services. Most global user exit programs cannot use EXEC CICS commands. By contrast, most global user exit programs can invoke some CICS services using the exit programming interface (XPI). For more information, see “Using CICS services” on page 5.

Note: Neither source nor object compatibility of CICS management modules is guaranteed from release to release. Any changes that affect exit programs are documented in the appropriate manual.

Global user exit programs

A global user exit program must be written in assembler language and must be quasireentrant. However, if your user exit program calls the XPI, it must be fully reentrant.² (For details about coding programs using XPI calls, refer to Chapter 3, “The user exit programming interface (XPI)” on page 231.)

Register conventions

The following register values are provided on entry to an exit program:

- Register 1 contains the address of the user exit parameter list DFHUEPAR.
Write-to-operator (WTO) commands use register 1. If your exit program uses WTO commands, you should save the address of DFHUEPAR first.
- Register 13 contains the address of the standard register save area where your exit program should store its own registers immediately after being invoked. This address is also in the field UEPEPSA in the parameter list pointed to by register 1.
If you want to issue operating system requests that use register 13 to point to a save area, you must switch register 13 to point to another save area. You must restore register 13 to its original contents before returning from your user exit program to the caller.
- Register 14 contains the return address to which the exit program should branch on completion of its work. You do this using the BR 14 instruction after restoring the calling module’s registers, or using the RETURN macro.
- Register 15 contains the entry address of the exit program.

² A “reentrant” program is coded to allow one copy of itself to be used concurrently by several tasks; it does not modify itself while running. A “quasireentrant” program is serially reusable by different tasks. When it receives control it must be in the same state as when it relinquished control. Such a program can modify itself while running, and is therefore not fully reentrant.

No other register values are guaranteed, and they should not be relied on. The exit program should save and restore any registers that it modifies, using the save area addressed by register 13.

31-bit addressing implications

- The global user exit is invoked in 31-bit AMODE.
- The global user exit may be either RMODE 24 or RMODE ANY.
- If you find it necessary to switch to 24-bit AMODE in the exit program, be sure to return correctly in 31-bit AMODE.
- Ensure the exit program is in 31-bit AMODE for XPI calls.
- Some of parameters passed in DFHUEPAR are addresses of storage above the 16MB line.

Access register implications:

- The global user exit is invoked in primary-space translation mode. For information about translation modes, see the *IBM ESA/370 Principles of Operation* manual.
- The contents of the access registers are unpredictable. For information about access registers, see the *IBM ESA/370 Principles of Operation* manual.
- If the global user exit modifies any access registers, it must restore them before returning control. CICS does **not** provide a save area for this purpose.
- The global user exit must return control in primary addressing mode.

Using CICS services

The rules governing the use of CICS services in exit programs vary, depending on the exit point from which the exit program is being invoked. The following general rules apply:

- No CICS services can be invoked from any exit point in the dispatcher domain or the journal control program.
- CICS services can be invoked using the exit programming interface (XPI) from most exits. If you use the XPI, note the rules and restrictions that are listed for each exit and each of the XPI macros. The XPI is described in Chapter 3, “The user exit programming interface (XPI)” on page 231.
- Some CICS services can be requested using EXEC CICS commands from some exits. The valid commands are listed in the detailed descriptions of the exits. If no commands are listed, it means that no EXEC CICS API or SPI commands are supported.

|
|
|
|

#

An exit program invoked at an exit that does not support the use of EXEC CICS commands should not call a task-related user exit program (TRUE). (Calling a TRUE is equivalent to issuing an EXEC CICS command.) TRUEs are described in Chapter 2, “Task-related user exit programs” on page 201.

Note: In exits which support the use of EXEC CICS File Control commands, file commands that form a related sequence (such as EXEC CICS STARTBR, EXEC CICS READNEXT, and EXEC CICS ENDBR) should all be issued in the same invocation of the exit program.

global user exit programs

For example, if one invocation of an exit program issues an EXEC CICS
STARTBR command, and the next invocation of the exit program for
that same task issues an EXEC CICS READNEXT command, the
READNEXT fails with an INVREQ condition.

- All exit programs that issue EXEC CICS commands must first address the EIB. This is **not** done automatically via the DFHEIENT macro, as is the case with normal EXEC assembler-language programs. Therefore, the first EXEC command to be issued from an exit program must be EXEC CICS ADDRESS EIB (eib-register), where “eib-register” is the default register (R11) or the register given as a parameter to the DFHEIENT macro.

All exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See “Returning values to CICS” on page 10.

Important

- If your global user exit program does *not* contain EXEC CICS commands, **do not** use the CICS command-level translator when assembling the program.
- Do not make non-CICS (for example, RACF or MVS) system service calls from global user exit programs.
- If an operating system request causes a wait, your whole CICS system will stop until the operating system request has been serviced.

Using EXEC CICS and XPI calls in the same exit program

There are a number of exits where you can use both EXEC CICS commands and XPI calls, but you should ensure that there is no conflict on the usage of register 13. To avoid such conflict, use the DATAREG option on the DFHEIENT macro (see “XPI register usage” on page 239 for information).

For an example of how to use EXEC CICS commands and XPI calls in the same global user exit program, see Appendix E, “The example program for the XTSEREQ global user exit, DFH\$XTSE” on page 717.

Using EDF with global user exits

If you use the Execution Diagnostic Facility (EDF) to monitor your applications, you must take care when compiling exit programs that issue EXEC CICS commands.

Normally, if an exit program issues EXEC CICS commands, these are displayed by EDF, if the latter is active. They appear between the “Start of Command” and “End of Command” screens for the command that caused the exit to be driven. If you want to suppress the display of EXEC CICS commands issued by your exit program, you must specify the NOEDF option when you translate the program.

Exits in the recovery modules

If an exit program invoked from one of the recovery modules listed in “Using CICS services” on page 5 issues EXEC CICS commands, you must translate it with the NOEDF option. Failure to do so may cause EDF to abend.

The global work area

When you enable an exit program, you can ask CICS to provide a global work area for the exit program. An exit program can have its own global work area, or it can share a work area that is owned by another exit program. Note that the work area is associated with the exit **program** rather than with the exit **point**. For ease of problem determination, the global work area should be shared only by exit programs that are invoked from the same management module or domain. The address and length of the global work area are addressed by parameters UEPGAA and UEPGAL of the DFHUEPAR parameter list, which is described in “DFHUEPAR standard parameters” on page 8. If a user exit program does not own a global work area, UEPGAA is set to zero.

Application programs can communicate with user exit programs that use or share the same global work area. The application program uses the EXEC CICS EXTRACT EXIT command to obtain the address and length of the global work area.

A work area is freed only when all of the exit programs that use it are disabled. For examples of how to use a global work area, see the sample global user exit programs. They are listed in “Sample global user exit programs” on page 14.

Making trace entries

If tracing is active, an entry in the CICS trace table can be made immediately before and immediately after the execution of an exit program. To specify that these entries are to be made, use the UE option of either:

- The CETR transaction
- The EXEC CICS SET TRACETYPE command.

For global user exits in domains, extra trace calls giving more information are also available if you have set the AP option of EXEC CICS SET TRACETYPE to level 1 or 2. For information about trace entries, refer to the *CICS/ESA Problem Determination Guide*.

In some cases, when tracing is active, you can also make trace entries from within a user exit program, using the XPI DFHTRPTX TRACE_PUT macro described in Chapter 3, “The user exit programming interface (XPI)” on page 231. The individual descriptions of the global user exit points show whether the XPI DFHTRPTX macro can be used at each point.

Parameters passed to the global user exit program

The address of a parameter list is passed to the user exit program in register 1. The list contains some standard parameters that are passed to all global user exit programs, and may also contain some exit-specific parameters that are unique to the exit point from which the exit program is being invoked. Not all of the exit points have these extra parameters.

The exit-specific parameters are described with the individual exits in the section “The global user exit points” on page 17. The standard parameter list is described in the following section.

You can map the parameter list using the DSECT DFHUEPAR, which is generated by the macro instruction

global user exit programs

```
DFHUEXIT TYPE=EP,ID=exit_point_identifier
```

The ID parameter provides the extra data definitions that you need to map any exit-specific parameters. For example, the macro instruction

```
DFHUEXIT TYPE=EP,ID=XJCWR
```

generates the DSECT to map the standard parameters followed by the parameters that are specific to exit point XJCWR in the journal control program. If your exit program is to be invoked at more than one exit point, you can code up to 256 characters of relevant exit identifiers on a single DFHUEXIT macro instruction. For example:

```
DFHUEXIT TYPE=EP,ID=(XJCWR,XMNOU,XSTOUT)
```

If your exit program is to be invoked at every global user exit point, you can code:

```
DFHUEXIT TYPE=EP,ID=ALL
```

If your user exit program is to be used both as a global user exit program and as a task-related user exit program, you must code both:

```
DFHUEXIT TYPE=EP,ID=exit_point_identifier
```

and:

```
DFHUEXIT TYPE=RM
```

(in this order) to generate the DSECTs appropriate to both types of user exit.

+

APAR PN 86384

+

Documentation for APAR PN 86384 added 17th July 1996.

+

+

If a global user exit program needs to use the DFHRMCAL macro to invoke an external RMI, the DFHRMCAL macro instruction must follow the DFHUEXIT macro.

DFHUEPAR standard parameters

```
DFHUEPAR DSECT
```

```
* STANDARD PARAMETERS
```

UEPEXN	DS	A	ADDRESS OF EXIT NUMBER
UEPGAA	DS	A	ADDRESS OF GLOBAL WORK AREA
*			(ZERO = NO WORK AREA)
UEPGAL	DS	A	ADDRESS OF GLOBAL WORK AREA LENGTH
UEPCRC	DS	A	ADDRESS OF CURRENT RETURN-CODE
UEPTCA	DS	A	RESERVED
UEPCSA	DS	A	RESERVED
UEPEPSA	DS	A	ADDRESS OF REGISTER SAVE AREA
*			FOR USE BY EXIT PROGRAM
UEPHMSA	DS	A	ADDRESS OF SAVE AREA USED FOR
*			HOST MODULE'S REGISTERS
UEPGIND	DS	A	ADDRESS OF CALLER'S TASK INDICATORS
UEPSTACK	DS	A	ADDRESS OF KERNEL STACK ENTRY
UEPXSTOR	DS	A	ADDRESS OF STORAGE FOR XPI PARAMETERS
UEPTRACE	DS	A	ADDRESS OF TRACE FLAG

UEPEXN

points to a 1-byte binary field whose contents identify the global user exit point from which the exit program is being invoked. You need this information if your exit program can be invoked from more than one exit point.

DFHUEXIT TYPE=EP generates a list of equated values that relate the exit names (exitids) to the exit numbers used internally by CICS to identify the exits. You should always use the exitids, because the exit numbers may change in any future releases of CICS.

UEPGAA

points to the global work area that was provided for the exit program when it was enabled. This is set to zero if no global work area is provided.

UEPGAL

points to a halfword that contains the length of the global work area.

UEPCRCA

points to a halfword that is to contain the return code value from the exit program. When more than one program is called at a user exit, this field contains (on entry to the second and subsequent programs) the return code that was set by the previously invoked program.

UEPTCA

points to fetch-protect storage. Use of this field results in an abend ASRD at execution time.

UEPCSA

points to fetch-protect storage. Use of this field results in an abend ASRD at execution time.

UEPEPSA

points to a save area in which the exit program should store its own registers on entry. When the exit program is entered, register 13 is also pointing to this area. The convention is to save registers 14, 15, 0–12 at offset 12 (decimal) onward.

UEPHMSA

points to the save area containing the registers of the calling module. Values for registers 14, 15, 0–13 are stored in this order from offset 12 (decimal) in this area.

Apart from register 15, which contains the return code value from the exit program, the values in this save area are used by CICS to reload the registers when returning to the calling CICS module. They should not be corrupted.

This address is **not** passed to global user exit programs invoked from exit points in CICS domains.

UEPGIND

points to a 1-byte field containing indicator bits for use in AP domain user exits. The indicators are represented by two symbolic values, UEPGANY and UEPGCICS, which you can test to determine whether data locations can be above or below the 16MB line and whether the application's storage is in CICS-key or user-key storage. For non-AP domain user exits, the indicator bits are always zero.

UEPGANY The application can accept addresses above the 16MB line. If the symbolic value is not UEPGANY, the application must be returned an address below the 16MB line.

UEPGCICS The application's working storage and the task's life-time storage are in CICS-key storage (TASKDATAKEY=CICS). If the symbolic value is not UEPGCICS, the application's working storage and the task's life-time storage are in user-key storage (TASKDATAKEY=USER).

UEPSTACK

points to the kernel stack entry. This value must be moved to the exit program's register 13 before invoking the XPI. For more information, refer to Chapter 3, "The user exit programming interface (XPI)" on page 231. The storage addressed by this field **must not be altered**. If it is corrupted, your exit program will have unpredictable effects on your CICS system.

UEPXSTOR

points to a 260-byte area of DFHUEH-owned LIFO storage that the exit program should use when invoking the XPI. For more information, refer to Chapter 3, "The user exit programming interface (XPI)" on page 231.

UEPTRACE

points to the trace flag, which indicates whether tracing is on in the calling management module or domain. This enables you to control your use of the XPI TRACE_PUT macro in line with the tracing in the CICS module or domain. The XPI TRACE_PUT function should be used only when tracing is on. The trace flag is a single byte, whose top bit is set on when tracing is switched on. You test this setting using the symbolic value UEPTRON. The rest of the byte addressed by UEPTRACE is reserved, and its contents should not be corrupted.

Returning values to CICS

At some exit points, you can influence what CICS does on return from an exit program by supplying a return code value. The return code value must be set in register 15 before leaving the exit program. Character strings equating to valid return code values are provided with the parameter list appropriate for each exit point. Always use the equated values rather than using hard-coded values. For example, at exit XMNOUT in the monitor domain, you are presented with the address of a monitoring record. If you decide in your exit program that this record should not be written to SMF, you can set the return code value UERCBYP (meaning "bypass this record") before returning to CICS, and CICS suppresses the record.

You cannot influence CICS actions in this way at all exit points. If you supply a return code value that is not expected at a particular exit point, the default return code indicating a normal response (usually UERCNORM) is assumed, unless the return code UERCPURG is set (see note below about UERCPURG). You are strongly advised not to let the return code default to the normal response as the result can be unpredictable. The normal response tells CICS to continue processing as if the exit program had not been invoked, and it is a valid option at most global user exit points. The exceptions are shown in the list of return codes provided with each exit description.

The return code currently established for an exit is addressed by parameter UEPCRA of DFHUEPAR, and it is needed when two or more exit programs are used at one exit. For more information, see “Invoking more than one exit program at a single exit” on page 13.

The return codes that are valid at each of the global user exit points are described in “The global user exit points” on page 17.

Important

- At some exit points, the return code UERCPURG is valid. These exits are identified in the following tables. To prevent unpredictable results, you must **not** set the return code UERCPURG except as described on page 238.
- Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. The DFHEIRET macro:
 - Restores registers
 - Places a return code in register 15 after the registers are restored
 - Returns control to the address in register 14.

For example:

```
DFHEIRET RCREG=nn
```

where “nn” is the number of any register (other than 13) that contains the return code to be placed in register 15 after the registers are restored.

Restrictions on the use of fields as programming interfaces

The *CICS/ESA Data Areas* manual contains definitions of the control block fields that form part of the Product-sensitive and General-use programming interfaces of CICS. Fields that are **not** defined in the *CICS/ESA Data Areas* manual as either Product-sensitive programming interface or General-use programming interface fields are **not** intended for your use as part of a CICS programming interface.

Exit programs and the CICS storage protection facility

When you are running CICS with the storage protection facility, there are two points you need to consider for global user exits:

1. The execution key in which your user exit programs run
2. The storage key of data storage obtained for your exit programs.

Execution key for global user exit programs

When you are running with storage protection active, CICS always invokes global user exit programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to the exit program. However, if a global user exit program itself passes control to another program (via a link or transfer-control command), the program thus invoked is executed according to the execution key (EXECKEY) defined in its program resource definition.

Important

You are strongly recommended to specify EXECKEY(CICS) when defining both global user exit programs and programs to which an exit program passes control.

Data storage key for global user exit programs

The storage key of storage used by global user exit programs depends on how the storage is obtained:

- The CICS-supplied storage addressed by the UEPXSTOR parameter of DFHUEPAR, and any global work area specified when an exit program is enabled, are always in CICS key.
- Global user exit programs that can issue EXEC CICS commands can obtain storage by:
 - Explicit EXEC CICS GETMAIN commands
 - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the exit program is invoked.

As an example, consider a transaction defined with TASKDATAKEY(USER) that issues a file control request, which causes an XFCREQ global user exit program to be invoked. In this case, any implicit or explicit storage acquired by the exit program by means of an EXEC CICS command is, by default, in user-key storage. However, on an EXEC CICS GETMAIN command, the exit program can override the TASKDATAKEY option by specifying either CICSATAKEY or USERATAKEY.

- When an exit program obtains storage by means of an XPI GETMAIN call, the storage key depends on the value specified on the STORAGE_CLASS option, which is mandatory, and which overrides the value of TASKDATAKEY.

Errors in user exit programs

Because global user exit programs are an extension to CICS code, they are subject to the environment that CICS is running in when they are called. If an error is detected at an exit point, CICS issues messages indicating which exit program was in error, the place in the program at which the error occurred, and the name of the associated exit point. The detection of an error is not guaranteed, because it depends on the CICS environment at the time of error, and on the nature of the error. For example, CICS might not recognize a looping user exit program, since the detection mechanism may have been turned off. Also, an abend in one of the exits XPCABND, XPCTA, or XSRAB may cause CICS to terminate abnormally, because an abend during abend processing causes CICS to terminate.

Exit programs invoked at some exit points (for example, XTSEREQ, XTSEREQC, XICEREQ, XICEREQC, XTDEREQ, or XTDEREQC) can enter a loop by issuing a recursive command (such as a TS command at exit point XTSEREQ). The exits most likely to be affected provide a recursion count parameter, UEPRECUR, that you can use to prevent such loops.

Warning: When coding user exit programs, you should bear in mind that the code is executed as an extension of CICS code, rather than as a transaction, and any errors could have disastrous results.

Defining, enabling, and disabling an exit program

When you have written an exit program, you must define it to CICS using the CEDA DEFINE PROGRAM command. (Note that you must specify RELOAD(NO).)

Having defined the exit program, you must also enable it. You do this using the EXEC CICS ENABLE command³. When you have finished using the exit program, you should disable it, using the EXEC CICS DISABLE command.

Note: If a global user exit program is enabled before it has been installed, CICS scans the LPA for the program and may issue message DFHLD0107I, meaning that it was unable to find the program in the LPA and is using the DFHRPL version.

For programming information about the EXEC CICS ENABLE and DISABLE commands, see the *CICS/ESA System Programming Reference* manual. For examples of how to enable and disable global user exit programs, see the sample programs listed on page 14.

Invoking more than one exit program at a single exit

There may be times when you want to invoke more than one exit program from a single global user exit point. For example, you might have two or more application packages that supply programs for the same CICS exit. Although such programs may work independently, you should note the following points:

- An exit program is only called at an exit if it has been made available for execution with the START option of the EXEC CICS ENABLE command. The order of invocation, when more than one exit program has been started at an exit point, is the order in which the programs were activated (that is, the order in which the EXEC CICS ENABLE commands associated them with the exit point). When programs work on the same data area, you should consider the order in which they are invoked. For example, in a terminal control output exit, an exit program might manipulate the same message in different ways, depending on the way an earlier exit program acted.
- Return code management is more complicated than it is for single programs. Each exit program sets a return code in register 15 as usual. The second and subsequent programs invoked from a single exit point can access the return code value set by the preceding program (the “current return code”) using the parameter UEPCRCA of DFHUEPAR.

The following rules apply to return codes if a second user exit program sets a different return code value from that selected by the previous program:

- If the new program supplies the same return code value as the current return code (addressed by UEPCRCA), then CICS acts on that value.
- If the new program supplies a different return code value from the current value addressed by UEPCRCA, CICS ignores both values and resets the

³ Exit programs for exits in the transaction backout programs can also be enabled using the TBEXITS system initialization parameter.

global user exit programs

“current return code” to the default value, usually UERCNORM, before calling any further exit programs for that exit point.

- If the new program sets an eligible value in register 15 **and** changes the “current value” field to match (as addressed by UEPCRCA), the new value is adopted and passed on to the next program (if any), or back to the calling CICS module or domain.

Invoking a single exit program at more than one exit

To invoke a single exit program from more than one exit point, you must issue an ENABLE command for each of the exit points. For programming information about how to issue an ENABLE command, see the *CICS/ESA System Programming Reference* manual. Be careful to specify GALENGTH or GAENTRYNAME on only the first ENABLE command, otherwise ‘INVEXITREQ’ may be returned.

Take into account the restrictions that apply to the use of CICS services, because these are dictated by the exit point itself rather than by the exit program. A command that can be issued from one exit point may cause problems when issued from a different exit point.

The global work area is associated with the exit **program**, rather than with the exit **point**: this means that the same global work area is used at each of the exit points at which the exit program is invoked.

Sample global user exit programs

CICS supplies a set of sample programs that show you how to:

- Enable a global user exit program and allocate a global work area (GWA).
- Obtain the address of an exit program’s GWA.
- Access CICS system information, and make that information available to user exit programs.
- Share a GWA between global user exit programs, thereby making the information it contains available to more than one program, and overcoming limitations on the size of GWAs.
- Access information held in a global user exit program’s GWA.
- Write a program to be invoked from a specific exit, to do a specific task. For example, the DFH\$XP4 sample program shows you how to use the XMEOUT exit to reroute a console message to a transient data queue.

The supplied programs and copy books are:

DFH\$DTAD	A sample global user exit program, designed to be invoked from the XDTAD exit.
DFH\$DTLC	A sample global user exit program, designed to be invoked from the XDTLC exit.
DFH\$DTRD	A sample global user exit program, designed to be invoked from the XDTRD exit.
DFH\$PCPI	This is designed to be invoked as part of program list table post initialization (PLTPI) processing, and is described in detail in “The DFH\$PCPI program” on page 15.
DFH\$PCPL	A dummy program, invoked by DFH\$PCPI, that causes the XPCFTCH user exit to be driven.

- DFH\$PCEX** A sample global user exit program, designed to be invoked from the XPCFTCH exit.
- DFH\$PCGA** Copy book of the DFH\$PCEX global work area.
- DFH\$PCTA** A sample global user exit program, designed to be invoked from the XPCTA exit. It is described on page 117.
- DFH\$SXPN** A set of sample global user exit programs designed to be invoked from the XMEOUT exit (where 'n' is 1 through 6).

DFH\$XDRQ

APAR PN64292

Documentation for PN64292 added on 23 January 1995.

- A sample global user exit program, designed to be invoked from the XDUREQ exit.
- DFH\$XZIQ** A sample global user exit program, designed to be invoked from the XZIQUE exit.
- DFH\$ZCAT** A sample global user exit program, designed to be invoked from the XZCATT exit.
- DFH\$ZCGA** Copy book of the DFH\$ZCAT global work area.
- DFH\$XIS** A sample global user exit program, designed to be invoked from the XISCONA exit.
- DFH\$XTENF** A sample global user exit program, designed to be invoked from the XALTENF or XICTENF exit. It is listed on page 163.

DFH\$DTAD, DFH\$DTLC, and DFH\$DTRD are listed in the *CICS/ESA Shared Data Tables Guide*. The source of all the sample programs and copy books is supplied, in assembler language, in the CICS410.SDFHSAMP library. You should use the supplied programs as models upon which to base your own versions.

The DFH\$PCPI program

DFH\$PCPI consists of three main sections:

1. Section 1 obtains and processes any parameters passed to DFH\$PCPI on the INITPARMS system initialization parameter.
2. Section 2 shows how to use standard CICS facilities to obtain system information, and make that information available to a global user exit program. It performs the following processing:
 - Uses the EXEC CICS ENABLE command to enable the XPCFTCH sample user exit program, DFH\$PCEX, and allocate it a global work area.
 - Uses the EXEC CICS EXTRACT EXIT command to obtain the address of DFH\$PCEX's global work area.
 - Obtains CICS system information, and places it in DFH\$PCEX's global work area. The information obtained includes:
 - Job name
 - Applid
 - Sysid
 - CICS release
 - Date in various formats, including DATFORM
 - The address of the common work area (CWA)
 - CICS startup type (COLD, WARM).

Most of the above information is obtained using EXEC CICS API commands such as:

- INQUIRE SYSTEM
 - ASSIGN
 - ADDRESS
 - ASKTIME
 - FORMATTIME.
- Uses the START option of the EXEC CICS ENABLE command to make DFH\$PCEX available for execution. This causes DFH\$PCEX to be driven for all LINKs and XCTLs.
 - Links to the dummy program, DFH\$PCPL, in order to drive DFH\$PCEX.
 - Uses the STOP option of the EXEC CICS DISABLE command to make DFH\$PCEX unavailable for execution. Note that this leaves DFH\$PCEX's global work area still allocated and accessible through the EXEC CICS EXTRACT EXIT command.
3. Section 3 of DFH\$PCPI shows how to share the system information in an exit program's global work area with other exit programs. (In doing so it demonstrates how application programs can access the same information by means of the EXEC CICS EXTRACT EXIT command.) It shows too how to use CICS shared storage to overcome the limitation of 32KB on the size of a GWA. The program obtains an area of 64KB below the 16MB line and an area of 128KB above it (using GETMAIN). The use of shared storage enables the second user exit program (DFH\$ZCAT) to use a global work area of only 12 bytes below 16MB.

The section performs the following processing:

- Uses EXEC CICS ENABLE to enable the DFH\$ZCAT user exit program, and allocate it a global work area
- Uses EXEC CICS EXTRACT EXIT to obtain the address of DFH\$ZCAT's global work area
- Stores the address of DFH\$PCEX's global work area in DFH\$ZCAT's global work area
- Uses GETMAIN to obtain the shared storage above and below the 16MB line, and stores the addresses in DFH\$ZCAT's global work area.

Sample program definitions

The following are examples of the RDO definitions required to define the sample programs to the CSD:

```
DEFINE PROGRAM(DFH$PCEX) GROUP(EXITGRP)
  LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
  EXECKEY(CICS)
DEFINE PROGRAM(DFH$PCPI) GROUP(EXITGRP)
  LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
  EXECKEY(CICS)
DEFINE PROGRAM(DFH$PCPL) GROUP(EXITGRP)
  LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
```



```

EXECKEY(CICS)
DEFINE PROGRAM(DFH$ZCAT) GROUP(EXITGRP)
  LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO) USAGE(NORMAL)
  USELPACOPY(NO) STATUS(ENABLED) CEDF(YES) DATALOCATION(ANY)
EXECKEY(CICS)

```

DFH\$PCPI is designed to be run as a PLT program. If you write a similar program, you should define it in the **second** part of the PLTPI list (after the PROGRAM=DFHDELIM entry) so that it executes in the **third** stage of CICS initialization. Information about how to do this is in the *CICS/ESA Resource Definition Guide*.

Example program

As well as the sample programs supplied in source code, there is an example listing, DFH\$XTSE, that shows you how to: The example shows you how to:

- Use EXEC CICS commands in a global user exit program
- Use EXEC CICS commands and XPI calls in the same exit program
- Modify the command parameter list in EXEC interface exits such as XTSERREQ and XICERREQ.

DFH\$XTSE is listed on page 717.

The global user exit points

The following table lists the exits in alphabetical order, giving a brief description and a page reference at which more information about each exit can be found. After the table, the following detailed information is provided for each of the global user exit points:

- The exit identifier
- The location of the exit
- The DFHUEPAR parameters, if any, that are unique to the exit
- The return codes that are valid for this exit point
- XPI calls that can be invoked.

In the detailed part of this section, the exit points are grouped according to their functional relationships. This usually means according to the CICS module or domain in which they occur. However, where exit points in different modules share the same identifier and set of parameters (as with XRCINIT in the DL/I backout program and the file control backout program, for example), or where exit points in different modules serve a similar function (XALTENF in the terminal allocation program and XICTENF in the interval control program, for example), the exits are grouped under a generic name (for example, “Exits in the transaction backout programs”). The groups of exits are presented in alphabetical order of module or generic name.

Accessing fields in CICS control blocks

When writing a program to be invoked from any of the global user exit points, note the warning contained in “Restrictions on the use of fields as programming interfaces” on page 11 about the use of control block fields as programming interfaces.

global user exit points

Table 1 (Page 1 of 5). Alphabetical list of global user exit points

Exit name	Module or domain	Where or when invoked	Page
XAKUSER	Activity keypoint program	Immediately before the 'end of keypoint' record is written.	23
XALCAID	Terminal allocation program	Whenever an AID with data is canceled.	152
XALTENF		When an ATI request from transient data or interval control requires a terminal that is unknown in this system.	159
XBMIN	Basic Mapping Support	When an input mapping operation completes successfully.	25
XBMOUT		When a page of output has been built successfully.	25
XDBDERR	Dynamic transaction backout program	When the DL/I backout routine detects an error.	50
XDBFERR		If the dynamic transaction backout program or the file control program detects an error.	48
XDBIN		Each time CICS reads a non-DL/I dynamic log record.	47
XDBINIT		On entry to the dynamic transaction backout program.	47
XDLIPOST	DL/I interface program	On exit from the DL/I interface program.	36
XDLIPRE		On entry to the DL/I interface program.	34
XDSAWT	Dispatcher domain	After an operating system wait.	33
XDSBWT		Before an operating system wait.	32
XDTAD	Shared data tables management program	When a write request is issued to a data table.	128
XDTLC		At the completion of loading of a data table.	130
XDTRD		During the loading of a data table, whenever a record is retrieved from the source data set.	125
XDUCLSE	Dump domain	After the domain closes a transaction dump data set.	43
XDUOUT		Before the domain writes a record to the transaction dump data set.	45
XDUREQ		Before the domain takes a system or transaction dump.	39
XDUREQC		After a system or transaction dump has been taken (or failed or been suppressed).	41
XEIIN	EXEC interface program	Before the execution of an EXEC CICS command.	52
XEIOUT		After the execution of an EXEC CICS command.	53

Table 1 (Page 2 of 5). Alphabetical list of global user exit points

Exit name	Module or domain	Where or when invoked	Page
XFCNREC	File control open/close program	When a mismatch is detected between the backout recovery setting for a file and its associated dataset during file open processing.	55
XFCREQ	File control EXEC interface program	Before CICS processes a file control API request.	64
XFCREQC		After a file control API request has completed.	65
XFCSREQ	File control file state program	Before a file OPEN, CLOSE, ENABLE, or DISABLE command is attempted.	67
XFCSREQC		After a file OPEN, CLOSE, ENABLE, or DISABLE command has been completed.	67
XGMTEXT	“Good morning” message program	Before the “good morning” message is sent.	76
XICERREQ	Interval control EXEC interface program	Before CICS processes an interval control API request.	84
XICERREQC		After an interval control API request has completed.	86
XICEXP	Interval control program	After expiry of an interval control time interval.	83
XICREQ		At the start of the interval control program, before request analysis.	82
XICTENF		When an EXEC CICS START command requires a terminal that is unknown in this system.	161
XISCONA	Intersystem communication program	After an attempt to allocate a session for function shipping fails because no sessions are immediately available.	77
XISLCLQ		After an attempt to allocate a session for a function shipped START NOCHECK request fails because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available and your XISCONA exit program has specified that the request is not to be queued in the issuing region.	81
XJCWB	Journal control program	Before each physical WRITE to each non-SMF journal.	97
XJCWR		After building a journal record in the buffer, but before writing it.	96
XKCREQ	Task control program	Before an ENQUEUE or DEQUEUE request.	139

global user exit points

#

|
|

<i>Table 1 (Page 3 of 5). Alphabetical list of global user exit points</i>			
Exit name	Module or domain	Where or when invoked	Page
XLDLOAD	Loader domain	After an instance of a program is brought into storage, and before the program is made available for use.	98
XLDELETE		After an instance of a program is released by CICS and just before the program is freed from storage.	99
XMEOUT	Message domain	Before a message is sent from the message domain to its destination.	101
XMNOUT	Monitoring domain	Before a record is either written to SMF or buffered before a write to SMF.	104
XPCABND	Program control program	Before a dump call is made.	118
XPCFTCH		Before an application program is given control.	113
XPCHAIR		Before a HANDLE ABEND routine is given control.	115
XPCREQ		Before a LINK request is processed.	106
XPCREQC		After a LINK request has been completed.	107
XPCTA		After an abend occurs, and before the environment is modified.	116
XRCINIT		DL/I backout program	At the beginning and end of the DL/I backout program.
	File control backout program	At the beginning and end of the file control backout program.	167
	Message and ISC state recovery program	At the beginning and end of the message and ISC state recovery program.	167
	User backout program	At the beginning and end of the user backout program.	168
XRCINPT	File control backout program	Whenever a non-DL/I record is read from the restart data set.	168
	Message and ISC state recovery program	Whenever a non-DL/I record is read from the restart data set.	169
	User backout program	Whenever a non-DL/I record is read from the restart data set.	169
XRCFCER	File control backout program	When the file control program returns an error during backout processing.	170
XRCOPER		If an error occurs while opening a file control data set.	170

Table 1 (Page 4 of 5). Alphabetical list of global user exit points

Exit name	Module or domain	Where or when invoked	Page
XRMIIIN	Resource manager interface program	Before execution of an EXEC DLI, EXEC SQL, or RMI command.	119
XRMIOUT		After execution of an EXEC DLI, EXEC SQL, or RMI command.	119
XRSINDI	Resource management modules	Immediately after a successful install or discard of a resource.	121
XSNOFF	Security manager domain	After a terminal user signs off.	132
XSNON		After a terminal user signs on.	131
XSRAB	System recovery program	When the system recovery program finds a match for an MVS abend code in the SRT.	135
XSTERM	System termination program	During a normal system shutdown, immediately before TD buffers are cleared.	138
XSTOUT	Statistics domain	Before a statistics record is written to SMF.	133
XSZARQ	Front End Programming Interface	After a FEPI request has completed.	75
XSZBRQ		Before a FEPI request is actioned.	75
XTCATT	Terminal control program	Before task attach.	155
XTCIN		After an input event.	154
XTCOUT		Before an output event.	154
XTCTIN		After a TCAM input event.	155
XTCTOUT		Before a TCAM output event.	156
XTDEREQ		Transient data EXEC interface program	Before CICS processes a transient data API request.
XTDEREQC	After a transient data API request has completed.		179
XTDIN	Transient data program	After receiving data from QSAM (extrapartition) or VSAM (intrapartition).	175
XTDOUT		Before passing data to a QSAM (extrapartition) or VSAM (intrapartition) user-defined transient data queue.	176
XTDREQ		Before request analysis.	175
XTSREQ	Temporary storage EXEC interface program	Before CICS processes a temporary storage API request.	143
XTSREQC		After a temporary storage API request has completed.	144
XTSIN	Temporary storage control program	After an input event.	140
XTSOUT		Before an output event.	141
XTSREQ		Before request analysis.	140

global user exit points

Table 1 (Page 5 of 5). Alphabetical list of global user exit points

Exit name	Module or domain	Where or when invoked	Page
XXDFA	DBCTL interface control program	In the active CICS when CICS-DBCTL connection fails.	29
XXDFB	DBCTL tracking program	In the alternate CICS when DBCTL fails.	30
XXDTO		In the alternate CICS when active DBCTL fails.	30
XXMATT	Transaction manager domain	When a user transaction is attached.	173
XXRSTAT	XRF request processing program	After a VTAM failure or a predatory takeover.	198
XZCATT	VTAM terminal management program	Before task attach.	185
XZCIN	VTAM working set module	After an input event.	186
XZCOUT		Before an output event.	186
XZCOUT1		Before a message is broken into RUs.	187
XZIQUE		<ol style="list-style-type: none"> 1. When an allocate request for a session is about to be queued. 2. When an allocate request succeeds following previous suppression of queuing. 	188

Activity keypoint program exit XAKUSER

The XAKUSER exit is invoked during the activity keypointing process. This exit should be used to record, on the system log, user data that must be restored following an emergency restart. For further information about the use of the exit, see the *CICS/ESA Recovery and Restart Guide*.

For best performance, journal control requests should not specify WAIT or STARTIO. CICS will force the records by writing a synchronous end of keypoint record upon return from the exit program.

Your exit program should be translated with the NOEDF option. Any program it links to should also be translated with this option. It is not possible to link to programs written in PL/I.

To ensure that it is called during every keypoint, including the one taken at CICS startup, your exit program should be enabled by means of a first phase PLTPI program (that is, a program that is invoked during the *second* stage of CICS initialization—see “Initialization programs” on page 335). However, if it enabled at this stage, your program should not attempt to link to any program coded in VS COBOL II or C, as it may be invoked before these are initialized.

Table 2. Exit XAKUSER

When invoked	During the activity keypointing process.
Exit-specific parameters	None.
Return codes	UERCNORM Continue processing.
XPI calls	XPI must not be used.
API calls	The following commands are supported: ADDRESS CWA ADDRESS EIB LINK (but only to local programs; distributed program links may not be used). RETURN WRITE JOURNALNUM.

Important

Only the listed EXEC CICS commands are allowed in the XAKUSER exit. The exit should link only to other programs with the same restrictions.

Basic Mapping Support exits XBMIN and XBMOU

#

— APAR PQ12071 —

#

Documentation for APAR PQ 12071 added 13th February 1998.

#

The XBMIN exit allows you to intercept a RECEIVE MAP request after BMS has successfully processed the request. The XBMOU exit allows you to intercept a SEND MAP request after BMS has successfully processed the request; or, if cumulative mapping is in progress, on completion of each page of output.

#

#

#

#

The XBMIN exit is called, if enabled, when all the following are true:

#

- A RECEIVE MAP command has been successfully processed.

#

- The map referenced in the command contains at least one field specified as VALIDN=USEREXIT.

#

#

- At least one USEREXIT field has been returned in the inbound datastream and has been mapped into the application data structure.

#

#

Using XBMIN, you can:

#

- Analyze each field defined as VALIDN=USEREXIT mapped to the application on this request

#

#

- Use the mapset name, map name, and field length defined in the map, and the actual length of field data returned in the inbound datastream

#

#

- Modify the data in each field.

#

The XBMOU exit is called, if enabled, when all the following are true:

#

- A SEND MAP command has been successfully processed.

#

- The map referenced in the command contains at least one field specified as VALIDN=USEREXIT.

#

#

- At least one USEREXIT field has been generated in the outbound datastream.

#

Using XBMOU, you can:

#

- Analyze each field defined as VALIDN=USEREXIT which has been generated in the outbound datastream

#

#

- Use the mapset name, map name, and field length defined in the map, and the actual length of field data placed in the outbound datastream

#

#

- Modify the data in each field

#

- Modify the attributes sent with each field.

#

Both exits are passed four exit-specific parameters:

#

1. The address of the TCTTE associated with the mapping request

#

2. The address of the system EIB associated with the task issuing the mapping request

#

3. The address of a halfword binary count of the number of elements in the *field element table*

#

#

4. The address of the field element table.

Basic Mapping Support exits

BMXMAP
is a 7-byte area which contains the name of the map associated with this field.

BMXFDFB
is a one-byte field copied from the field specification in the map load module. It
contains indicators as follows:

X'80' CASE=MIXED
X'40' Group field entry
X'20' Group field descriptor
X'10' ATTRB=DET
X'08' JUSTIFY=ZERO
X'04' JUSTIFY=RIGHT
X'02' INITIAL,XINIT, or GINIT specified
X'01' Named field (DSECT entry exists)

BMXMAPLN
is a halfword binary value which contains the field length defined in the
LENGTH option of the DFHMDF macro.

BMXACTLN
is a halfword binary value which contains the actual length of the data received
or transmitted in this field.

BMXDATA
is the address of the field data.

In the XBMIN exit, BMXDATA points into a work area which BMS has obtained
for input mapping purposes. When the exit returns control, this work area is
copied to the application data structure associated with this map.

In the XBMOU exit, BMXDATA points into a terminal input/output area (TIOA)
in which BMS has generated an output datastream. When the exit returns
control, the TIOA is disposed of in accordance with the disposition of the
TERMINAL (the default), SET, or PAGING option specified on the SEND MAP
request.

BMXATTR
is only relevant in the XBMOU exit. It is the address of the attributes (if any)
which BMS has placed in the output datastream preceding this field.

BMXMAPOF
is the offset of the field in the map. For example, if a map is defined as
MYMAP DFHMDF SIZE=(12,40)

and a field in this map is defined as
FLDA DFHMDF POS=(5,1)

the offset of this field (relative to zero) is 160 in decimal notation. In this
example, BMXMAPOF would contain the value X'00A0'.

BMXBUF
is the offset of the field in the device buffer. Usually—that is, when the map
dimensions are the same as the current screensize in use by the device—this
value will be the same as that of BMXMAPOF. However, using the example
given in the BMXMAPOF description above, if MYMAP is sent to a device
currently using a 24 by 80 screensize, the offset of the field in the device buffer

(again relative to zero) is 320 in decimal notation. In this example, BMXBUF
would contain the value X'0140'.

Programming the XBMIN exit

This section contains some considerations specific to the XBMIN exit.

The actual data length (in BMXACTLN) may be less than the length defined in the
map (in BMXMAPLN). This could happen, for example, if a terminal operator does
not completely fill a data entry field. In this case, BMS will have right- or
left-justified the data in the field and padded the field with blank or zero characters.
This justification and padding occurs before the exit is invoked. Your exit program
can, by checking the bit settings in the BMXFDFB field, determine how BMS
performed justification and padding for the field.

The actual data length (in BMXACTLN) may be greater than the length defined in
the map (in BMXMAPLN). This could happen, for example, if a map contains an
unprotected field which is not immediately followed by another field. This allows
the terminal operator to enter data past the end of the field. When this occurs, the
data field is truncated by BMS according to the length defined for the field in the
map. However, BMXACTLN contains the length of data found in the inbound
datastream.

When modifying data in the XBMIN exit, the safest method is to use the length
provided in BMXMAPLN, but to ensure that any pad characters added by BMS are
preserved.

BMXATTR must be ignored in the XBMIN exit; it always contains binary zeroes.

Programming the XBMOU exit

This section contains some considerations specific to the XBMOU exit.

The actual data length (in BMXACTLN) may be less than the length defined in the
map (in BMXMAPLN). This occurs due to the compression of trailing nulls
performed by BMS for each output field.

The actual length of data cannot be changed in the exit program. The exit is
invoked after the output datastream has been generated; consequently, an attempt
to alter the data length could result in an invalid datastream. Therefore, if an
XBMOU exit program modifies data, it must do so with reference to the length
value in BMXACTLN.

BMXDATA may contain a null value. This can be caused by a SEND MAP request
with the MAPONLY option when the map has fields without default data; this
causes BMS to send an attribute sequence for the field but no data.

BMXATTR may contain a null value. This can be caused by a SEND MAP request
with the DATAONLY option, when the application is updating the data in a field and
not the attributes.

Cumulative mapping operations: When an application is performing cumulative
mapping—that is, issuing a sequence of SEND MAP commands with the ACCUM
option—BMS builds composite display in which a single page of output might be
constructed from multiple SEND MAP requests. When cumulative mapping occurs,

Basic Mapping Support exits

the XBMOU exit is called when a page has been built, not as each SEND MAP
request is processed.

Message routing: When an application builds a routing message—for example, it
issues a ROUTE command followed by one or more SEND MAP commands with
the SET or PAGING option specified—the XBMOU exit is invoked in the same
way as for a non-routed mapping request.

However, the UEPBMTCT parameter is passed as a null value for a routed
message. This is because a routed message may be destined for multiple devices,
and BMS has optimized the features supported by the devices targeted by the
routed message. When processing a routed message in the XBMOU exit,
referencing the TCTTE for any of these devices would probably not be relevant.

DBCTL interface control program exit XXDFA

<i>Table 5. Exit XXDFA</i>		
When invoked		By an active CICS when its connection to DBCTL fails. Your exit program is invoked after the active CICS has informed the alternate CICS of the failure. Information about this exit is provided in the <i>CICS/ESA CICS-IMS Database Control Guide</i> .
Exit-specific parameters	UEPDBXR	Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.
Return codes	UERCNOAC	Take no action.
	UERC SWCH	Switch to the alternate DBCTL.
	UERCABNO	Abend CICS without a dump.
	UERCABDU	Abend CICS with a dump.
XPI calls		TRANSACTION_DUMP must not be used.

DBCTL tracking program exits XXDFB and XXDTO

XXDFB

<i>Table 6. Exit XXDFB</i>	
When invoked	By the alternate CICS when it receives a message from the active CICS indicating that connection to DBCTL has failed. The alternate and active CICS systems are running in different MVS images, perhaps in different central processing complexes (CPCs). More information about these exits is provided in the <i>CICS/ESA CICS-IMS Database Control Guide</i> .
Exit-specific parameters	UEPDBXR Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.
Return codes	<p>UERCNOAC Take no action.</p> <p>UERC SWCH Switch to the alternate DBCTL.</p> <p>UERCABNO Abend CICS without a dump.</p> <p>UERCABDU Abend CICS with a dump.</p> <p>The return code 'UERCNORM' is not available for use at this exit point.</p>
XPI calls	The following must not be used: INQUIRE_MONITORING_DATA MONITOR TRANSACTION_DUMP WRITE_JOURNAL_DATA.

XXDTO

<i>Table 7 (Page 1 of 2). Exit XXDTO</i>	
When invoked	<p>By an alternate CICS when it performs takeover under the following conditions:</p> <ul style="list-style-type: none"> • The active and alternate CICS systems are in different MVS images, perhaps in different processors. • The active CICS was connected to, or trying to connect to, a DBCTL subsystem. (This does not include disconnecting from one DBCTL and reconnecting to another.) • The takeover was not initiated by the XXDFB exit, or the takeover was initiated by XXDFB but the active system reestablished a DBCTL connection before takeover occurred and XXDTO was driven for a new DBCTL takeover decision. <p>More information about this exit is provided in the <i>CICS/ESA CICS-IMS Database Control Guide</i>.</p>
Exit-specific parameters	UEPDBXR Address of CICS XRF information for use with DBCTL. The CICS XRF information can be mapped using the DSECT DFHDXUEP.

<i>Table 7 (Page 2 of 2). Exit XXDTO</i>	
Return codes	<p>UERCNOAC Take no action.</p> <p>UERC SWCH Switch to the alternate DBCTL.</p> <p>UERCABNO Abend CICS without a dump.</p> <p>UERCABDU Abend CICS with a dump.</p> <p>The return code UERCNORM is not available for use at this exit point.</p>
XPI calls	<p>The following must not be used:</p> <p>INQUIRE_MONITORING_DATA</p> <p>MONITOR</p> <p>TRANSACTION_DUMP</p> <p>WRITE_JOURNAL_DATA.</p>

Dispatcher domain exits XDSBWT and XDSAWT

The XDSBWT and XDSAWT exit points are located before and after the operating system wait. CICS services **cannot** be used in any exit program invoked from these exit points.

The XDSBWT and XDSAWT exits can be used to control the swapping state of the CICS address-space. It should be noted, however, that if the default state of the address-space is non-swappable then these exits cannot be used to override this state.

CICS uses a counter which is incremented for every SYSEVENT DONTSWAP request and decremented for every SYSEVENT OKSWAP request down to a minimum of 0. A SYSEVENT DONTSWAP request is issued when this counter goes up from 0 to 1. A SYSEVENT OKSWAP request is issued when this counter goes down from 1 to 0. In all other circumstances, the SYSEVENT is not issued.

XDSBWT

<i>Table 8. Exit XDSBWT</i>	
When invoked	Before an operating system wait issued by the quasireentrant CICS TCB.
Exit-specific parameters	None.
Return codes	UERCNORM Continue processing. UERCSWAP Issue SYSEVENT to allow address-space swapping.
XPI calls	Must not be used.

XDSAWT

Table 9. Exit XDSAWT

When invoked	After an operating system wait issued by the quaireentrant CICS TCB.	
Exit-specific parameters	UEPSYSRC	<p>Address of the 4-byte return code from the SYSEVENT request made before the operating system wait. This return code will be in one of two different forms:</p> <ol style="list-style-type: none"> 1. The SYSEVENT OKSWAP return code, or 2. If the SYSEVENT request was rejected by CICS, a special CICS return code which will take one of the following fullword decimal values: <ul style="list-style-type: none"> 17 The SYSEVENT OKSWAP was not issued. The outstanding count of SYSEVENT OKSWAP requests exceeds the count of SYSEVENT DONTSWAP requests. Before a SYSEVENT OKSWAP can be issued, a SYSEVENT DONTSWAP must be requested. 19 The SYSEVENT OKSWAP was not issued. The outstanding count of SYSEVENT DONTSWAP requests still exceeds the count of SYSEVENT OKSWAPs. Further SYSEVENT OKSWAPs must be requested before a SYSEVENT OKSWAP is issued by CICS.
Return codes	UERCNORM	Continue processing.
	UERCNOSW	Issue SYSEVENT to suppress address-space swapping.
XPI calls	Must not be used.	

DL/I interface program exits XDLIPRE and XDLIPOST

The XDLIPRE and XDLIPOST exit points are invoked following the issue of an EXEC DLI command or DL/I call. Exit XDLIPRE is invoked before the request is processed and XDLIPOST is invoked after the request is processed.

When the request is function shipped, the exits are invoked from both the application-owning region and the database-owning region. However, there are restrictions when they are invoked in a database-owning region, as described below.

The *CICS/ESA CICS-IMS Database Control Guide* contains a sample program for the XDLIPRE exit.

Notes:

1. The descriptions of the exits that follow show the general format of the application's parameter list. For detailed information about the format of the CALL-level DL/I parameter list, refer to the *IMS/ESA Application Programming: DL/I Calls* manual.
2. For all EXEC DLI calls, the application's parameter list is in assembler-language format (that is, the value of the program language byte pointed to by UEPLANG is always UEPASM, and the parameter list pointed to by UEPALIST is always in assembler-language format). This is because all EXEC DLI calls are converted into assembler-language CALL-level requests.

XDLIPRE

<i>Table 10 (Page 1 of 3). Exit XDLIPRE</i>	
When invoked	On entry to the DL/I interface program.
Exit-specific parameters	<p>UEPCTYPE Address of type-of-request byte. Values are:</p> <p>UEPCEXEC The original request was an EXEC DLI request.</p> <p>UEPCCALL The original request was a CALL-level request.</p> <p>UEPCSHIP The request has been function shipped from another region. When this value is set, restrictions apply to the setting and use of the rest of the exit parameters, as described below.</p>

Table 10 (Page 2 of 3). Exit XDLIPRE

UEPAPLIST Address of application's parameter list. The general format for COBOL and assembler language is:

```
plist address --> parm1 address --> parm1
                parm2 address --> parm2
                parm3 address --> parm3
                .....
                up to a maximum of 18 parameters
                excluding the optional parmcount.
```

The general format for PL/I is:

```
plist address --> parm1 address --> parm1 (parmcount)
                parm2 address --> locator descriptor --> parm2
                parm3 address --> locator descriptor --> parm3
                .....
                up to a maximum of 18 parameters
```

When UEPCTYPE is not UEPCSHIP, your exit program can change any of the parameters in the application parameter list. For UEPCSHIP requests, your exit program **cannot** change any of the parameters. Furthermore, for UEPCSHIP requests, UEPAPLIST points to a copy of the parameter list in the above format, but which contains only the first two parameters, parm1 and parm2.

Note: For PL/I applications, parm1 may or may not contain a parameter-count. Your exit program should check this field before using it.

UEPLANG Address of program language byte. Values are:

```
UEPPLI PL/I
UEPCBL COBOL
UEPASM Assembler language.
```

For UEPCSHIP requests, the language is always assembler.

UEPIOAX Address of I/O area existence flag byte:

UEPIOA1 I/O area exists.

For UEPCSHIP requests, the I/O area existence flag is always off.

UEPIOA Address of I/O area. This is the application's IOAREA, or DFHEDP's IOAREA in the case of EXEC DLI. The contents of the IOAREA can be overwritten in the exit: the new contents are used when the DL/I request is processed. However, it should be noted that IOAREAs can be in a program's static storage and, in this case, should not be overwritten.

For UEPCSHIP requests, UEPIOA is always zero.

UEPPSBNX Address of PSB existence flag byte:

UEPPSB1 A PSB exists.

UEPPSBNM Address of an area containing the 8-character PSB name. The contents of the area can be overwritten by the exit, for all types of request including UEPCSHIP; the new contents are used when the DL/I request is processed.

DL/I interface program exits

<i>Table 10 (Page 3 of 3). Exit XDLIPRE</i>	
Return codes	UERCNORM Continue processing UERCBYBYP Bypass DL/I request and return UERCPUARG Task purged during XPI call.
XPI calls	All can be used.

XDLIPOST

<i>Table 11 (Page 1 of 3). Exit XDLIPOST</i>	
When invoked	On exit from the DL/I interface program.

Table 11 (Page 2 of 3). Exit XDLIPOST

Exit-specific parameters	UEPCTYPE	Address of type-of-request byte. Values are:
	UEPCEXEC	An EXEC DLI request
	UEPCCALL	A CALL-level request
	UEPCSHIP	The request has been function shipped from another region. When this value is set, restrictions apply to the setting and use of the rest of the exit parameters, as described below.
	UEPAPLIST	Address of application's parameter list. The general format for COBOL and assembler language is:
		<pre> plist address --> parm1 address --> parm1 parm2 address --> parm2 parm3 address --> parm3 up to a maximum of 18 parameters excluding the optional parmcount. </pre>
		The general format for PL/I is:
		<pre> plist address --> parm1 address --> parm1 (parmcount) parm2 address --> locator descriptor --> parm2 parm3 address --> locator descriptor --> parm3 up to a maximum of 18 parameters. </pre>
		When UEPCTYPE is not UEPCSHIP, your exit program can change any of the parameters in the application parameter list. For UEPCSHIP requests, your exit program cannot change any of the parameters. Furthermore, for UEPCSHIP requests, UEPAPLIST points to a copy of the parameter list in the above format, but which contains only the first two parameters parm1 and parm2. See also the notes on page 34.
		Note: For PL/I applications, parm1 may or may not contain a parameter-count. Your exit program should check this field before using it.
	UEPLANG	Address of program language byte. Its values are:
	UEPPLI	PL/I
	UEPCBL	COBOL
	UEPASM	Assembler language.
		For UEPCSHIP requests, the language is always assembler.
	UEPIOAX	Address of I/O area existence flag byte:
	UEPIOA1	I/O area exists.
		For UEPCSHIP requests, the I/O area existence flag is always off.

<i>Table 11 (Page 3 of 3). Exit XDLIPOST</i>	
UEPIOA	Address of I/O area. This is the application's IOAREA, or DFHEDP's IOAREA in the case of EXEC DLI. The contents of the IOAREA can be overwritten in the exit and are returned to the application program in the new form. However, it should be noted that the application's IOAREA could be in the program's static storage and, in this case, should not be overwritten. For UEPCSHIP requests, UEPIOA is always zero.
UEPUIBX	Address of UIB existence flag byte: UEPUIB1 a UIB exists.
UEPUIB	Address of the UIB, which is mapped by DFHUIB in module DFHDBCOP. The contents of the UIB can be overwritten in the exit for all types of request, including UEPCSHIP. The new contents are returned to the application or to the region that function shipped the request.
Return codes	UERCNORM Continue processing. UERCPURG Task purged during XPI call.
XPI calls	All can be used.

+
+
+
+
+
+
+
+
+
+
+

<i>Table 12 (Page 3 of 3). Exit XDUREQ</i>	
UEPXDDAE	Address of a 1-byte field which represents the current dump table DAEOPTION setting. It contains one of the following values: UEPXDYES The dump is eligible for DAE suppression. UEPXDNO The dump will not be suppressed by DAE. This field may be modified by the exit to update the dump table DAEOPTION setting.
UEPDMPID	Address of a 9-character field in the format xxxx/xxxx, containing the dump identifier. The dump ID is in the same format as that output by the corresponding dump message.
Return codes	UERCNORM Continue processing. UERCBYP Suppress dump. UERCPURG Task purged during XPI call.
XPI calls	WAIT_MVS can be used only when a UEPDUMPT indicates that a transaction dump is being taken. Do not use any other calls.

+
+
+

The sample program for the XDUREQ exit, DFH\$XDRQ

— **APAR PN64292** —
Documentation for PN64292 added on 23 January 1995.

+
+
+

CICS supplies a sample program for the XDUREQ exit, which shows you how to manipulate the dump table entry, and how to permit or suppress the dump.

— **APAR PN89471** —
Documentation for PN89471 added on 4 November 1996.

+
+
+
+
+
+
+

XDUREQC

<i>Table 13 (Page 1 of 3). Exit XDUREQC</i>	
When invoked	Immediately after a system or transaction dump has been taken (or has failed or been suppressed).
Exit-specific parameters	UEPTRANID Address of the 4-byte transaction ID. UEPUSER Address of the 8-byte user ID. UEPTERM Address of the 4-byte terminal ID. UEPPROG Address of the 8-byte application program name. UEPDUMPC Address of copy of the 8-byte dump code.

dump domain exits

<i>Table 14 (Page 2 of 2). Exit XDUCLE</i>		
Exit-specific parameters	UEPTRANID	Address of the 4-byte transaction ID.
	UEPUSER	Address of the 8-byte user ID.
	UEPTERM	Address of the 4-byte terminal ID.
	UEPPROG	Address of the 8-byte application program name.
	UEPDMPDD	Address of the 8-byte dump data set ddname.
	UEPDMPDSN	Address of the 44-byte dump data set dsname.
Return codes	UERCNORM	Continue processing.
	UERCSWCH	The autoswitch flag is set on.
XPI calls	WAIT_MVS can be used. Do not use any other calls.	

XDUOUT

<i>Table 15. Exit XDUOUT</i>	
When invoked	Before a record is written to the transaction dump data set.
Exit-specific parameters	<p>UEPTRANID Address of the 4-byte transaction ID.</p> <p>UEPUSER Address of the 8-byte user ID.</p> <p>UEPTERM Address of the 4-byte terminal ID.</p> <p>UEPPROG Address of the 8-byte application program name.</p> <p>UEPDMPFC Address of the 1-byte function code. The equated values are:</p> <p>UEPDMPWR Buffer is about to be written.</p> <p>UEPDMPRE Dump is about to restart after autoswitch.</p> <p>UEPDMPAB Abnormal termination of dump.</p> <p>UEPDMPDY Buffer is about to be written, and the CICS dump data set is a dummy file or is closed.</p> <p>UEPDMPBF Address of the dump buffer, whose length is addressed by the parameter UEPDMPLEN.</p> <p>UEPDMPLEN Address of the 2-byte dump-buffer length.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYP Suppress dump record output.</p>
XPI calls	WAIT_MVS can be used. Do not use any other calls.

Dynamic transaction backout program exits XDBINIT, XDBIN, XDBFERR, and XDBDERR

For guidance information about the exits in the dynamic transaction backout program, refer to the *CICS/ESA Recovery and Restart Guide*.

Coding the exit programs

You can modify recoverable resources in dynamic transaction backout exit programs, but note the following:

- CICS services can be invoked from exit programs at any of these exits using the XPI or EXEC CICS commands, but there are restrictions on the use of EXEC CICS:
 - Your exit program must not issue EXEC CICS commands if recovery is as the result of an EXEC CICS SYNCPOINT ROLLBACK request. (You can discover whether this is the case by examining the UEPDBTYP exit-specific parameter passed to the XDBINIT exit, which is invoked on entry to the dynamic transaction backout program.)
 - It is unwise to issue any file control requests when backing out file resources. If your exit program does issue file control requests, note that:
 - A READ UPDATE should be properly unlocked, either implicitly or explicitly, or backout may be locked out.
 - In the XDBFERR exit, do not disable or close files, or you could cause further error conditions.
 - In the XDBINIT exit, avoid changes to recoverable transient data and temporary storage because they will back out immediately.
 - In the XDBINIT exit, the current DL/I PSB should be left scheduled; it should not be terminated.
 - Exit programs that issue EXEC CICS commands must first address the EIB. See “Using CICS services” on page 5.
 - Exit programs which issue EXEC CICS requests cannot link to a program running AMODE 24.
 - Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See “Returning values to CICS” on page 10.
 - If they issue EXEC CICS commands, programs invoked from these exits must be translated with the NOEDF option. See “Using EDF with global user exits” on page 6.
- File control operations are performed by the dynamic transaction backout program and changes made to files (including those performed in user exits) are recorded in the system log by the file control program.
- In the XDBIN exit, you can set a return code to ignore a file-related record if, for example, backout for a particular file is to be suppressed for some reason.

Warning: Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a dynamic transaction backout request is issued from any of these exits.

Use of the recursion counter UEPRECUR is recommended.

XDBINIT

<i>Table 16. Exit XDBINIT</i>		
When invoked	On entry to the dynamic transaction backout program.	
Exit-specific parameters	UEPDBTYP	1-byte flag indicating why dynamic backout was invoked: UEPROLLB SYNCPOINT ROLLBACK request UEPABEND Abend processing.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing.
	UERCBYPD	Suppress all DL/I backout.
	UERCBYPA	Suppress all backout.
	UERCPURG	Task purged during XPI call.
	Return code values of 'UERCBYPD' and 'UERCBYPA' cause any databases updated by DL/I to be closed by DL/I backout failure processing. For more information about databases updated by DL/I, refer to the <i>CICS/ESA Operations and Utilities Guide</i> .	
XPI calls	All can be used.	

XDBIN

<i>Table 17. Exit XDBIN</i>		
When invoked	Each time a non-DL/I dynamic log record is read in.	
Exit-specific parameters	UEPDLOGR	Address of dynamic log record. The dynamic log record can be mapped using the DSECT DFHDBRDS. You can determine the type of dynamic log record read by testing field DBRMODFN (defined in DFHDBRDS) with values provided by copy book DFHFIDS.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing.
	UERCBYP	The dynamic transaction backout program should ignore this dynamic log record.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

XDBFERR

Table 18 (Page 1 of 2). Exit XDBFERR

When invoked	The dynamic transaction backout program invokes the file control program to perform backout. Exit XDBFERR is invoked if the file control program returns with an error, or if the dynamic transaction backout program itself detects an error during its file backout processing.
Exit-specific parameters	<p>UEPDLOGR The address of the dynamic log record. The dynamic log record can be mapped using the DSECT DFHDBRDS.</p> <p>This parameter is followed by a 4-byte reserved field in the parameter list.</p> <p>UEPFCTE The address of the file control table (FCT) entry, if any (see note below). The FCT can be mapped using the DSECT DFHFCTDS.</p> <p>UEPFCRSP The address of the file control response byte. This can have one of the following values:</p> <ul style="list-style-type: none"> UENORESP Normal response UEFILERR File not found UEINVREQ Invalid request UEILOGIC VSAM illogical error UENOTOPN Not open UEDISAB Disabled UEENDFIL End of file UEIOEROR I/O error UENOTFND Record not found UEDUPREC Duplicate record UENOSPAC No space available UELENGER Length error UEDUPKEY Duplicate key in alternate index (AIX) UEPURGED Transaction canceled UEDISAST Disastrous error detected. <p>A value of UEPFCRSP other than one of those listed above indicates that File Control detected an invalid request.</p> <p>UEPERR The address of a one-byte field containing the error type. The values of the error-byte and their meanings are described in "Values of the error byte pointed to by UEPERR" on page 49, and are defined using copy book DFHDBRDS.</p> <p>UEPFDATA Address of a variable-length field containing the data in the file control request.</p> <p>UEPFLEN Address of a fullword containing the length of the data in the file control request.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>

<i>Table 18 (Page 2 of 2). Exit XDBFERR</i>	
	Note: UEPFCTE, UEPFDATA, and UEPFLEN contain valid values depending on the type of error reported by UEPFCRSP. For example, if the byte addressed by UEPFCRSP contains 'UEFILERR' (file not found), then UEPFCTE and UEPFDATA are zero.
Return codes	<p>UERCNORM Continue processing and invoke CICS backout failure control.</p> <p>UERCBYP Ignore the error (do not invoke CICS backout failure control) and continue.</p> <p>UERCRTY Retry the request or, if the error type (UEPERR) is DBFEWA, reapply the record.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	All can be used.

Values of the error byte pointed to by UEPERR

Field UEPERR in the XDBFERR parameter list points to the error byte in the dynamic log record. The error byte can contain one of the following values:

DBFEGU

An error response has been returned from the file control program (DFHFCVS) while servicing a GET UPDATE request. DFHDBP has attempted to retrieve the existing copy of the record before backing it out. Parameter UEPFCRSP (which gives the address of the byte containing the response to the file control request) in combination with the type of dynamic log record pointed to by UEPDLOGR ("read for update" or "write add") can be used in the exit to determine the specific problem.

DBFELE

The area acquired in DFHFCVS is not large enough to receive the before-copy data picked up from the dynamic log to perform the backout. The parameter UEPFDATA points to the record. Parameter UEPFLEN points to a field containing the length of the record. Parameter UEPFCRSP does not apply to this error.

DBFEPU

An error response has been returned from DFHFCVS while servicing a PUT UPDATE request. DFHDBP has attempted to replace the existing copy of the record on the file with the "before-copy" held in the dynamic log record pointed to by UEPDLOGR. Use parameter UEPFCRSP to determine which error occurred.

DBFEPN

An error response has been returned from DFHFCVS while servicing a PUT NEW request. DFHDBP has attempted to add the "before-copy" of a deleted VSAM KSDS record. Use parameter UEPFCRSP to determine the specific error.

DBFEWA

If the record read from the restart data set is a WRITE ADD, the record is read in from the file using a GET UPDATE. For BDAM and VSAM-ESDS data sets, no delete function exists. Because DFHDBP does not attempt to delete the record that has been added, you may, in your exit program, "mark" the existing record on the file as deleted according to application-dependent logic. Mark

dynamic transaction backout program exits

the version of the record to which UEPFDATA points. If you want to reapply this version, specify return code 'UERCRTY'. UEPFLEN points to a field containing the length of the record.

Parameter UEPFCSRSP does not apply; it contains a normal response from the READ UPDATE.

DBFEVD

An error response has been returned from the DFHFCVS while servicing a VSAM-DELETE request. DFHDBP has attempted to delete a new record added to a VSAM key-sequenced data set or a VSAM relative-record data set. Use parameter UEPFCSRSP to determine the specific error.

XDBDERR

Table 19. Exit XDBDERR

When invoked	When the DL/I backout routine detects an error, its error message is routed to CSMT and exit XDBDERR is given control.
Exit-specific parameters	UEPDLOGR Address of dynamic log record. The dynamic log record can be mapped using the DSECT DFHDBRDS. UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM Continue processing. Note: Any databases updated by DL/I are now closed by DL/I backout failure control. UERCPURG Task purged during XPI call.
XPI calls	All can be used.

EXEC interface program exits XEIN and XEIOU

There are two global user exit points in the EXEC interface program. XEIN is invoked before the execution of an EXEC CICS command. The sequence is:

TRACE – XEIN – EDF – command

XEIOU is invoked after execution of an EXEC CICS command. The sequence is:

command – EDF – XEIOU – TRACE

Note: Asynchronous processing of XEIN and XEIOU may occur if the transaction is suspended (for example, during file I/O wait). This situation may also occur under CEDF because it issues its own EXEC CICS commands between the application's XEIN and XEIOU exits.

Thus if the same GWA is shared between the XEIN and XEIOU exits, these possibilities must be allowed for to ensure integrity of the data and to prevent unpredictable results.

On entry to the exits, the exit-specific parameter UEPARG contains the address of the command parameter list.

The command parameter list

The first parameter in the list points to a string of data known as **argument 0**. The other parameters point to the values specified for the parameters passed on the command.

Argument 0 begins with a 2-byte function code that identifies the command. (Function codes are documented in Appendix A of the *CICS/ESA Application Programming Reference* manual.) The function code is followed by a 2-byte field containing "existence bits", that indicate whether arguments are passed on the command. For example, consider the command:

```
EXEC CICS LINK PROGRAM('MYPROG')
```

Here, argument 0 begins with the function code X'0E02' (LINK). Existence bit 1 is set, indicating that there is an argument 1 (namely, 'MYPROG').

The correspondence between command parameters (such as PROGRAM) and their positions and values in the parameter list (in this case, argument 1, 'MYPROG') can be deduced from the translated code for the particular command.

EXEC interface program exits

Warning: Modifying CICS commands by tampering with argument 0 is *not* supported, and leads to unexpected errors or results.

For example, if an application program is written in assembler or PL/I and you modify argument 0, you will be writing to program storage (that is, storage occupied by the program itself), which could cause 0C4 abends. Furthermore, modifying argument 0 not only alters the CICS command for *this execution* of the command in the application program, it changes the CICS command in the virtual storage copy of the application program. This means that the next task to invoke the same copy of the program will also execute the modified command.

This particular example of the danger of tampering with argument 0 does not apply to COBOL or C/370 application programs, but nevertheless you should not modify CICS commands for application programs written in any supported language.

Bypassing commands

Your XEIIIN exit program can bypass execution of a command by setting the UERCBYP return code. If you do this, EDF is not invoked, but XEIOUT and exit trace are invoked if they are active.

Before setting UERCBYP, your program should check the value pointed to by UEPPGM, to ensure that it is not bypassing an EXEC CICS command issued by CICS.

XEIIIN

<i>Table 20. Exit XEIIIN</i>		
When invoked	Before the execution of an EXEC CICS command.	
Exit-specific parameters	UEPARG	Address of the EXEC command parameter list.
	UEPEXECB	Address of the system EIB.
	UEPUSID	Address of the 8-character userid.
	UEPPGM	Address of the 8-character application program name.
	UEPLOAD	Address of the application program's load-point.
Return codes	UERCNORM	Continue processing.
	UERCBYP	Bypass the execution of this command.
	UERC PURG	Task purged during XPI call.
XPI calls	All can be used.	

XEIOUT

<i>Table 21. Exit XEIOUT</i>		
When invoked	After the execution of an EXEC CICS command.	
Exit-specific parameters	UEPARG	Address of the EXEC command parameter list.
	UEPEXECB	Address of the system EIB.
	UEPUSID	Address of the 8-character userid.
	UEPPGM	Address of the 8-character application program name.
	UEPLOAD	Address of the application program's load-point.
	UEPRSA	Address of the application's register save area. This contains the contents of the registers at the point when the program issued the EXEC CICS command.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

Note: The UEPGROUP exit-specific parameter, passed to XEIOUT in previous releases, is obsolete. You must modify any XEIOUT exit programs that use the UEPGROUP parameter.

File control open/close program exit XFCNREC

You can use XFCNREC to suppress the open failure that occurs when a mismatch is detected between the backout recovery setting for the file and its associated data set.

XFCNREC is invoked only if a mismatch is detected.

It is intended for use by those who wish to continue with open processing even though the backout recovery settings for different files associated with the same base data set are not consistent.

After an open failure has been suppressed, CICS can no longer guarantee integrity for the data set and will mark it as such. Any subsequent EXEC CICS INQUIRE DSNAME OR CEMT INQUIRE DSNAME RECOVSTATUS will return NOTRECOVABLE. Logging continues for the data set for requests via any file that has BACKOUT on its definitions but not for those that do not have BACKOUT.

The mismatch state of the data set survives until the next EXEC CICS or CEMT SET DSNAME REMOVE or COLD START (if the associated data set is not in backout failed state).

At the point at which the mismatch is accepted, CICS issues a message to warn that integrity can no longer be guaranteed.

The order in which files are opened for the same base data set will determine the content of the message received on suppression of an open failure using XFCNREC. If the base cluster block is set as unrecoverable and a mismatch has been allowed, access may be allowed to the data set, via an unrecoverable file, before the data set is fully recovered.

To provide a means of selecting which mismatches to accept and which to reject, three parameters are passed to the exit. These are the address of the filename, the address of the base data set name, and the address of a byte containing the file backout indicator. Because the exit is driven only if there is a mismatch, the data set backout indicator can be derived from the setting for the file.

Note: If XFCNREC is used to suppress an open failure due to a mismatch, the global user exit XFCSREQC will pass the base data set backout setting as the exit parameter UEFBCLOG, and not the file backout setting, which may be different.

Table 22. Exit XFCNREC

When invoked	Before file open when a mismatch is detected between the backout recovery setting for the file and its associated data set.	
Exit-specific parameters	UEFILE	Address of the 8-bit file name. If the file name is less than 8 characters in length, it will be padded with blanks.
	UEDSETN	Address of the 44-byte base data set name. If the data set name is less than 44 characters in length, it will be padded with blanks. Note: In previous releases of CICS, this parameter was called UEDSNAME. If you are migrating exit programs from previous releases to CICS/ESA 4.1, all references to UEDSNAME must be changed to UEDSETN.
	UEPFRCV	Address of a 1-byte field containing the backout recovery setting for the file. Possible values are: UEFBCLOG Backout specified for file.
Return codes	UERCNORM	Fail open as normal.
	UERCBYBYP	Bypass open failure—accept mismatch.
XPI calls	Must not be used.	
SPI calls	Must not be used.	
API calls	Must not be used.	

File control EXEC interface program exits XFCREQ and XFCREQC

The XFCREQ exit allows you to intercept a file control request before any action has been taken on it by file control. The XFCREQC exit allows you to intercept a file control request after file control has completed its processing.

The XFCREQ and XFCREQC exits can be written only in assembler language.

Using XFCREQ, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Modify any value specified by the request before the command is executed.
- Set return codes to specify that either:
 - CICS should continue with the (possibly modified) request.
 - CICS should bypass the request. (Note that if you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.)

Using XFCREQC, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

Both exits are passed seven parameters as follows:

- The address of the command-level parameter structure
- The address of a token (UEPFCTOK) used to pass 4 bytes of data from XFCREQ to XFCREQC
- The addresses of copies of three pieces of return code information from the EIB
- The address of a token (UEPTSTOK) that is valid throughout the life of a task
- The address of a recursion count field.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request. (For example, the second address always points to the file name.)

Only the first 8 addresses and the last address can be referenced by the user exit. Addresses nine through eleven are reserved for CICS internal use.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request. (For example, you could change the name of the file involved in the request.)

+

End of parameter list indicator+
+
+
+
+

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

The original parameter list, as it was before XFCREQ was invoked, is restored after the completion of XFCREQC. It follows that the execution diagnostic facility (EDF) displays the original command before **and** after execution. **EDF does not display any changes made by the exit.**

+
+
+
+

The UEPCLPS exit-specific parameter: The UEPCLPS exit-specific parameter is included in both exit XFCREQ and exit XFCREQC. It is the address of the command-level parameter structure. The command-level parameter structure contains 12 addresses, FC_ADDR0 through FC_ADDRB. It is defined in the DSECT FC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHFCEDS.

The command-level parameter list is made up as follows:

FC_ADDR0

is the address of a 9-byte area called the EID, which is made up as follows:

FC_GROUP
FC_FUNCT
FC_BITS1
FC_BITS2
FC_EIDOPT5
FC_EIDOPT6
FC_EIDOPT7
FC_EIDOPT8

+

The name of the DSECT mapping the EID is FC_EID.

FC_GROUP Always X'06', indicating that this is a file control request.

FC_FUNCT One byte that defines the type of request:

X'02' READ
X'04' WRITE
X'06' REWRITE
X'08' DELETE
X'0A' UNLOCK
X'0C' STARTBR
X'0E' READNEXT
X'10' READPREV
X'12' ENDBR
X'14' RESETBR

FC_BITS1

Existence bits that define which keywords that contain values were specified. To obtain the value associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If

the existence bit is set off, the keyword was not specified in the request and the address should not be used.

- X'80'** Set if the request contains the keyword FILE. If set, **FC_ADDR1** is meaningful.
- X'40'** Set if the request contains any of the keywords INTO, SET, or FROM. If set, **FC_ADDR2** is meaningful.
- X'20'** Set if the request specifies LENGTH or NUMREC, or if a STARTBR, RESETBR, or ENDBR request specifies REQID. If set, **FC_ADDR3** is meaningful.
- X'10'** Set if the request specifies RIDFLD. If set, **FC_ADDR4** is meaningful.
- X'08'** Set if the request specifies KEYLENGTH. If set, **FC_ADDR5** is meaningful.
- X'04'** Set if the request is READNEXT or READPREV and specifies REQID. If set, **FC_ADDR6** is meaningful.
- X'02'** Set if the request specifies SYSID. If set, **FC_ADDR7** is meaningful.
- X'01'** Not used by file control.

+ **FC_BITS2**

Second set of existence bits.

- + **X'20'** Set if the request specifies TOKEN. If set, **FC_ADDRB** is meaningful.

+ **FC_EIDOPT5**

Indicates whether certain keywords that do not take values were specified on the request.

- X'04'** MASSINSERT specified.
- X'02'** RRN specified.
- X'01'** SET (and not INTO) was specified.

— **APAR PN 84038** —

APAR PN84038 documented 21st May 1996.

Note: Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

+ **FC_EIDOPT6**

Indicates whether certain keywords that do not take values were specified on the request.

- X'80'** RBA specified.
- X'40'** GENERIC specified.
- X'20'** GTEQ specified.

— **APAR PN 84038** —

APAR PN84038 documented 21st May 1996.

Note: Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

FC_EIDOPT7 Indicates whether certain keywords that do not take values were specified on the request.

X'04' UPDATE specified.

X'01' Either DEBREC or DEBKEY specified. (See **FC_EIDOPT8**)

+

APAR PN 84038

APAR PN84038 documented 21st May 1996.

+

+

+

+

Note: Your program must test for keywords at the bit level, because there may be more than one of these keywords present.

FC_EIDOPT8 Indicates whether certain keywords that do not take values were specified on the request.

X'80' DEBKEY specified.

X'40' DEBREC specified.

FC_ADDR1

is the address of an 8-byte area containing the name from FILE.

FC_ADDR2

is the address of one of the following:

+

+

+

+

+

- A 4-byte address returned for SET (if the request is READ, READNEXT, or READPREV, and if **FC_EIDOPT5** indicates that this is SET).
- Data returned for INTO (if the request is READ, READNEXT, or READPREV, and if **FC_EIDOPT5** indicates that this is not SET).
- Data from FROM (if the request is WRITE or REWRITE).

FC_ADDR3

is the address of one of the following:

- The halfword value of LENGTH (if the request is READ, WRITE, REWRITE, READNEXT, or READPREV).

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

+

- The returned halfword value of NUMREC (if the request is DELETE).
- The halfword value of REQID (if the request is STARTBR, RESETBR, or ENDBR).

FC_ADDR4

is the address of an area containing the value of the RIDFLD keyword.

FC_ADDR5

is the address of the halfword value of KEYLENGTH.

FC_ADDR6

is the address of the halfword value of REQID (if the request is READNEXT or READPREV).

FC_ADDR7

is the address of an area containing the value of SYSID.

file control EXEC interface program exits

FC_ADDR8

is the address of a value intended for CICS internal use only. It must not be used.

FC_ADDR9

is the address of a value intended for CICS internal use only. It must not be used.

FC_ADDRA

is the address of a value intended for CICS internal use only. It must not be used.

+
+
+

FC_ADDRB

is the address of the fullword value of TOKEN (if the request is READ, REWRITE, DELETE, or UNLOCK).

Modifying fields in the command-level parameter structure

Some fields that are passed to file control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

A list of input and output fields: The following are always input fields:

FILE
FROM
KEYLENGTH
REQID
SYSID

The following are always output fields:

INTO
NUMREC
SET

Whether LENGTH and RIDFLD are input or output fields depends on the request, as shown in Table 23. A dash (—) means that the keyword cannot be specified on the request.

Request	LENGTH	RIDFLD
READ	Output	See Note 1.
WRITE	Input	See Note 2.
REWRITE	Input	—
DELETE	—	Input
UNLOCK	—	—
STARTBR	—	Input
READNEXT	Output	Output
READPREV	Output	Output
ENDBR	—	—
RESETBR	—	Input

Notes:

1. Normally, this is an input field. However, if UPDATE is specified and the file is a BDAM file using extended key search, RIDFLD is used for both input and output.
2. The use of RIDFLD on a WRITE request depends on the file type. For a VSAM KSDS or RRDS, or a fixed-format BDAM file, RIDFLD is an input field. For all other file types, it is used either for output only, or for both input and output, and should be treated like an output field.

Modifying input fields: The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Warning: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in “Modifying input fields” is not suitable for modifying output fields. (The results would be returned to the new area instead of the application’s area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application’s data in place, because the application is expecting the field to be modified anyway.

Modifying fields used for both input and output: An example of a field that is used for both input and output is LENGTH on a READ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a WRITE request to a READ request.

However, you can make minor changes to requests, such as to turn on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

FC_BITS1

- X'20'** The existence bit for LENGTH, NUMREC, or REQID
- X'08'** The existence bit for KEYLENGTH
- X'04'** The existence bit for REQID if the request is READNEXT or READPREV
- X'02'** The existence bit for SYSID.

FC_EIDOPT5

file control EXEC interface program exits

X'04' MASSINSERT specified.

FC_EIDOPT6

X'40' GENERIC specified

X'20' GTEQ specified.

Bits in the EID should be modified in place. You should not modify the pointer to the EID: any attempt to do so is ignored by CICS.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the file control request only.

Warning: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that are permitted. For instance, it is possible to change a DELETE into a GENERIC DELETE, but to make such a change may be dangerous.

Use of the task token UEPTSTOK

UEPTSTOK provides the address of a 4-byte area that you can use to pass information between successive file control requests in the same task. (By contrast, UEPFCTOK is usable only for the duration of a single file control request, because its contents may be destroyed at the end of the request.) For example, if you need to pass information between successive invocations of the XFCREQ exit, UEPTSTOK provides a means of doing this.

Use of the parameter UEPFSHIP

— **APAR PN80580** —

Documentation for PN80580 added on 15 March 1996.

UEPFSHIP contains the address of a 16-byte area. This area consists of 4 characters, followed by 3 fullwords. If the first byte contains 'Y', this request has been function shipped to this region. In this case, if your exit program wants to bypass file control (by setting a return code of UERCBYP), it must set the 3 fullwords as follows:

Fullword 1 The length of the buffer area

Fullword 2 The length of the record

Fullword 3 The length of the modified RIDFLD.

Doing this ensures that the data and RIDFLD are correctly shipped back.

The EIB

Copies of EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion information in XFCREQ and XFCREQC
- Examine completion information in XFCREQC.

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. File Control copies your values into the real EIB after

the completion of XFCREQC; or if you specify a return code of 'bypass' in XFCREQ.

You must set valid file control responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by File Control to describe a valid completion. **File Control does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2.** To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by File Control are specified in DFHFCEDS.

Example of how XFCREQ and XFCREQC can be used

XFCREQ and XFCREQC can be used for a variety of purposes. One example of a possible use is given below.

In this example, XFCREQ and XFCREQC are used to obtain a record containing compressed data, to decompress the data, and to return it to the area specified by the user program as INTO. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the function.

In XFCREQ:

1. Issue an EXEC CICS GETMAIN to obtain an area large enough to hold the decompressed data.
2. Change the INTO pointer to point to this new area, so that File Control uses it when it processes the request. (The decompressed data is copied to the *user's* INTO area, and the INTO pointer reset, before return to the application program—see stages 4 and 7 of the processing to be done by XFCREQC.)
3. Set UEPFCTOK to be the address of the new area so that XFCREQC can also use this area.
4. Return to CICS.

In XFCREQC:

1. Check 'UEPRCODE' to make sure that the file control request completed without error.
2. Use UEPFCTOK to find the address of the area. This area now holds the compressed data.
3. Decompress the data in place.
4. Copy the data from the new area to the user's INTO area. Use the user-specified LENGTH (from the command-level parameter list) to ensure that the data fits and that the copy does not cause a storage violation.
5. Set 'LENGERR' in UEPRESP, UEPRESP2, and UEPRCODE if the data does not fit.
6. Use EXEC CICS FREEMAIN to free the work area pointed to by UEPFCTOK.
7. At this point the command-level parameter list points to the now free area as the address for INTO. This is not a problem, because after completion of XFCREQC File Control restores this pointer to point to the area supplied by the user program.

8. Return to CICS.

XFCREQ

<i>Table 24. Exit XFCREQ</i>	
When invoked	Before CICS processes a file control request.
Exit-specific parameters	<p>UEPCLPS Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 57.</p> <p>UEPFCTOK Address of the 4-byte token to be passed to XFCREQC. This allows you, for example, to pass a work area to exit XFCREQC.</p> <p>UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.</p> <p>UEPRES P Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.</p> <p>UEPRES P2 Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.</p> <p>UEPTSTOK Address of a 4-byte token that is valid throughout the life of a task. See “Use of the task token UEPTSTOK” on page 62.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p> <p>UEPFSHIP</p> <p>APAR PN80580: Documentation for PN80580 added on 15 March 1996.</p> <p>Address of a 16 byte area. See “Use of the parameter UEPFSHIP” on page 62.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERC BYP The file control EXEC interface program should ignore this request.</p> <p>UERC PURG Task purged during XPI call.</p>
XPI calls	<p>All can be used. You can also use certain API calls at this user exit (see below).</p> <p>Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.</p>
API calls	All can be used.

Warning:

Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCREQ exit. Use of the recursion counter UEPRECUR is recommended.

Notes:

1. Exit programs that issue EXEC CICS commands must first address the EIB. See “Using CICS services” on page 5.
2. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See “Returning values to CICS” on page 10.

XFCREQC

Table 25. Exit XFCREQC

When invoked	After a file control request has completed, and before return from the file control EXEC interface program.	
Exit-specific parameters	UEPCLPS	Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 57.
	UEPFCTOK	Address of the 4 byte token to be passed to XFCREQC. This allows you, for example, to pass a work area to exit XFCREQC.
	UEPRCODE	Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.
	UEPRES P	Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.
		Note: If the file that has just been accessed is remote, the addressed field contains zeros (even if UEPRCODE is non-zero).
	UEPRES P2	Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.
		Note: If the file that has just been accessed is remote, the addressed field contains zeros (even if UEPRCODE is non-zero).
	UEPTSTOK	Address of a 4-byte token that is valid throughout the life of a task. See “Use of the task token UEPTSTOK” on page 62.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing.
	UERC PURG	Task purged during XPI call.
XPI calls	All can be used. You can also use certain EXEC CICS API commands at this user exit (see below). Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.	
API calls	All can be used.	

file control EXEC interface program exits

Warning: No EXEC CICS commands other than those listed above are allowed in the XFCREQ and XFCREQC exits. Any attempt to use them will lead to unpredictable results, normally a failure.

Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCREQC exit.

Use of the recursion counter UEPRECUR is recommended.

Notes:

1. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 5.
2. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

Example program

CICS supplies—in hardcopy only—an example program, DFH\$XTSE, that shows how to modify fields in the command-level parameter structure passed to EXEC interface exits. DFH\$XTSE is listed on page 717.

File control file state program exits XFCSREQ and XFCSREQC

Two user exits are provided in the file control state program. You can use XFCSREQ, which is invoked **before** a file ENABLE, DISABLE, OPEN, or CLOSE request is acted on, to gather information about the state of the file—for example, which file requests (SERVREQs) are valid, which journaling options are set. Based on this information, you can suppress the request, if appropriate. (See return code UERCBYP on page 69.)

You can use XFCSREQC, which is invoked **after** the file request has been acted on, to gather information about the data set associated with the file—for example, which recovery options are set. Note that XFCSREQC is invoked even if you have used XFCSREQ to suppress the file request.

For ENABLE, DISABLE, and OPEN requests, each exit is invoked only once. However, for CLOSE requests, because a file can be quiesced before actual closure, the exits might be invoked more than once.

For a single CLOSE request, XFCSREQ and XFCSREQC are invoked more than once if closure is attempted while the file is being accessed by other tasks. For example, the result of a CLOSE NOWAIT command in these circumstances is that XFCSREQ is invoked before the closure is attempted. Because there are still users of the file, the closure is delayed. However, because it specified NOWAIT, the CLOSE request completes, and invokes XFCSREQC with UEPFSRSP set to 'UEFSPEND', meaning closure is pending. When all activity against the file is complete, the file is closed, and XFCSREQ and XFCSREQC are invoked under the task that actually closed it.

For a CLOSE WAIT command, the exits are invoked as follows. XFCSREQ is invoked, the task requests a closure of the file and waits for the closure to happen. When all activity against the file is complete, the file is closed, and XFCSREQ and XFCSREQC are invoked under the task that closed it. Finally, because the closure has now been completed, the task that issued the CLOSE WAIT is resumed, completes its CLOSE request, and invokes XFCSREQC.

Note: There are three occasions when the user exits XFCSREQ and XFCSREQC are not invoked during a close request:

1. On a controlled, non-immediate shutdown of CICS, when CICS closes all files.
2. After an I/O error has been encountered on a VSAM file. (For an I/O error the file is not closed immediately. CICS quiesces the file, setting the status to 'close pending'. When all activity against the file, including the task receiving the I/O error is complete, CICS issues a subsequent close request and XFCSREQ and XFCSREQC are invoked.)
3. After loading a user maintained data table. When the data table load has completed the source dataset is no longer required. CICS subsequently closes and de-allocates the file, leaving the data table open.

XFCSREQ

Table 26 (Page 1 of 2). Exit XFCSREQ

When invoked	Before a file ENABLE, DISABLE, OPEN, or CLOSE is attempted.	
+ + + + + + + + + + + + + + +	Exit-specific parameters	<p>UEPFSREQ Address of a 2-byte field that indicates the type of file request. The first byte contains one of the following values:</p> <p style="margin-left: 20px;">UEPFSOPN Open request UEPFSCLS Close request UEPFSENB Enable request UEPFSDIS Disable request.</p> <p>If the first byte indicates a close request (UEPFSCLS), the second byte shows the type of close:</p> <p style="margin-left: 20px;">UEPFSNC Normal close UEPFSCP Close pending UEPFSELM End of load mode close.</p> <p>UEPFILE Address of the 8-byte file name.</p> <p>UEPFINFO Address of a storage area containing information about the file. The area can be mapped using the DSECT DFHUEFDS, which contains the following fields:</p> <p style="margin-left: 20px;">UEFLNAME The 8-character file name. UEDSNAME The 44-character dsname of the data set associated with the file, if this has been set before the file request was issued. UEFSERV One byte indicating the SERVREQ settings for this file. The possible values are:</p> <p style="margin-left: 40px;">UEFRDIM Read valid UEFUPDIM Update valid UEFADDIM Add valid UEFDELIM Delete valid UEFBRZIM Browse valid.</p> <p style="margin-left: 20px;">UEFDSJL One byte indicating the automatic journaling options set for this file. The possible values are:</p> <p style="margin-left: 40px;">UEFJRO Journal read-only. UEFJRU Journal read for update. UEFJWU Journal write update. UEFJWA Journal write add. UEFJDSN Dsname has been journaled. UEFJSYN Journal read synchronously. UEFJASY Journal write asynchronously.</p>

Table 26 (Page 2 of 2). Exit XFCSREQ

	<p>UEFDSVJL One byte indicating a further automatic journaling option which applies to VSAM files only. The value is:</p> <p>UEFJWAC Write add complete.</p> <p>UEFDSJID One byte containing the number of the journal to be used for automatic journaling, if any.</p> <p>UEFDSACC One byte indicating the access method of the file. The possible values are:</p> <p>UEFVSAM VSAM file UEFBDAM BDAM file.</p> <p>UEFBCRV Set to nulls for this exit.</p> <p>UEFFRLOG Set to nulls for this exit.</p> <p>UEFCDATE Set to nulls for this exit.</p> <p>UEFCTIME Set to nulls for this exit.</p> <p>UEFBCBS Set to nulls for this exit.</p> <p>UEFACBCP Set to nulls for this exit.</p> <p>Note: Only the first seven fields of UEPFINFO are set for this exit. The remaining fields are set to nulls.</p> <p>APAR PN 81921</p> <p>Information for APAR PN 81921 added 13th May 1996.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYB Suppress the file request. The specific effect of setting UERCBYB depends on the type of file request. In the case of a file ENABLE, DISABLE, or CLOSE request, the request is ignored; that is, the state of the file is not changed. In the case of a file OPEN request, the file is set to a closed, unenabled state. In either case, CICS issues message DFHFC0996:</p> <p>OPEN/CLOSE/ENABLE/DISABLE of file <u>filename</u> suppressed due to intervention of user exit</p> <p>Note that, in one special case, setting UERCBYB to suppress a file request has no effect. If a file is to be closed as a result of backout failure processing, CICS does not allow the CLOSE request to be suppressed, as this would cause a failure in backout failure processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	All can be used.
API calls	All can be used.

#

+

+

+

+

+

+

+

+
+
+
+
+
+
+
+
+
+

<p>APAR PN 81921</p> <p>Information for APAR PN 81921 added 13th May 1996.</p>

Warning:

Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCSREQ exit. Use of the recursion counter UEPRECUR is recommended.

Notes:

1. Exit programs that issue EXEC CICS commands must first address the EIB. See "Using CICS services" on page 5.
2. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

XFCSREQC

Table 27 (Page 1 of 3). Exit XFCSREQC

When invoked	After a file ENABLE, DISABLE, OPEN, or CLOSE command has completed.	
Exit-specific parameters	UEPFSREQ	Address of a 2-byte field that indicates the type of file request. The first byte contains one of the following values:
	UEPFSOPN	Open request
	UEPFSCLS	Close request
	UEPFSENB	Enable request
	UEPFSDIS	Disable request.
		If the first byte indicates a close request (UEPFSCLS), the second byte shows the type of close:
	UEPFSNC	Normal close
	UEPFSCLP	Close pending
	UEPFSCLM	End of load mode close.
	UEPFILE	Address of the 8-byte file name.
	UEPFINFO	Address of a storage area containing information about the file. The area can be mapped using the DSECT DFHUEFDS, which contains the following fields:
	UEFLNAME	The 8-character file name.
	UEDSNAME	The 44-character dsname of the data set associated with the file.
	UEFSERV	One byte indicating the SERVREQ settings for this file. The possible values are:
	UEFRDIM	Read valid
	UEFUPDIM	Update valid
	UEFADDIM	Add valid
	UEFDELIM	Delete valid
	UEFBRZIM	Browse valid.
	UEFDSJL	One byte indicating the automatic journaling options set for this file. The possible values are:
	UEFJRO	Journal read-only.
	UEFJRU	Journal read for update.
	UEFJWU	Journal write update.
	UEFJWA	Journal write add.
	UEFJDSN	Dsname has been journaled.
	UEFJSYN	Journal read synchronously.
	UEFJASY	Journal write asynchronously.
	UEFDSVJL	One byte indicating a further automatic journaling option which applies to VSAM files only. The value is:
	UEFJWAC	Write add complete.

Table 27 (Page 2 of 3). Exit XFCSREQC

UEFDSJID	One byte containing the number of the journal to be used for automatic journaling, if any.
UEFDSACC	One byte indicating the access method of the file. The possible values are: UEFVSAM VSAM file UEFBDAM BDAM file.
UEFBCRV	One byte indicating the recovery attributes of the data set associated with this file. The possible values are: UEFBCFR Forward recovery specified UEFBCLOG Logging specified UEFBCVAL Flag indicating that recovery attributes are valid.
UEFFRLOG	One byte containing the number of the journal to be used for forward recovery, if any.
UEFCDATE	A date (YYYYDDD+) in packed decimal format. This field is set only when the file is the last file to close against the VSAM sphere with which it is associated. It contains the date when activity against the VSAM sphere was quiesced.
UEFCTIME	A time (HHMMSST+) in packed decimal format. This field is set only when the file is the last file to close against the VSAM sphere with which it is associated. It contains the time when activity against the VSAM sphere was quiesced.
UEFBCBS	A byte containing flags indicating the backout failure status of this data set. The possible values are: UEFBCBF Backout failed UEFBCBE Backout failure complete.
UEFACBCP	Address of a readonly copy of the ACB (for a VSAM file) or the DCB (for a BDAM file).

#

Table 27 (Page 3 of 3). Exit XFCSREQC

UEPFSRSP Address of a byte containing the return codes for the request. This has one of the following values:

- UEFSNORM** Normal response.
- UEFSWARN** Warning response.
- UEFSFAIL** Failure response.
- UEFSPEND** Pending response. The 'Pending' response can be returned only after a CLOSE request. It indicates that, as a result of the CLOSE request, a closure is pending on the file, the file is being quiesced. When all activity against the file has completed, it is closed. Note that, if enabled, the XFCSREQ and XFCSREQC exits are driven again, when the actual closure takes place.

APAR PN 81921

Information for APAR PN 81921 added 13th May 1996.

UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.

Notes:

1. The first seven fields of UEPFINFO (UEFLNAME through UEFDACC) are set for all requests; that is, following an OPEN, CLOSE, ENABLE, or DISABLE request.
2. The next two fields (UEFBCRV and UEFFRLOG) are set only after a successful OPEN request. After all other requests, or if the OPEN fails, these fields are set to nulls.
3. Fields UEFCDATE through UEFCBS are set only after a successful CLOSE request. After all other requests, if the file is already closed, if the closure fails, or if the closure is pending, these fields are set to nulls.
4. Field UEFACBCP is set only after a successful OPEN request. After all other requests, or if the OPEN fails, the field is set to nulls.

Return codes **UERCNORM** Continue processing.
UERCPURG Task purged during XPI call.

XPI calls All can be used.

API calls All can be used.

APAR PN 81921

Information for APAR PN 81921 added 13th May 1996.

Warning:

Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a file control request is issued from the XFCSREQC exit. Use of the recursion counter UEPRECUR is recommended.

file control file state program exits

+
+
+
+
+
+

Notes:

1. Exit programs that issue EXEC CICS commands must first address the EIB.
See "Using CICS services" on page 5.
2. Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See "Returning values to CICS" on page 10.

Front End Programming Interface exits **XSZARQ** and **XSZBRQ**

Exits XSZARQ and XSZBRQ are invoked from the CICS/ESA Front End Programming Interface (FEPI), if FEPI is installed. For details of these exits, see the *CICS/ESA Front End Programming Interface User's Guide*.

“Good morning” message program exit XGMTEXT

<i>Table 28. Exit XGMTEXT</i>		
When invoked		Before the “good morning” message is transmitted.
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls		All can be used.

Intersystem communication program exits XISCONA and XISLCLQ

The two exits in the intersystem communication program allow you to control the queuing of function shipping requests.

Note: There are several methods that you can use to control the length of intersystem queues. For a description of the available methods, see the *CICS/ESA Intercommunication Guide*.

The XISCONA exit

Important

It is recommended that you use the XZIQUE exit, in the VTAM working-set module, to control the length of intersystem queues, rather than XISCONA. (XZIQUE is described on page 188.) XZIQUE provides more functions, and is of more general use than XISCONA (it is driven for transaction routing, DPL, and distributed transaction processing requests, as well as for function shipping, whereas XISCONA is driven only for function shipping). If you enable both exits, XZIQUE and XISCONA could both be driven for function shipping requests, which is not recommended.

If you already have an XISCONA exit program, you may be able to modify it for use at the XZIQUE exit point.

The purpose of XISCONA is to help you prevent the performance problems that can occur when function shipping requests awaiting free sessions for a connection are queued in the issuing region. The exit permits you to control the number of outstanding ALLOCATE requests by allowing you to reject any function shipping request that would otherwise be queued.

Requests that are to be function shipped to a resource-owning region are queued by default if all bound contention winner⁴ sessions are busy, so that no sessions are immediately available. If the resource-owning region is unresponsive (for example, if it is a file-owning region, it may be waiting for a system journal to be archived), the queue can become so long that the performance of the issuing region is severely impaired. Further, if the issuing region is an application-owning region, its impaired performance can spread back to the terminal-owning region.

To control the queuing of function shipping requests, use the XISCONA exit to tell CICS, whenever a session cannot be allocated immediately, whether to queue the request, or to return 'SYSIDERR' to the application. The exit works like this:

1. If the XISCONA exit program is **not** active, CICS queues the request when necessary.
2. If the exit program is active, it is invoked *only if all bound contention winner sessions are in use*. For other failures (for example, 'Mode name not found' or 'Out of service'), CICS bypasses the exit and returns to the application.

⁴ "Contention winner" is the terminology used for LU6.2 connections. The XISCONA exit applies also to MRO and LU6.1 connections: in these, the SEND sessions (defined in the session definitions) are used first for ALLOCATE requests; when all SEND sessions are in use, queuing starts.

3. If it is invoked, your exit program must decide whether or not to queue the request by analyzing the statistics provided through the user exit parameter list. Your exit program could:

- Stipulate that queuing is never to be used. This is the simplest way to code the exit, and avoids complexities of tuning. It should be effective if you define enough contention winner sessions to handle the peak transaction load for the connection. If you suppress all queuing, you must specify AUTOCONNECT(YES) on the SESSIONS definition, because the queuing mechanism no longer binds sessions for you.

With this approach, a danger arises if you base your estimate of required sessions on average conditions and the transaction load subsequently varies widely; when CICS cannot use queuing to cope with the variation, users may suffer transaction abends when there is no significant problem in the resource-owning region.

- Examine the number of requests currently in the queue. The program could, for example, stop queuing when the number exceeds 120% of the maximum number of sessions. You could use this approach to cope with intermittent stoppages in the resource-owning region.

You could use a table of thresholds for the connections in your system, with values determined from previous experience of queuing problems. Alternatively, you could use the EXEC CICS interface in a separate program to inquire about the state of the connection, and pass the information in a work area to the XISCONA exit program.

- Examine the type of request and the resource being accessed (which can be discovered by examining the request parameter list). The program could, for example, reject file read requests but queue file updates.

Warning: Because a failure of the exit program could affect system availability, it is recommended that you make the logic of your program as simple as possible, thus reducing the possibility of errors.

There are some function shipping problems that XISCONA cannot solve. For example, if you have specified both a large number of sessions and a large value for MXT, CICS may develop the short-on-storage (SOS) condition *before* XISCONA is invoked because there are no further sessions available.

The sample XISCONA global user exit program, DFHXIS

Note that there is a CICS-supplied sample exit program, DFHXIS, that shows one way of limiting the queue of ALLOCATE requests, based on the information passed to the program. For more information about the sample global user exit programs, see “Sample global user exit programs” on page 14.

<i>Table 29 (Page 1 of 2). Exit XISCONA</i>		
When invoked		After an attempt to function ship a request to a remote system has failed because all bound contention winner sessions are in use.
Exit-specific parameters	UEPISPCA	<p>Address of a parameter list containing the following fields. You can map the parameter list using the DSECT DFHXISDS.</p> <p>UEPCONST Address of the Connection statistics record.</p> <p>Connection statistics records are of type STICONSR (STID value 52). Your exit program can map the record using the DSECT DFHA14DS. See notes.</p> <p>UEPMODST Address of the Mode Entry statistics record, or zero. A Mode Entry statistics record is built <i>only</i> if:</p> <ul style="list-style-type: none"> • The connection-type is LU6.2 (see field UEPCONTY). • The profile DFHCICSF (which is always used for function shipping) defines a specific MODENAME to be used in the allocation of LU6.2 sessions. <p>Mode Entry statistics records are of type STICONMR (STID value 76). Your exit program can map the record (if present) using the DSECT DFHA20DS.</p> <p>UEPEIPPL Address of the request parameter list.</p> <p>UEPCONTY A 1-byte field indicating the connection-type. Possible values are:</p> <p>UEPMRO (X'80') Request for an MRO connection</p> <p>UEPLU6 (X'40') Request for an LU6.1 connection</p> <p>UEPLUC (X'20') Request for an LU6.2 connection.</p>

<i>Table 29 (Page 2 of 2). Exit XISCONA</i>	
UEPNETNM	
An 8-character field containing the NETNAME for the connection— that is, the identifier (applid) of the remote CICS region or system.	
Notes:	
<ol style="list-style-type: none"> 1. The general format of statistics records is described in “CICS statistics record format” on page 583. 2. For a list of statistics record-types and their associated copy books, see Figure 72 on page 587. 3. For a description of the fields in Connection and Mode Entry statistics records, see the <i>CICS/ESA Performance Guide</i>. 	
Return codes	UERCAQUE Queue the request. This is the default. UERCAPUR Do not queue the request, unless local queuing is possible.
XPI calls	All can be used.

Important

There is no ‘UERCNORM’ return code at this exit point, because the exit is invoked after a failure. The choice is whether or not to take the system default action of queuing the request.

The XISLCLQ exit

XISLCLQ enables you to specify what action to take after a function shipping request fails to allocate a session with a remote system for one of the following reasons:

- The remote system is not in service.
- A connection to the remote system cannot be established.
- No sessions are immediately available, and your XISCONA exit program has specified that the request is not to be queued in the issuing region.

Note that this exit is invoked *only* if the request to be shipped is of type EXEC CICS START NOCHECK. For EXEC CICS requests other than those with the NOCHECK option (which is only available on START commands) the ‘SYSIDERR’ condition is raised in the application program.

You can use the exit to specify whether or not the failed request is to be locally queued, to be executed when the connection is reestablished.

<i>Table 30. Exit XISLCLQ</i>	
When invoked	After a function shipping request of type EXEC CICS START NOCHECK has failed because the remote system is not in service, a connection to the remote system cannot be established, or no sessions are immediately available, and your XISCONA exit program has specified that the request is not to be queued in the issuing region.
Exit-specific parameters	<p>UEPISPP Address of a parameter list that contains:</p> <p>UEPTCTSE Address of the relevant terminal control table system entry. The TCT system entry can be mapped using the DSECT DFHTCTTE.</p> <p>UEPXXTE Address of the local transaction name, or 0 if SYSID was specified in the command.</p> <p>Note: Your program can use the transaction manager XPI call INQUIRE_TRANDEF to obtain details of the local transaction (see page 318).</p> <p>UEPPLIST Address of the parameter list for the command.</p> <p>Note: No DSECT is provided for the above parameter list. You have to code your own DSECT to access the named fields.</p>
Return codes	<p>UERCYSYS Take the system action. This is determined by the value of the LOCALQ attribute in the local TRANSACTION definition for the remote transaction:</p> <p>LOCALQ(YES) The request is queued locally.</p> <p>LOCALQ(NO) 'SYSIDERR' is returned to the application program.</p> <p>UERCQUE Queue the request locally (overriding the LOCALQ(NO) attribute, if specified).</p> <p>UERCIGN Override the LOCALQ(YES) attribute, if specified, and return with 'SYSIDERR'.</p> <p>UERCPUrg Task purged during XPI call.</p>
XPI calls	All can be used.

Important

There is no 'UERCNORM' return code at this exit point, because the exit is invoked after a failure. The choice is whether to take the system default action or to handle the error in some other way.

Interval control program exits XICREQ, XICEXP, and XICTENF

You can use some XPI calls in exit programs invoked from the interval control program. However, when any of these exits are invoked for expiry analysis, any actions that delay the execution of the interval control program can have adverse effects on other transactions that are waiting for intervals to expire. You can determine whether the exits have been invoked for expiry analysis by examining the type-of-request field, TCAICTR, a copy of which is pointed to by the UEPICRQ1 exit-specific parameter.

Note: The XICREQ exit is invoked by internal requests made by CICS code, as well as by requests made by applications. For example, if you use the CICS extended recovery facility (XRF), the XRF surveillance program uses interval control services. DFHXRSP issues an interval control WAIT every 2 seconds; this means that any interval control exit programs are also invoked every 2 seconds.

XICREQ

<i>Table 31 (Page 1 of 2). Exit XICREQ</i>	
When invoked	At the beginning of the interval control program, before request analysis.
Exit-specific parameters	UEPICQID Address of an 8-byte field containing the request ID parameter on request. See notes below.
	UEPICTID Address of a 4-byte field containing the terminal ID, if any, specified on an EXEC CICS START command. See notes below.
	UEPICTI Address of 4 bytes containing the transaction ID specified on an EXEC CICS START command. See notes below.
	UEPICRQ1 Address of a 1-byte field containing a copy of TCAICTR, the first request code field for requests to the interval control program.
	UEPICRQ2 Address of a 1-byte field containing a copy of TCAICTR2, the second request code field for requests to the interval control program.
	UEPICRT Address of a 4-byte field containing the expiry time or interval, in packed decimal format. The value is in the form 0HHMMSSF, where H=hours, M=minutes, S=seconds, and F is a positive sign.
Notes:	
<ol style="list-style-type: none"> 1. The contents of the fields addressed by UEPICQID and UEPICTID are unpredictable if the associated data items were not specified on the request. You must test the copy of TCAICTR to determine whether they contain meaningful values. 2. Your exit program can change the values of the fields addressed by UEPICQID, UEPICTID, UEPICTI, and UEPICRT. Changing the values of the fields addressed by UEPICRQ1 or UEPICRQ2 has no effect. 	
Return codes	UERCNORM Continue processing.
	UERCPURG Task purged during XPI call.

Table 31 (Page 2 of 2). Exit XICREQ

XPI calls	The following must not be used: ADD_SUSPEND DELETE_SUSPEND RESUME SUSPEND WAIT_MVS.
------------------	--

XICEXP

Table 32. Exit XICEXP

When invoked	After an interval control time interval has expired.
Exit-specific parameters	UEPICE Address of the interval control element (ICE) that has just expired. The ICE can be mapped using the DSECT DFHICEDS.
Return codes	UERCNORM Continue processing. UERCPURG Task purged during XPI call.
XPI calls	The following must not be used: ADD_SUSPEND DELETE_SUSPEND RESUME SUSPEND WAIT_MVS.

XICTENF

Table 33. Exit XICTENF

When invoked	Exit XICTENF is also invoked from the interval control program. However, this exit relates to the 'terminal not known' condition and so is considered in detail in "Terminal not known' condition exits XALTENF and XICTENF" on page 157.
---------------------	---

Interval control EXEC interface program exits XICEREQ and XICEREQC

XICEREQ is invoked on entry to the interval control program before CICS processes an interval control request. Using XICEREQ, you can:

- Analyze the request to determine its type, the keywords specified, and their values.
- Modify any value specified by the request before the command is executed.
- Set return codes to specify that either:
 - CICS should continue with the request, modified or unmodified.
 - CICS should bypass the request. (Note that if you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.)

XICEREQC is invoked after the interval control program request is completed. Using XICEREQC, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

+

CICS passes seven parameters to these exits as follows:

- The address of the command-level parameter structure (UEPCLPS)
- The address of a token (UEPICTOK) used to pass 4 bytes of data from XICEREQ to XICEREQC
- The addresses of copies of three pieces of return code information from the EIB
- The address of a token (UEPTSTOK) that is valid throughout the life of a task
- The address of an exit recursion count (UEPRECUR).

Note: The XICEREQ exit is invoked by internal requests made by CICS code, as well as by requests made by applications.

XICEREQ

<i>Table 34 (Page 1 of 2). Exit XICEREQ</i>	
When invoked	Before CICS processes an interval control API request.

<i>Table 34 (Page 2 of 2). Exit XICEREQ</i>	
Exit-specific parameters	UEPCLPS Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 88.
	UEPICTOK Address of a 4-byte token to be passed to XICEREQC. This allows you, for example, to pass a work area to exit XICEREQC.
	UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.
	UEPRESP Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.
	UEPRESP2 Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.
	UEPTSTOK Address of a 4-byte token that is valid throughout the life of a task. See “Using the task token UEPTSTOK” on page 94.
	UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM Continue processing.
	UERCBYP The interval control EXEC interface program should ignore this request.
	UERCPURG Task purged during XPI call.
XPI calls	All can be used. You can also use EXEC CICS API commands at this user exit. Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, you are recommended to use the EXEC CICS GETMAIN and FREEMAIN commands instead.
API commands	All can be used.

Warning: Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when an interval control request is issued from the XICEREQ exit. Use of the recursion counter UEPRECUR is recommended.

XICEREQC

<i>Table 35. Exit XICEREQC</i>	
When invoked	After an interval control API request has completed, and before return from the interval control EXEC interface program.
Exit-specific parameters	<p>UEPCLPS Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 88.</p> <p>UEPICTOK Address of a 4 byte token passed from XICEREQ. This allows XICEREQ to, for example, pass a work area to exit XICEREQC.</p> <p>UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.</p> <p>UEPRES Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.</p> <p>UEPRES2 Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.</p> <p>UEPTSTOK Address of a 4-byte token that is valid throughout the life of a task. See “Using the task token UEPTSTOK” on page 94.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>All can be used.</p> <p>Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, you are recommended to use the EXEC CICS GETMAIN and FREEMAIN commands instead.</p>
API commands	All can be used.

Warning: Care should be taken when issuing recursive commands. For example, you must avoid entering a loop when issuing an interval control request from the XICEREQC exit. Use of the recursion counter UEPRECUR is recommended.

The command-level parameter structure

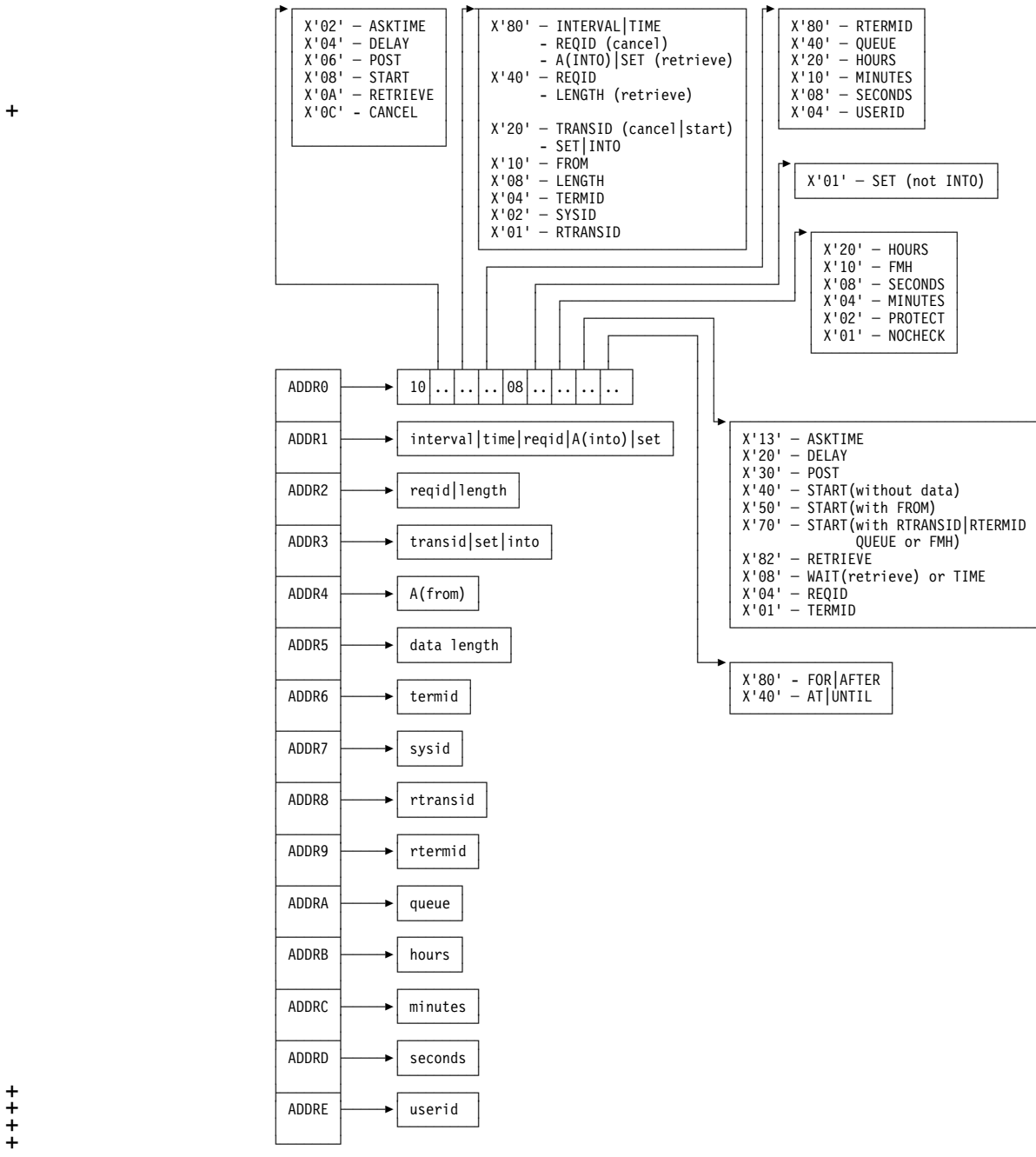


Figure 1. The command-level parameter structure for interval control

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a 9-byte area that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request. For example, the second address points to the interval for START requests.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the

interval control EXEC interface program exits

values of the keywords. You can also modify values of keywords specified on the request. For example, you could change the SYSID specified in the request.

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first four addresses (IC_ADDR0, the address of the EID, to IC_ADDR3, the address of the name of the transaction named in a START request), the high-order bit is set on in IC_ADDR3. If you extend the parameter list by setting the address of a SYSID in IC_ADDR7, you must unset the high-order bit in IC_ADDR3 and set it on in IC_ADDR7 instead.

The maximum size of parameter list is supplied to the exit, thus allowing your exit program to add any parameters not already specified without needing to first obtain more storage.

The original parameter list, as it was before XICEREQ was invoked, is restored after the completion of XICEREQC. It follows that the execution diagnostic facility (EDF) displays the original command before **and** after execution: **EDF does not display any changes made by the exit.**

The UEPCLPS exit-specific parameter: The UEPCLPS exit-specific parameter is included in both exit XICEREQ and exit XICEREQC. It is the address of the command-level parameter structure. The command-level parameter structure contains 15 addresses, IC_ADDR0 through IC_ADDRE. It is defined in the DSECT IC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHICUED.

+
+
+
+

The command-level parameter list is made up as follows:

IC_ADDR0

is the address of a 9-byte area called the EXEC interface descriptor (EID), which is made up as follows:

IC_GROUP
IC_FUNCT
IC_BITS1
IC_BITS2
IC_EIDOPT5
IC_EIDOPT6
IC_EIDOPT7
IC_EIDOPT8
IC_EIDOPT9
IC_EIDOPT10
IC_EIDOPT11
IC_EIDOPT12
IC_EIDOPT13

IC_GROUP	Always X'10', indicating that this is an interval control request.
IC_FUNCT	One byte that defines the type of request: X'02' ASKTIME X'04' DELAY X'06' POST X'08' START X'0A' RETRIEVE X'0C' CANCEL
IC_BITS1	Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used. X'80' Set if the request contains INTERVAL or TIME arguments, or if a CANCEL request specifies REQID, or if a RETRIEVE request specifies SET or INTO. If set, IC_ADDR1 is meaningful. X'40' Set if the request other than CANCEL specifies REQID or if a RETRIEVE request specifies LENGTH. If set, IC_ADDR2 is meaningful. X'20' Set if the request specifies TRANSID or if a request other than RETRIEVE specifies SET or INTO. If set, IC_ADDR3 is meaningful. X'10' Set if the request specifies FROM. If set, IC_ADDR4 is meaningful. X'08' Set if a request other than RETRIEVE specifies LENGTH. If set, IC_ADDR5 is meaningful. X'04' Set if the request specifies TERMID. If set, IC_ADDR6 is meaningful. X'02' Set if the request specifies SYSID. If set, IC_ADDR7 is meaningful. X'01' Set if the request specifies RTRANSID. If set, IC_ADDR8 is meaningful.
IC_BITS2	Further argument existence bits. X'80' Set if the request specifies RTERMID. If set, IC_ADDR9 is meaningful. X'40' Set if the request specifies QUEUE. If set, IC_ADDRA is meaningful. X'20' Set if the request specifies HOURS. If set, IC_ADDRB is meaningful. X'10' Set if the request specifies MINUTES. If set, IC_ADDR C is meaningful. X'08' Set if the request specifies SECONDS. If set, IC_ADDRD is meaningful. X'04' Set if the request specifies USERID. If set, IC_ADDRE is meaningful.
IC_BITS3	One byte not used by interval control.

+
+

interval control EXEC interface program exits

	IC_EIDOPT5	Indicates whether certain keywords were specified on the request.
+		APAR PN92143 added 20th February 1997.
+	X'01'	On a RETRIEVE command, SET (and not INTO) was specified. On a START command, ATTACH was specified. You cannot modify this field in your user exit.
+		
+		
+	IC_EIDOPT6	Existence bits that indicate whether certain keywords were specified on the request.
+	X'20'	HOURS specified.
+	X'10'	FMH specified.
+	X'08'	SECONDS specified.
+	X'04'	MINUTES specified.
+	X'02'	PROTECT specified.
+	X'01'	NOCHECK specified.
	IC_EIDOPT7	Indicates whether certain functions or keywords were specified on the request.
	X'F0'	CANCEL specified.
	X'82'	RETRIEVE specified.
	X'40'	START specified.
	X'30'	POST specified.
	X'20'	DELAY, RTRANSID, RTERMID, or QUEUE specified, and/or FMH.
	X'13'	ASKTIME specified.
	X'10'	FROM, RTRANSID, or RTERMID specified, and/or QUEUE.
	X'08'	TIME or WAIT specified.
	X'04'	REQID specified.
	X'01'	TERMID specified.
	IC_EIDOPT8	Indicates whether certain keywords were specified on the request.
	X'80'	FOR or AFTER specified.
	X'40'	AT or UNTIL specified.

IC_ADDR1

is the address of one of the following:

- An 8-byte area containing the value of the INTERVAL keyword (or TIME keyword if **IC_EIDOPT7** indicates that TIME is specified).
- An 8-byte area containing the value of REQID (if the request is CANCEL).
- Data returned for INTO (if the request is RETRIEVE, and if **IC_EIDOPT5** indicates that this is not SET).
- A 4-byte address returned for SET (if the request is RETRIEVE and **IC_EIDOPT5** indicates that this is SET).

IC_ADDR2

is the address of one of the following:

- An 8-byte area containing the value of REQID (if the request is DELAY, POST or START).

- A halfword containing the value of LENGTH (if the request is RETRIEVE).

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

IC_ADDR3

is the address of one of the following:

- An area containing the value of TRANSID (if the request is CANCEL or START).
- A 4-byte address returned for SET (if the request is START or POST and IC_EIDOPT5 indicates that this is SET).

+
+

IC_ADDR4

is the address of an area containing the data from FROM.

IC_ADDR5

is the address of the halfword value of LENGTH.

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

IC_ADDR6

is the address of an area containing the value of TERMID.

IC_ADDR7

is the address of an area containing the value of SYSID.

IC_ADDR8

is the address of an area containing the value of RTRANSID.

IC_ADDR9

is the address of an area containing the value of RTERMID.

IC_ADDRA

is the address of an area containing the value of QUEUE.

IC_ADDRB

is the address of an area containing the value of HOURS.

IC_ADDR C

is the address of an area containing the value of MINUTES.

IC_ADDRD

is the address of an area containing the value of SECONDS.

+
+

IC_ADDRE

is the address of an area containing the value of USERID.

Modifying fields in the command-level parameter structure: Some fields that are passed to interval control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

INTERVAL
TIME

interval control EXEC interface program exits

REQID
FROM
TERMID
SYSID
HOURS
MINUTES
SECONDS
USERID

+

The following are always output fields:

INTO
SET

The following are input fields on a START request and output fields on a RETRIEVE request:

RTRANSID
RTERMID
QUEUE

+

+

LENGTH is an input field on a START request, an output field on a RETRIEVE with SET specified, and an input/output field on a RETRIEVE with INTO specified.

Modifying input fields: The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Warning: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in “Modifying input fields” is not suitable for modifying output fields. (The results would be returned to the new area instead of the application’s area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application’s data in place, because the application is expecting the field to be modified anyway.

Modifying the EID: It is not possible to modify the EID to make major changes to requests, such as changing a DELAY request to a START request.

However, you can make minor changes to requests, such as turning on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

Some interval control commands use 2 bits in the EID to indicate a single keyword; the EXEC CICS START command, for example, uses 2 bits to indicate TERMID. The first bit, in IC_BITS1, indicates that ADDR6 in the command parameter list is valid (ADDR6 points to TERMID) and the second, in IC_EIDOPT7, is the keyword existence bit to show that the TERMID keyword was specified on the command.

Where this occurs you must ensure that both bit settings are changed (consistently) if you wish to modify these commands from within a user exit program, or the results will be unpredictable.

The list that follows shows the bits in the EID that **can** be modified. Any attempt to modify any other part of the EID is ignored.

IC_BITS1

X'80'	The existence bit for REQID (if the request is CANCEL)
X'40'	The existence bit for LENGTH (if the request is RETRIEVE) or REQID
X'10'	The existence bit for FROM
X'08'	The existence bit for LENGTH
X'04'	The existence bit for TERMID
X'02'	The existence bit for SYSID
X'01'	The existence bit for RTRANSID

IC_BITS2

X'80'	The existence bit for RTERMID
X'40'	The existence bit for QUEUE
X'20'	The existence bit for HOURS
X'10'	The existence bit for MINUTES
X'08'	The existence bit for SECONDS

IC_EIDOPT6

X'20'	Secondary existence bit for HOURS
X'10'	Existence bit for FMH
X'08'	Secondary existence bit for SECONDS
X'04'	Secondary existence bit for MINUTES
X'02'	Existence bit for PROTECT
X'01'	Existence bit for NOCHECK

IC_EIDOPT7

Bits in IC_EIDOPT7 should only be modified within the same functional group – that is, only those existence bits defined as valid for a START request should be set on a START request.

ASKTIME requests

X'13' ASKTIME request. This value is fixed for all ASKTIME requests, and should not be modified.

DELAY requests

X'20'	DELAY request
X'08'	TIME specified
X'04'	REQID specified

POST requests

X'30'	POST request
X'08'	TIME specified
X'04'	REQID specified

interval control EXEC interface program exits

START requests

- X'40' START request (without DATA)
- X'50' START with DATA request
- X'70' START with one or more of RTRANSID, RTERMID, QUEUE, or FMH specified.
- X'08' TIME specified
- X'04' REQID specified
- X'01' TERMID specified

RETRIEVE requests

- X'82' RETRIEVE request

CANCEL requests

- X'F0' CANCEL request
- X'04' REQID specified

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the interval control request only. **Warning:** Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that **are** permitted.

Using the interval control request token UEPICTOK: UEPICTOK provides the address of a 4-byte area that you can use to pass information between the XICEREQ and XICEREQC user exits for the same interval control request. For example, the address of a piece of storage obtained by the XICEREQ user exit, which is to be freed by the XICEREQC exit, can be passed in the UEPICTOK field.

+ **Using the task token UEPTSTOK:** UEPTSTOK provides the address of a 4-byte
+ area that you can use to pass information between successive interval control
+ requests in the same task. (By contrast, UEPICTOK is usable only for the duration
+ of a single interval control request, because its contents may be destroyed at the
+ end of the request.) For example, if you need to pass information between
+ successive invocations of the XICEREQ exit, UEPTSTOK provides a means of
+ doing this.

The EIB: Copies of EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion information in XICEREQ and XICEREQC
- Examine completion information in XICEREQC.

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. Interval control copies your values into the real EIB after the completion of XICEREQC; or if you specify a return code of 'bypass' in XICEREQ.

You must set valid interval control responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by CICS interval control to describe a valid completion. **CICS does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2.** However, if EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS overrides EIBRESP with a non-zero value. To aid you in setting the values of EIBRCODE,

EIBRESP, and EIBRESP2, the values used by interval control are specified in DFHICUED.

Example of how XICEREQ and XICEREQC can be used: XICEREQ and XICEREQC can be used for a variety of purposes. One example of a possible use is given below.

In this example, XICEREQ and XICEREQC are used to route START requests to a number of different CICS regions to provide a simple load balancing mechanism. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the function.

In XICEREQ:

1. Scan the global work area (GWA) to locate a suitable CICS region (for example, the region currently processing the least number of START requests).
2. Having decided which system to route the request to, increment the use count for this system.
3. Obtain a 4-byte area in which to store the SYSID for this request. This can be allocated from the GWA to avoid issuing a GETMAIN. If the area is obtained by issuing a GETMAIN, set UEPICTOK to the address of the storage obtained.
4. Set IC_ADDR7 to be the address of the 4-byte area so that XICEREQC can also use this area.
5. If setting IC_ADDR7 now makes it the last address, set the high-order bit in the address, and reset the high-order bit in what was previously the last address.
6. Set the X'02' existence bit on in IC_BITS1 to indicate that a SYSID is specified.
7. Return to CICS.

In XICEREQC:

1. Scan the global work area (GWA) and locate the entry for the CICS region specified in the SYSID parameter.
2. Decrement the use count for this system.
3. If a GETMAIN was issued in XICEREQ to obtain an area to hold the SYSID, issue a FREEMAIN for the address held in UEPICTOK.
4. Return to CICS.

Example program

CICS supplies—in hardcopy only—an example program, DFH\$XTSE, that shows how to modify fields in the command-level parameter structure passed to EXEC interface exits. DFH\$XTSE is listed on page 717.

Journal control program exits XJCWR and XJCWB

There are two global user exits in the journal control program. CICS invokes XJCWR after building a journal record in the buffer, but before writing it to its target journal. XJCWB is invoked before each physical WRITE to each non-SMF journal. It could be used, for example, to pass copies of CICS log records to another system to aid in recovery of the prime-site system after a disaster.

XJCWR

Table 36. Exit XJCWR

When invoked	After a journal record is built in the buffer and before it is written to its target journal.	
Exit-specific parameters	UEPJCTE	Address of the JCT entry of the journal being written to. The JCT entry can be mapped using the DSECT DFHJCTTE.
	UEPJREC	Address of the journal record. The journal control record can be mapped using the information supplied in Chapter 18, "CICS journaling" on page 521.
Return codes	UERCNORM	Continue processing.
XPI calls	APAR PN 88244	
	Documented 3rd September 1996.	
	XPI calls must not be used.	

+
+
+

XJCWB*Table 37. Exit XJCWB*

When invoked	Before each physical WRITE to each non-SMF journal.	
Exit-specific parameters	UEPJCJID	Address of a halfword containing the number (ID) of the journal being written to. The value of the journal ID is restricted to the range 1 through 99.
	UEPJCSFX	Address of an area containing the single character suffix of the journal being written to. The suffix is either 'A', 'B', or 'X'.
	UEPJDCB	Address of the BSAM DCB of the journal data set.
	UEPJCBUF	Address of the journal data block. This is the 31-bit address of the current block of journal records.
	UEPJCBFL	Address of a fullword containing the length of the journal data block.
Return codes	UERCNORM	Continue processing.
XPI calls	APAR PN 88244	Documented 3rd September 1996. XPI calls must not be used.

+
+
+

Loader domain exits XLDLOAD and XLDELETE

There are two global user exits in the loader domain. XLDLOAD is invoked when a
 # new instance of a program is loaded into storage, before the program is made
 # available for use.

XLDELETE is invoked after an instance of a program is released by CICS and
 # before the program is freed from storage.

For LPA-resident programs, the exits are still invoked when a program is acquired
 # or released, even though the program is not physically loaded or freed.

These are both information-only exits. Any changes made to the exit parameters
 # by the exit program are ignored by CICS, as is any return code which it sets.

<i>Table 38. Exit XLDLOAD</i>	
When invoked	After an instance of a program is brought into storage, and before the program is made available for use.
Exit-specific parameters	UEPPROGN Address of an 8-character field containing the name of the program that is being loaded.
	UEPPROGL Address of a 4-byte field containing the length, in bytes, of the program that is being loaded.
	UEPLDPT Address of a 4-byte field containing the address at which the program has been loaded.
	UEPENTRY Address of a 4-byte field containing the address of the program's entry point.
	UEPTRANID Zero, or the address of a 4-byte field containing the transaction ID which applied when the exit was invoked.
	UEPUSER Zero, or the address of an 8-byte field containing the userid in control at the time the exit was invoked.
	UEPTERM Zero, or the address of a 4-byte field containing the terminal name associated with the transaction under which the exit was invoked.
UEPPROG Zero, or the address of an 8-character field containing the name of the program that was in control at the time the exit was invoked.	
Return codes	UERCNORM Continue processing.
XPI calls	Must not be used.
API and SPI calls	Must not be used.

Message domain exit XMEOUT

The XMEOUT exit allows you to suppress or reroute CICS messages that use the message domain.

Note that your exit program is subject to certain restrictions:

- It cannot suppress or reroute messages sent to terminal operators, but only those sent to the system console or to transient data queues. (XMEOUT is not invoked for the former type of message.)
- It can only suppress or reroute messages that use the message domain. You can deduce which messages this applies to from the *CICS/ESA Messages and Codes* manual: the description of each message that causes XMEOUT to be driven contains a list of "XMEOUT parameters"; if no XMEOUT parameters are listed for a message, the latter does not cause the exit to be driven. For example, message 'DFHDX8320' causes XMEOUT to be invoked, but message 'DFHDU0205' does not.
- It cannot change the text of a message, nor the message inserts. (If it tries to do so, CICS ignores the changes.)
- It cannot suppress or reroute messages issued during the first phase of CICS initialization (because the exit cannot be enabled before the second phase).
- It cannot reroute a message to transient data queues during CICS shutdown unless the original message destination included one or more transient data queues. If it attempts to do so, the message in question is routed to its original destination, and message DFHME0120I is issued to the console. Message DFHME0120I cannot be re-routed by the user exit program but it may be suppressed.

This restriction is necessary because the message domain is required to handle messages during CICS shutdown even after the transient data queue function has ended.

To discover whether CICS shutdown has started, your exit program can check for the first instance of message DFHME0120. It can stop rerouting messages to TD queues after DFHME0120 has been issued.

APAR PN65433

Documentation for PN65433 added on 16 January 1995.

Note: If a message is being rerouted to a transient data queue and the transient data request fails, the message is lost. The MEME exception trace point ID X'0328' is written. The interpretation string of this trace entry provides an explanation of why the transient data request failed.

Important

Because of the danger of recursion, your XMEOUT exit program should **not** try to reroute:

- Any DFHTDxxxx messages, produced by the transient data program.

APAR PN63068

Documentation for PN63068 added on 28 June 1995.

- User domain messages in the range DFHUS0002–DFHUS0006, plus message DFHUS0150.
- Transaction manager messages DFHXM0212, DFHXM0213, DFHXM0304 and DFHXM0308.
- Application messages DFHAP0001, DFHAP0002, DFHAP0004, DFHAP0601, DFHAP0602, and DFHAP0603.
- Any user domain (DFHUSxxxx) messages to an intrapartition queue defined with a TRIGLEV value of anything other than zero, if the messages are produced while the user domain is performing error recovery processing.

APAR PN63068

Documentation for PN63068 added on 28 June 1995.

The message definition template contains an indicator called “noreroute”. This indicator is set on if the message being issued cannot be rerouted to a transient data queue by the XMEOUT exit program. The address of the indicator is passed to XMEOUT in the UEPN RTE exit-specific parameter. Your exit program can check the value of the indicator before deciding whether or not to reroute a particular message.

Note: If the exit program tries to reroute an ineligible message, the message domain inhibits the rerouting and issues the message to the console instead, along with message DFHME0137.

Each of the messages affected by this restriction is identified by a note in the *CICS/ESA Messages and Codes* manual.

Your exit program can suppress or reroute messages by altering the values held in the addresses pointed to by the UEPMROU, UEPMNRC, UEPMTDQ, and UEPMNTD fields of the parameter list. These four sets of values (route codes, number of route codes, transient data queue names, and number of TDQs) are the only ones that your program can change.

Table 40 (Page 1 of 3). Exit XMEOUT

When invoked	Before the message domain sends a CICS message to its destination.
---------------------	--

Table 40 (Page 2 of 3). Exit XMEOUT

+
+
+
+

Exit-specific parameters	Parameter Name	Description
	UEPTRANID	Address of the 4-byte transaction ID.
	UEPUSER	Address of the 8-byte user ID.
	UEPTERM	Address of the 4-byte ID of the terminal under which the current transaction is running. If the current transaction is not associated with a terminal, the addressed field contains hexadecimal zeroes.
	UEPPROG	Address of the 8-byte application program name.
	UEPMNUM	Address of a 4-byte field containing the message number.
	UEPMDOM	Address of a 2-byte field containing the domain identifier of the CICS message.
	UEPMROU	Address of an array of up to 28 route codes. Route codes must be numbers in the range 1 through 28.
	UEPMNRC	Address of a halfword containing the number of route codes in the route code array.
	UEPMTDQ	Address of an array of up to 25 transient data queue names to which the message is to be sent. TD queue names must consist of 4 alphanumeric characters.
	UEPMNTD	Address of a halfword containing the number of TDQs in the queues array.
	UEPINSN	Address of a 2-byte field containing the number of message inserts.

Table 40 (Page 3 of 3). Exit XMEOUT

<p>UEPINSA Address of an array, each element of which contains information about a single message insert. The size of the array depends on the number of inserts. Each array element has the following structure:</p> <p>INSERT_FORMAT_P DS A Address of the 1-byte insert type-code, which has one of the following hexadecimal values:</p> <p>0 Not present 1 Character 2 Hexadecimal 3 Decimal 4 The insert is a number representing one item in a list of options. (See the example below.)</p> <p>INSERT_P DS A Address of the message insert</p> <p>INSERT_LENGTH_P DS A Address of a fullword containing the length of the insert</p> <p>INSERT_TYPE_P DS A Reserved.</p> <p>You can find the order of the inserts in the array from the entry for the particular message in the <i>CICS/ESA Messages and Codes</i> manual. For example,</p> <p>DFHJC4512 <i>date time applid</i> CICS {System Log Journal} <i>nn</i> no longer available - output volume-switch failure</p> <p>The XMEOUT inserts are <i>date</i>, <i>time</i>, <i>applid</i>, a number, and <i>nn</i>. The fourth insert (number) has a value of 1 or 2, representing 'System Log' or 'Journal' respectively.</p> <p>APAR PN63068: Documentation for PN63068 added on 28 June 1995.</p> <p>UEPNRTE Address of 1-character flag indicating whether or not the message can be rerouted by XMEOUT. The possible values are:</p> <p>X'0' The message can be routed. X'1' The message cannot be routed.</p>	<p>Return codes</p> <p>UERCNORM Continue processing.</p> <p>UERCBYB Suppress the message for all destinations.</p> <p>XPI calls WAIT_MVS can be used. Do not use any other calls.</p>
--	--

+
+
+
+
+
+
+

The sample XMEOUT global user exit programs

CICS supplies the following sample programs, which show you how to use the XMEOUT exit to suppress or reroute messages:

- DFH\$SXP1** Suppress a message by message number
- DFH\$SXP2** Suppress a message by destination route code
- DFH\$SXP3** Suppress a message destined for the CSCS transient data queue (which receives signon and sign-off messages)
- DFH\$SXP4** Reroute a console message to a TDQ
- DFH\$SXP5** Reroute a TDQ message to another TDQ
- DFH\$SXP6** Reroute a TDQ message to a console.

Monitoring domain exit XMNOUT

XMNOUT is invoked before an exception class monitoring record is passed to SMF, and before a performance class monitoring record is written to the performance record buffer. You can use this exit to examine the record, to suppress its output to SMF, or to change the data it contains. You must ensure that any changes you make do not conflict with the dictionary description of the data.

You can also add data to performance class data records. To do this you need to define dummy user event-monitoring points (EMPs) in the monitoring control table (MCT) to reserve data fields of the required size and type.

Table 41 (Page 1 of 2). Exit XMNOUT

When invoked	Before an exception class monitoring record is written to SMF, and before a performance class monitoring record is buffered for a later write to SMF.
Exit-specific parameters	<p>UEPTRANID Address of the 4-byte transaction ID. This field is not available at task termination.</p> <p>UEPUSER Address of the 8-byte user ID. This field is not available at task termination.</p> <p>UEPTERM Address of the 4-byte terminal ID. This field is not available at task termination.</p> <p>UEPPROG Address of the 8-byte application program name. This field is not available at task termination.</p> <p>UEPDICT Address of the dictionary. The sequence of dictionary entries is mapped by the DSECT generated from the macro DFHMCTDR. This field only has meaning for performance class records. If the monitoring record type is exception class (refer to parameter UEPMRTYP), this field is set to 0.</p> <p>UEPDICTE Address of the fullword number of dictionary entries. This field only has meaning for performance class records. If the monitoring record type is exception class (refer to parameter UEPMRTYP), this field is set to 0.</p> <p>UEPFCL Address of the field connector list, containing a series of halfword connector values. This field only has meaning for performance class records. If the monitoring record type is exception class (refer to parameter UEPMRTYP), this field is set to 0.</p> <p>UEPFCLNO Address of the fullword number of field connectors. This field only has meaning for performance class records. If the monitoring record type is exception class (refer to parameter UEPMRTYP), this field is set to 0.</p>

Table 41 (Page 2 of 2). Exit XMNOUT

	UEPMRTYP	Address of the halfword monitoring record type. If the value is 4, the type is exception class. If the value is 3, the type is performance class.
	UEPMRLEN	Address of the fullword monitoring record length.
	UEPMREC	Address of monitoring record, whose length is addressed by the parameter UEPMRELEN.
	UEPSRCTK	Address of the MVS workload manager service reporting class token for the current transaction. If CICS support for MVS workload management is not available, this token is null.
Return codes	UERCNORM	Continue processing.
	UERCBYP	Suppress monitor record output.
	UERCPURG	Task purged during XPI call.
XPI calls	WAIT_MVS can be used. Do not use any other calls.	

Program control program exits XPCREQ, XPCREQC, XPCFTCH, XPCHAIR, XPCTA, and XPCABND

There are six user exit points in the program control program.

XPCREQ and XPCREQC

XPCREQ is invoked by the EXEC interface program before a link request is processed. If the request is a distributed program link, the XPCREQ exit is driven on both sides of the link; that is, in both the client and the server regions. The exit program is passed the address of the application's parameter list (in UEPCPLPS), and can modify this as required. For example, you can use this exit to modify the SYSID at the time of a distributed program link request. One way you can achieve this is to write an application program to manage a list of SYSIDs in a global work area (GWA). The global user exit program can obtain access to the GWA, and use the information stored there to redirect DPL requests.

XPCREQC is invoked after the link request is completed. You can use this exit to pass back a response to the application via the EIBRESP or EIBRESP2 fields. Such responses could be used to keep status information about a link request up-to-date. For example, if a link request fails because a connection is unavailable, XPCREQC could set EIBRESP=500 (a response code not used by CICS) to indicate the failure, enabling the application, in conjunction with the other exit, XPCREQ, to determine a suitable course of action.

Table 42. Exit XPCREQ

When invoked	By the EXEC interface program before a link request is processed.	
Exit-specific parameters	UEPCPLPS	Address of the command parameter list.
	UEPPCTOK	Address of a 4-byte token to be passed to XPCREQC. This allows you, for example, to pass a work area to exit XPCREQC.
	UEPRCODE	Address of a 6-byte hexadecimal copy of EIBRCODE.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
	UEPRES	Address of a 4-byte copy of EIBRESP.
	UEPRES2	Address of a 4-byte copy of EIBRESP2.
	UEPTSTOK	Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 111.
Return codes	UERC	Program control is to ignore the request.
	UERNORM	Continue processing.
	UERPURG	Task purged during XPI call.
XPI calls	All can be used. Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.	
API calls	All can be used.	

<i>Table 43. Exit XPCREQC</i>		
When invoked	On completion of a program control link request.	
Exit-specific parameters	UEPCLPS	Address of the command parameter list.
	UEPPCTOK	Address of a 4-byte token passed from XPCREQ. This allows XPCREQ to, for example, pass a work area to XPCREQC.
	UEPRCODE	Address of a 6-byte hexadecimal copy of EIBRCODE.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
	UEPRES P	Address of a 4-byte copy of EIBRESP.
	UEPRES P2	Address of a 4-byte copy of EIBRESP2.
	UEPTSTOK	Address of a 4-byte token that is valid throughout the life of a task. See "Using the task token UEPTSTOK" on page 111.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used. Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.	
API calls	All can be used.	

Warning: Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when a program control request is issued from the XPCREQ or XPCREQC exits.

Use of the recursion counter UEPRECUR is recommended.

The command parameter structure

The command parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request; for instance, the second address always points to the program name. You can examine the parameters in the list to determine the values of the keywords. You can also modify values of parameters specified on the request. For example, you could change the name of the program involved in the request, or add the SYSID to route the link request to a remote system.

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first two addresses (PC_ADDR0, the address of the EID, and PC_ADDR1, the address of the name of the program named in the link request), the high-order bit is set on in PC_ADDR1. If you extend the parameter list by setting the address of a SYSID in PC_ADDR7, you must unset the high-order bit in PC_ADDR1 and set it on in PC_ADDR7 instead.

The original parameter list, as it was before XPCREQ was invoked, is restored after the completion of XPCREQC. It follows that EDF will display the original command before and after execution: **EDF will not display any changes made by the exit.**

The UEPLPS exit-specific parameter: The UEPLPS exit-specific parameter is included in both exit XPCREQ and exit XPCREQC. It is the address of the command-level parameter structure. The command-level parameter structure contains 9 addresses, PC_ADDR0 through PC_ADDR8. It is defined in the DSECT PC_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHPCEDS.

+
+
+

The command-level parameter list is made up as follows:

PC_ADDR0

is the address of a 7-byte area called the EXEC interface descriptor (EID), which is made up as follows:

- PC_GROUP**
- PC_FUNCT**
- PC_BITS1**
- PC_BITS2**
- PC_EIDOPT5**
- PC_EIDOPT6**

PC_GROUP Always X'0E', indicating that this is a program control request.

PC_FUNCT One byte which defines the type of request, which for XPCREQ and XPCREQC is always X'02', indicating a LINK request.

PC_BITS1 Existence bits that define which keywords that contain values were specified. To obtain the value associated with a keyword, you need to use the appropriate address from the command-level parameter list. Before using this address you must check the associated existence bit to ensure that the address is valid. If the existence bit is set off, the keyword was not specified in the request and the address should not be used. The symbolic and hexadecimal values of the existence bits are as follows:

PC_EXIST1 (X'80 ')

Set if the request contains the keyword PROGRAM. If set, **PC_ADDR1** is meaningful. (This bit should always be set for a LINK request.)

PC_EXIST2 (X'40 ')

Set if the request specifies the COMMAREA parameter. If set, **PC_ADDR2** is meaningful.

PC_EXIST3 (X'20 ')

Set if the request specifies the LENGTH parameter. If set, **PC_ADDR3** is meaningful.

PC_EXIST4 (X'10 ')

Set if the request specifies the INPUTMSG parameter. If set, **PC_ADDR4** is meaningful.

PC_EXIST5 (X'08 ')

Set if the request specifies the INPUTMSGLEN parameter. If set, **PC_ADDR5** is meaningful.

PC_EXIST6 (X'04 ')

Set if the request specifies the DATALENGTH parameter. If set, **PC_ADDR6** is meaningful.

PC_EXIST7 (X'02 ')

Set if the request specifies the SYSID parameter. If set, **PC_ADDR7** is meaningful.

PC_EXIST8 (X'01 ')

Set if the request specifies the TRANSID parameter. If set, **PC_ADDR8** is meaningful.

PC_BITS2 Two bytes not used by program control.

PC_EIDOPT5 Not used by program control.

PC_EIDOPT6 Indicates whether the request specifies the SYNCONRETURN option. If it does, X'80' is set.

PC_ADDR1

is the address of an 8-byte area containing the program name from the PROGRAM parameter.

PC_ADDR2

is the address of the COMMAREA data.

PC_ADDR3

is the address of a 2-byte area containing the length of the COMMAREA, as a half-word binary value.

PC_ADDR4

is the address of the INPUTMSG data.

PC_ADDR5

is the address of a 2-byte area containing the length of the INPUTMSG, as a half-word binary value.

PC_ADDR6

is the address of a 2-byte area containing the length specified on the DATALENGTH parameter, defining how much data is to be sent from the COMMAREA. The length is held as a half-word binary value.

program control program exits

PC_ADDR7

is the address of the 4-byte name of the remote system the LINK request is to be shipped to, as specified on the SYSID parameter.

PC_ADDR8

is the address of the 4-byte name of the mirror transaction to be attached in the remote system, as specified on the TRANSID parameter.

Modifying fields in the command parameter structure

Some fields that are passed to program control are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

Modifying input fields: The correct method of modifying an input field is to create a new copy of it, and to change the address in the command parameter list to point to your new data.

Warning: You must never modify an input field by altering the data that is pointed to by the command parameter list. To do so would corrupt storage belonging to the application program and could cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in “Modifying input fields” is not suitable for modifying output fields. (The results would be returned to the new area instead of the application’s area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field you can modify the application’s data in place, because the application is expecting the field to be modified anyway.

Modifying the EID

It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a LINK request to a different type of Program Control request.

However, you can make minor changes to requests, such as to turn on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

PC_BITS1

X'40'	The existence bit for the COMMAREA
X'20'	The existence bit for LENGTH
X'10'	The existence bit for INPUTMSG
X'08'	The existence bit for INPUTMSGLEN
X'04'	The existence bit for DATALENGTH
X'02'	The existence bit for SYSID
X'01'	The existence bit for TRANSID.

PC_EIDOPT5

Not used for a PC link request.

Bits in the EID should be modified in place. You should not modify the pointer to the EID. (Any attempt to do so is ignored by CICS.)

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the program control request only.

Your user exit program is prevented from making major changes to the EID.

Using the program control request token, UEPPCTOK

UEPPCTOK provides the address of a 4-byte area that you can use to pass information between the XPCREQ and XPCREQC user exits for the same program control request. For example, the address of a piece of storage obtained by the XPCREQ user exit, which has to be freed by the XPCREQC user exit, can be passed in the UEPPCTOK field.

Using the task token UEPTSTOK

+
+
+
+
+
+
+

UEPTSTOK provides the address of a 4-byte area that you can use to pass information between successive program control requests in the same task. (By contrast, UEPPCTOK is usable only for the duration of a single program control request, because its contents may be destroyed at the end of the request.) For example, if you need to pass information between successive invocations of the XPCREQ exit, UEPTSTOK provides a means of doing this.

The EIB

Copies of EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion information in XPCREQ and XPCREQC.
- Examine completion information in XPCREQC.

You can update the copies of EIBRCODE, EIBRESP and EIBRESP2 that you are given in the parameter list. Program Control copies your values into the real EIB after the completion of XPCREQC; or if you specify a return code of 'bypass' in XPCREQ.

You must set valid program control responses. You must set all three of EIBRCODE, EIBRESP and EIBRESP2 to a consistent set of values, such as would be set by Program Control to describe a valid completion. **Program Control does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2.** To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by Program Control are specified in DFHPCEDS.

Example of how XPCREQ and XPCREQC can be used

XPCREQ and XPCREQC can be used for a variety of purposes. One example of a possible use is given below.

In this example, XPCREQ and XPCREQC are used to route LINK requests to a number of different CICS regions to provide a simple load balancing mechanism. The example shows only the capabilities of the exits; it is not intended to indicate an ideal way of achieving the load balancing function. For the purpose of this

example, it is assumed that a global work area (GWA) already exists, and that it contains a list of available SYSIDs together with a count of the number of LINK requests currently being processed by each SYSID.

In XPCREQ:

1. Scan the global work area (GWA) to locate a suitable CICS region – for example, the region currently processing the least number of LINK requests.
2. Having decided which system to route the request to, increment the use count for this system.
3. Obtain a 4-byte area in which to store the SYSID for this request (this can be allocated from the GWA to avoid issuing a GETMAIN). If the area is obtained by issuing a GETMAIN, set UEPPCTOK to the address of the storage obtained.
4. Set PC_ADDR7 to the address of the 4-byte area.
5. If setting PC_ADDR7 now makes it the last address, set the high-order bit in the address, and unset the high-order bit in what was previously the last address.
6. Set the X'02' existence bit on in PC_BITS1 to indicate that a SYSID is specified.
7. Return to CICS.

In XPCREQC:

1. Scan the global work area (GWA) and locate the entry for the CICS region specified in the SYSID parameter.
2. Decrement the use count for this system.
3. If a GETMAIN was issued in XPCREQ to obtain an area to hold the SYSID, issue a FREEMAIN for the address held in UEPPCTOK.
4. Return to CICS.

XPCFTCH

XPCFTCH is invoked before a PPT-defined program (including internal CICS modules) receives control, which could be because it is the first program in a transaction, or as a result of a LINK, XCTL, or HANDLE ABEND PROGRAM request. You can use this exit to modify the entry address used when linking to the program. If the exit sets a return code of zero, or a modified address of zero, the entry address of the original application program is used.

The exit is intended to allow you to pass control to an application program or routine *before the original program is invoked*. This first program, after it has finished its processing, should pass control back to the entry point of the original program. You should not use the exit to cause a program to be invoked *instead of* the original program. If you do so, the results are unpredictable.

If a modified entry address is supplied, the program that is invoked receives control in the execution key that the original application program would have received control in—that is, as specified on the EXECKEY option of the original program’s resource definition.

Table 44 (Page 1 of 2). Exit XPCFTCH

When invoked	Before an application program receives control.	
Exit-specific parameters	UEPPCDS	<p>Address of a storage area that contains program- and terminal-related information, and that can be mapped using the DSECT DFHPCUE. When XPCFTCH is invoked, the following DFHPCUE fields are significant:</p> <p>PCUE_CONTROL_BITS 1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal. A setting of PCUENOTX (X'40') indicates that the program is not command level.</p> <p>PCUE_TASK_NUMBER 3-character packed decimal field containing the task number.</p> <p>PCUE_TRANSACTION_ID 4-character field containing the transaction ID.</p> <p>PCUE_TERMINAL_ID 4-character field containing the terminal ID (if any).</p>

<i>Table 44 (Page 2 of 2). Exit XPCFTCH</i>	
	<p>PCUE_PROGRAM_NAME 8-character field containing the name of the program that is to receive control.</p> <p>PCUE_PROGRAM_LANGUAGE 3-character field containing the language of the program that is to receive control.</p> <p>PCUE_LOAD_POINT The program's load point.</p> <p>PCUE_ENTRY_POINT The program's entry point.</p> <p>PCUE_PROGRAM_SIZE Fullword containing the size of the program, in bytes.</p> <p>PCUE_COMMAREA_ADDRESS Address of the program's communication area.</p> <p>PCUE_COMMAREA_SIZE Fullword containing the length of the program's communication area.</p> <p>PCUE_LOGICAL_LEVEL Fullword containing the number of chained DFHRSADS blocks (that is, logical level).</p> <p>PCUE_BRANCH_ADDRESS Fullword. Use this field to supply an alternative entry address. Set the top bit to specify that the alternative program is to run AMODE (31).</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p> <p>UERCMEA Entry address has been modified.</p>
XPI calls	All can be used.

The sample XPCFTCH global user exit program, DFH\$PCEX

Note that there is a CICS-supplied sample exit program, DFH\$PCEX, that is designed to be driven by the XPCFTCH exit. For more information about DFH\$PCEX, see "Sample global user exit programs" on page 14.

XPCHAIR

XPCHAIR is invoked before a HANDLE ABEND LABEL routine is given control. Note that this occurs only when a program abend causes a branch to an internal abend routine. (When the HANDLE ABEND request specifies PROGRAM, exit XPCFTCH is invoked, as described above.) You can use this exit to supply an alternative handle-abend address. If the exit sets a return code of zero, or an alternative address of zero, CICS passes control to the application program's specified internal routine.

If a modified entry address is supplied, the code that is invoked receives control in the execution key that the internal abend routine would have received control in—that is, the key in force when the EXEC CICS HANDLE ABEND LABEL command was issued.

<i>Table 45 (Page 1 of 2). Exit XPCHAIR</i>		
When invoked	Before a HANDLE ABEND routine is given control.	
Exit-specific parameters	UEPPCDS	<p>Address of a storage area that contains program- and terminal-related information, and that can be mapped using the DSECT DFHPCUE. When XPCHAIR is invoked, the following DFHPCUE fields are significant:</p> <p>PCUE_CONTROL_BITS 1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.</p> <p>PCUE_TASK_NUMBER 3-character packed decimal field containing the task number.</p> <p>PCUE_TRANSACTION_ID 4-character field containing the transaction ID.</p> <p>PCUE_TERMINAL_ID 4-character field containing the terminal ID (if any).</p> <p>PCUE_PROGRAM_NAME 8-character field containing the name of the program that issued the HANDLE ABEND LABEL command.</p> <p>PCUE_LOGICAL_LEVEL Fullword containing the number of chained DFHRSADS blocks (that is, logical level).</p> <p>PCUE_BRANCH_ADDRESS Fullword. Use this field to supply the address of an alternate abend routine. Set the top bit to specify that the alternate abend routine is to run AMODE (31).</p>
	UEPTACB	<p>Address of the transaction abend control block (TACB) for the abend. If the abend occurred because of a program check, the information in the TACB includes:</p> <ul style="list-style-type: none"> • The program status word (PSW) • The registers at the time of the abend • Details of the subspace and access registers current at the time of the abend. <p>You can map the TACB using the DFHTACB TYPE=DSECT macro.</p>

<i>Table 45 (Page 2 of 2). Exit XPCHAIR</i>	
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p> <p>UERCMEA The address of an alternate abend routine has been supplied.</p>
XPI calls	All can be used.

XPCTA

XPCTA is invoked immediately after a transaction abend, and before any processing that might modify the existing environment so that the task could not be resumed. You can use it to:

- Set a resume address, instead of letting CICS process the abend
- Specify the subspace that control is passed in.

If a resume address is passed back, registers 0 through 13 and 15 are restored to their values at the time of the abend. Register 14 is used to branch to the resume address. If the exit sets a return code of zero, or a resume address of zero, CICS processes the abend.

Note: If the transaction abend occurs as a result of a program check or an operating system abend, it is possible that the XDUREQ dump domain exit may be invoked before XPCTA. (For details of XDUREQ, see page 39.)

<i>Table 46 (Page 1 of 2). Exit XPCTA</i>	
When invoked	After an abend and before the environment is modified.
Exit-specific parameters	<p>UEPPCDS Address of a storage area that contains program- and terminal-related information, and that can be mapped using the DSECT DFHPCUE. When XPCTA is invoked, the following DFHPCUE fields are significant:</p> <p>PCUE_CONTROL_BITS 1-byte flag field. A setting of PCUECBTE indicates that the transaction is linked to a terminal.</p> <p>PCUE_TASK_NUMBER 3-character packed decimal field containing the task number.</p> <p>PCUE_TRANSACTION_ID 4-character field containing the transaction ID.</p> <p>PCUE_TERMINAL_ID 4-character field containing the terminal ID (if any).</p> <p>PCUE_PROGRAM_NAME 8-character field containing the name of the failing program.</p> <p>PCUE_LOGICAL_LEVEL Fullword containing the number of chained DFHRSADS blocks (that is, logical level).</p> <p>PCUE_BRANCH_ADDRESS Fullword. You can use this field to supply a resume address. Set the top bit to specify that the resumed task is to run AMODE (31).</p>

<i>Table 46 (Page 2 of 2). Exit XPCTA</i>	
	<p>PCUE_BRANCH_EXECKEY If storage protection is active, you can use this 1-byte field to specify the execution key of the resumed task. The possible values are: PCUE_BRANCH_USER User key PCUE_BRANCH_CICS CICS key.</p> <p>If storage protection is active, and you do not specify a value, the resumed task executes in User key.</p> <p>If storage protection is not active, the resumed task executes in CICS key.</p>
UEPTACB	<p>Address of the transaction abend control block (TACB) for the abend. If the abend occurred because of a program check, the information in the TACB includes:</p> <ul style="list-style-type: none"> • The program status word (PSW) • The registers at the time of the abend • The execution key at the time of the abend • Details of the subspace and access registers current at the time of the abend. <p>You can map the TACB using the DFHTACB TYPE=DSECT macro.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p> <p>UERCMEA A resume address has been supplied.</p>
XPI calls	All can be used.

The sample XPCTA global user exit program, DFH\$PCTA

The sample program tests whether the abend was caused by the application program trying to overwrite CICS-key storage in the CDSA or ECDSA, while running in user key. If this was the case, the sample changes the execution key to CICS, and retries the failing instruction.

You can use the sample program to identify, without abending, those programs that need to be defined with EXECKEY(CICS), because they intentionally modify a CICS-key DSA. For details of how to do this, see the prolog of DFH\$PCTA.

DFH\$PCTA can be extended for transaction isolation.

Resource manager interface program exits XRMIIN and XRMIOUT

XRMIIN

Table 48. Exit XRMIIN

When invoked	Before a task-related user exit program is invoked due to an application program issuing an RMI API request.
Exit-specific parameters	<p>UEPTRUEN Address of the name of the task-related user exit program.</p> <p>UEPTRUEP Address of the parameter list to be passed to the task-related user exit program. See note.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p> <p>Note: The task-related user exit program's parameter list is mapped by a DFHUEPAR DSECT that shares common field names with the global user exit program's DFHUEPAR parameter list. To include both DSECT definitions in your exit program, you must code:</p> <pre>DFHUEXIT TYPE=EP, ID=XRMIIN DFHUEXIT TYPE, TYPE=RM</pre> <p>The statements must be coded in this order.</p> <p>The two DFHUEPAR parameter lists, the global user exit's and the task-related user exit's, occupy separate areas of storage. The task-related user exit's parameter list is provided for information only; you should not amend it in any way.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	All can be used.
API commands	All can be used. However, CALLDLI, EXEC DLI, or EXEC SQL commands must not be used.

XRMIOUT

Table 49. Exit XRMIOUT

When invoked	After a task-related user exit program has returned from handling an RMI API request.
Exit-specific parameters	<p>UEPTRUEN Address of the name of the task-related user exit program.</p> <p>UEPTRUEP Address of the parameter list passed to the task-related user exit program. See note.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p> <p>Note: The task-related user exit program's parameter list is mapped by a DFHUEPAR DSECT that shares common field names with the global user exit program's DFHUEPAR parameter list. To include both DSECT definitions in your exit program, you must code:</p> <pre>DFHUEXIT TYPE=EP, ID=XRMIOUT DFHUEXIT TYPE, TYPE=RM</pre> <p>The statements must be coded in this order.</p> <p>The two DFHUEPAR parameter lists, the global user exit's and the task-related user exit's, occupy separate areas of storage. The task-related user exit's parameter list is provided for information only; you should not amend it in any way.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	All can be used.
API commands	All can be used. However, CALLDLI, EXEC DLI, or EXEC SQL commands must not be used.

Warning: It is not recommended that your exit program make calls to other external resource managers that use the RMI, because this causes recursion, and may result in a loop. It is your exit program's responsibility to avoid entering a loop. It could use the recursion counter field UEPRECUR to guard against this.

Resource management install and discard exit XRSINDI

The XRSINDI global user exit is driven, if it is enabled, immediately after CICS successfully installs or discards a resource definition.

The install and discard activities that drive the exit are as follows:

- The install function of the group list on a cold start of CICS
- The CEDA INSTALL command
- All autoinstall operations, as follows:
 - The autoinstall of a terminal, connection, program, mapset, or partitionset
 - The automatic discard of an unused terminal, controlled by the AILDELAY system initialization parameter and the SIGNOFF parameter on the TYPETERM resource definition.
- A CEMT DISCARD and EXEC CICS DISCARD command
- The front-end programming interface (FEPI) install and discard operations: the EXEC CICS FEPI INSTALL command and EXEC CICS FEPI DISCARD command.

The parameter list is designed to pass the names of more than one resource installed or discarded, in field UEPIDNAM. When designing your global user exit program, do not assume that the number of resource names passed is always one. You are recommended to analyze the resources within a loop based on the value referenced by UEPIDNUM.

Note that the names of modegroups are prefixed with the corresponding connection name. There is no separator between the two names: the first four characters form the connection name, followed by eight characters for the modegroup. The parts of the concatenated name are fixed length—if connection names are defined with less than four characters, they are padded with blanks in the concatenated names. Similarly, the connection names for a front-end programming interface (FEPI) connection is a concatenation of a FEPI node name and a FEPI target name, each of which is 8 characters long (fixed length) with no separator.

The exit is driven once for each individual resource in a group list installed during a CICS cold start. If you are concerned about the performance overhead on a cold start, you should not enable the exit until after the group list is installed. To obtain the information about resources installed prior to enabling the exit, you can write a program to scan the tables of installed resources, using the EXEC CICS INQUIRE *resource_name* browse function.

<i>Table 50 (Page 1 of 2). Exit XRSINDI</i>	
When invoked	Whenever CICS installs or discards a resource definition.
Exit-specific parameters	<p>UEPTRANID Address of the 4-byte transaction ID.</p> <p>UEPUSER Address of the 8-byte user ID.</p> <p>UEPTERM Address of the 4-byte terminal ID.</p> <p>UEPPROG Address of the 8-byte application program name.</p> <p>UEPIDREQ Address of the 1-byte install or discard identifier. The values are:</p> <p style="padding-left: 20px;">UEIDINS This request is for an install.</p> <p style="padding-left: 20px;">UEIDDIS This request is for a discard.</p> <p>UEPIDTYP Address of the 1-byte type of resource. The values are:</p> <p style="padding-left: 20px;">UEIDTRAN A transaction</p> <p style="padding-left: 20px;">UEIDPROF A profile</p> <p style="padding-left: 20px;">UEIDPROG A program</p> <p style="padding-left: 20px;">UEIDMAP A mapset</p> <p style="padding-left: 20px;">UEIDPSET A partitionset</p> <p style="padding-left: 20px;">UEIDTERM A terminal</p> <p style="padding-left: 20px;">UEIDCONN A connection</p> <p style="padding-left: 20px;">UEIDMODE A modegroup</p> <p style="padding-left: 20px;">UEIDSESS A session</p> <p style="padding-left: 20px;">UEIDFILE A file</p> <p style="padding-left: 20px;">UEIDPART A partner</p> <p style="padding-left: 20px;">UEIDTCLS A transaction class</p> <p style="padding-left: 20px;">UEIDAITM An autoinstall terminal model</p> <p style="padding-left: 20px;">UEIDFECO A FEPI connection</p> <p style="padding-left: 20px;">UEIDFENO A FEPI node</p> <p style="padding-left: 20px;">UEIDFEPO A FEPI pool</p> <p style="padding-left: 20px;">UEIDFEPS A FEPI propertyset</p> <p style="padding-left: 20px;">UEIDFETA A FEPI target.</p> <p>UEPIDLEN Address of the length of an individual resource name, as a full-word binary value.</p> <p>UEPIDNUM Address of the number of resources reported by this call, as a full-word binary value.</p> <p>UEPIDNAM Address of a variable-length list containing the names of the individual resources reported by this call.</p>

Table 50 (Page 2 of 2). Exit XRSINDI

UEPIDREC	Address of a 1-byte identifier indicating whether resources are recovered at a warm or emergency restart. The values are: UEIDKEEP The resources are recoverable at a warm or emergency restart. UEIDLOSE The resources are not recoverable. Note: The exit is not driven during a CICS restart.
Return codes	UERCNORM Continue processing. This is the default. UERCPURG Task purged during XPI call.
XPI calls	You can use all XPI calls.

Important

Abends in a program enabled at the XRSINDI exit point may cause CICS to terminate, because for some resources the exit is driven during syncpoint. If the exit returns code UERCPURG during syncpoint for these resources, abend code AUPEP is produced and CICS terminates.

Shared data tables management program exits XDTRD, XDTAD, and XDTLC

XDTRD and XDTAD allow you to control the selection of records for inclusion in a data table, XDTRD being used to make such selections during loading, and XDTAD being invoked when records are subsequently added. XDTRD also allows the contents of records that are included in a user-maintained table to be modified before they are added. XDTLC enables you to take action based on the fact that a data table has completed loading, which might be to end some restrictions that you have decided to place on access to the data table during loading, or to cater for an unsuccessful completion of the loading.

Note that a program invoked from any of these exit points must declare a DSECT defining the data tables user exit parameter list pointed to by field UEPDTPL. (Although UEPDTPL is defined by a DFHUEXIT call, the parameter list that it addresses is not.) To do this, your program can include the copybook DFHXDTDS, which defines the DT_UE_PLIST DSECT.

+
+

If any tables specify FILSTAT=OPENED or are opened implicitly, you should provide a program list table post-initialization (PLTPI) program to activate the user exits. Otherwise, the data table may start loading before the exits can be enabled. For more details about PLTPI programs, see Chapter 4, “Writing initialization and shutdown programs” on page 335.

+
+

Note: For additional information about using these exits with CICS shared data table support, see the *CICS/ESA Shared Data Tables Guide*.

+
+
+

+
+
+

+
+
+
+

+
+
+
+
+
+

+
+
+
+
+
+

+
+
+
+
+
+

+
+
+
+
+
+

+
+
+

XDTRD

The XDTRD user exit is invoked just before CICS attempts to add to the data table a record that has been retrieved from the source data set.

This normally occurs when the loading process retrieves a record during the sequential copying of the source data set. However, it can also occur when an application retrieves a record that is not in the data table and:

- For a user-maintained data table, loading is still in progress, or
- For a CICS-maintained data table, loading terminated before the end of the source data set was reached (because, for example, the data table was full).

The record retrieved from the source data set is passed as a parameter to the user exit program—see fields UEPDTRA and UEPDTRL. Your exit program can choose (depending, for example, on the key value—see fields UEPDTKA and UEPDTKL) whether to include the record in the data table or not.

Alternatively, if you are using CICS shared data tables support, the exit program can request that all subsequent records up to a specified key are skipped—see field UEPDTSKA; these records are not passed to the exit program. This facility is available only during loading. You can specify the key as a complete key, or you can specify just the leading characters by padding the skip-key area with binary zeros.

For a user-maintained data table, the program can also modify the data in the record to reduce the storage needed for the data table. Application programs that use the data table must be aware of any changes made to the record format by the exit program. If the record length is changed, the exit program must set the new length in the parameter list—see field UEPDTRL. The new length must not exceed the data buffer length—see field UEPDTRBL.

<i>Table 51 (Page 1 of 3). Exit XDTRD</i>	
When invoked	Just before CICS tries to add to the data table a record that has been retrieved from the source data set.

+

<i>Table 51 (Page 3 of 3). Exit XDTRD</i>		
Return codes	UERC DTAC	Add the record to the data table.
	UERC DTRJ	Reject the record: that is, do not add it to the table.
	UERC DTOP	For shared data tables only, skip this and the following records until a key is found that is equal to or greater than the key specified in the skip-key area.
XPI calls	All can be used.	

+

+

+

XDTAD

- + The XDTAD user exit is invoked when a write request is issued to a data table.
- + • For a user-maintained data table, the user exit is invoked once—before the
- + record is added to the data table.
- + • For a CICS-maintained data table, the user exit is invoked twice—before the
- + record is added to the source data set and then again before the record is
- + added to the data table.

- + The record written by the application is passed as a parameter to the user exit
- + program—see fields UEPDTRA and UEPDTRL. Your exit program can choose
- + (depending on the key value, for example—see fields UEPDTKA and UEPDTKL)
- + whether to include the record in the data table or not. This decision is indicated by
- + setting the return code.

- + The XDTAD exit must not modify the data in the record. If you used XDTRD to
- + truncate the data records when the user-maintained data table was loaded, you
- + must code your application so that it only tries to write records of the correct format
- + for the data table.

- + A sample XDTAD exit program is listed in the *CICS/ESA Shared Data Tables*
- + *Guide*.

<i>Table 52 (Page 1 of 2). Exit XDTAD</i>	
When invoked	One or more times during the processing of a write request to a data table.

XDTLC

+ The XDTLC user exit is invoked at the completion of data table loading—whether
 + successful or not. **The user exit is not invoked if the data table is closed for**
 + **any reason before loading is complete.**

+ The exit program is informed if the loading did not complete successfully—see field
 + UEPDTORC. This could occur, for example, if the maximum number of records
 + was reached or there was insufficient virtual storage. In this case, the exit program
 + can request that the file is closed immediately, by setting the return code.

Table 53. Exit XDTLC

When invoked	At the completion of table loading. It is not invoked if the loading process was terminated because the data table had been closed.	
Exit-specific parameters	UEPDTPL	Address of the data table user exit parameter list, which contains:
	UEPDTNAM	The 8-character data table name.
	UEPDTFLG	A 1-byte flag field. The possible bit settings are:
	UEPDTSDT (X'80')	The exit has been invoked by CICS shared data table support.
		The remaining flag may be set only if UEPDTSDT is set—that is, only if shared data tables support is active:
	UEPDTCMT (X'40')	This is a CICS-maintained table.
	UEPDTORC	Data table open result code. The possible values are: X'00' Load successful X'80' Load unsuccessful.
UEPDTDSL	For shared data tables only, the fullword length of the name of the source data set.	
UEPDTDSN	For shared data tables only, a 44-character field containing the name of the source data set.	
Return codes	UERCDTOK	Accept the data table in its present state
	UERCDTCL	Close the data table.
XPI calls	All can be used.	

Signon and signoff exits XSNON and XSNOFF

Exit XSNON is invoked after a terminal user signs on, and exit XSNOFF is invoked after a terminal user signs off (whether the signon or sign-off is successful or not). XSNON and XSNOFF do not make any security decisions; they are merely a means of tracking users logging on and off a CICS system.

The activities which drive the exits are:

- Invocation of an EXEC CICS SIGNON command for a terminal (when, for example, the terminal user enters the CICS-supplied CESN, or an equivalent, user-written, signon transaction)
- Invocation of an EXEC CICS SIGNON command for a surrogate terminal (that is, a terminal attached by the CRTE routing transaction, or by dynamic transaction routing)
- Invocation of an EXEC CICS SIGNOFF command for a terminal
- When a 'CANCEL' command is entered to terminate a CRTE routing session
- A timeout sign-off.

XSNON

Table 54. Exit XSNON

When invoked	When a user signs on.	
Exit-specific parameters	UEPUSRID	Address of the terminal userid.
	UEPUSRLN	Address of the terminal userid length.
	UEPGRPID	Address of the group ID. If the signon was successful, the group ID is that which the user is associated with in this signon session. If the signon was unsuccessful, it is that specified by the user when he or she tried to sign on.
	UEPGRPLN	Address of the group ID length.
	UEPNETN	Address of the terminal's netname.
	UEPTRMID	Address of the terminal id.
	UEPTCTUA	Address of the TCT user area.
	UEPTCTUL	Address of the TCT user area length.
	UEPTRMTY	Address of the terminal-type byte.
	UEPSNFLG	Address of a 2-byte field containing flags:
	UEPSNOK	Signon was successful
	UEPSNFL	Signon failed
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

XSNOFF

<i>Table 55. Exit XSNOFF</i>		
When invoked	When a user signs off.	
Exit-specific parameters	UEPUSRID	Address of the terminal userid.
	UEPUSRLN	Address of the terminal userid length.
	UEPGRPID	Address of the group ID.
	UEPGRPLN	Address of the group ID length.
	UEPNETN	Address of the terminal's netname.
	UEPTRMID	Address of the terminal id.
	UEPTCTUA	Address of the TCT user area.
	UEPTCTUL	Address of the TCT user area length.
	UEPTRMTY	Address of the terminal-type byte.
	UEPSNFLG	Address of a 2-byte field containing flags:
	UEPSNOK	Sign-off was successful
	UEPSNFL	Sign-off failed
	UEPSNML	Normal sign-off
	UEPSNTIM	Timeout sign-off.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

Statistics domain exit XSTOUT

On invocation, XSTOUT is passed the address of a buffer containing one or more statistics records. The buffer can contain records for various resource types (for example, connections and modenames), and both specific and global information (for example, loader statistics for individual programs, and loader statistics for all programs).

Your exit program can identify the types of records in the buffer by their STID values. (STID values are described in “CICS statistics data section” on page 586.)

You can use XSTOUT to prevent the contents of the statistics data buffer being written to SMF. Note that you cannot use it to selectively suppress individual records within the buffer. Your exit program should not modify the values of any of the exit-specific parameters.

Table 56 (Page 1 of 2). Exit XSTOUT

When invoked	Before a statistics record is written to SMF.
---------------------	---

Table 56 (Page 2 of 2). Exit XSTOUT

Exit-specific parameters	<p>Fields UEPTRANID, UEPUSER, UEPTERM, and UEPPROG have meaning only for requested statistics (when using CEMT PERFORM STATISTICS RECORD or the EXEC CICS PERFORM STATISTICS RECORD command).</p>
	<p>UEPTRANID Address of the 4-byte transaction ID.</p>
	<p>UEPUSER Address of the 8-byte user ID.</p>
	<p>UEPTERM Address of the 4-byte terminal ID.</p>
	<p>UEPPROG Address of the 8-byte application program name.</p>
	<p>UEPSTATS Address of a buffer containing one or more statistics records. For unsolicited statistics, the buffer always contains one record; for other types of statistics, it may contain several records. The length of the buffer is addressed by the UEPSRLEN parameter.</p>
	<p>UEPSRLEN Address of the 4-byte hexadecimal length of the statistics record.</p>
	<p>UEPSTYPE Address of the 3-byte character field statistics type. The values of the types are:</p>
	<p>INT Interval statistics</p>
	<p>EOD End-of-day statistics</p>
	<p>REQ Requested statistics</p>
	<p>RRT Requested reset statistics</p>
	<p>USS Unsolicited statistics.</p>
	<p>UEPSDATE Address of a 6-byte character field containing the collection date (MMDDYY).</p>
	<p>UEPSTIME Address of a 6-byte character field containing the collection time (HHMMSS).</p>
	<p>UEPSIVAL Address of a 6-byte character field containing the interval time (HHMMSS). This field has meaning only for interval statistics.</p>
	<p>UEPSIVN Address of the 4-byte interval number. This field has meaning only for interval statistics.</p>
	<p>UEPSCLD</p> <p>APAR PN69653: Documentation for PN69653 added on 8 December 1995.</p>
	<p>Address of an 8-byte character field containing the collection date (MMDDYYYY).</p>
Return codes	<p>UERCNORM Continue processing.</p>
	<p>UERCBYP Suppress output of statistics data buffer to SMF.</p>
XPI calls	<p>WAIT_MVS can be used. Note, however, that the wait cannot be purged using CEMT or SPI. Do not use any other calls.</p>

|

+
+
+
+
+
+

+
+
+
+
+

|

System recovery program exit XSRAB

<i>Table 57 (Page 1 of 3). Exit XSRAB</i>	
When invoked	When the system recovery program (DFHSRP) finds a match in the SRT for an MVS/ESA abend code. For information about defining entries in the SRT, refer to the <i>CICS/ESA Resource Definition Guide</i> .
Exit-specific parameters	<p>UEPERROR Address of the error data structure, SRP_ERROR_DATA, which contains the following fields:</p> <p>SRP_ERROR_TYPE The 4-character error type—always 'ASRB'.</p> <p>SRP_SYS_ABCODE 2 bytes containing the system abend code XXX in binary format (for example, D37).</p> <p>SRP_USER_ABCODE 2 bytes containing the user abend code NNNN in binary format (for example, 0999).</p> <p>SRP_ERROR_TRANID 4-character field containing the ID of the abending transaction.</p> <p>SRP_ERROR_STACK_NAME 8-character field containing the name of the current kernel stack entry for the transaction at the time of the abend.</p> <p>SRP_ERROR_PPT_NAME 8-character field containing the name of the current PPT entry for the transaction, if one exists. This field contains a value only if flag SRP_PPT_ENTRY is set.</p> <p>SRP_ERROR_OFFSET Fullword containing the offset into the program that abended, as follows:</p> <ul style="list-style-type: none"> • If flag SRP_PPT_ENTRY is set, gives the offset in SRP_ERROR_PPT_NAME • Otherwise, gives the offset in SRP_ERROR_STACK_NAME. <p>This field contains a value only if flag SRP_VALID_OFFSET is set.</p> <p>SRP_ERROR_FLAGS 1 byte containing flags:</p> <p>SRP_CICS_CODE The abend occurred while running CICS code.</p> <p>SRP_USER_CODE The abend occurred while running user application code.</p>

Table 57 (Page 2 of 3). Exit XSRAB

<p>SRP_PPT_ENTRY The abend occurred while running SRP_ERROR_PPT_NAME. If this flag is not set, the abend occurred while running SRP_ERROR_STACK_NAME.</p> <p>SRP_VALID_OFFSET A meaningful offset could be determined.</p> <p>SRP_VALID_REASON MVS has supplied a reason code for the abend.</p> <p>SRP_NOT_CICS_RB CICS RB was not in control at the time of the abend (that is, the abend occurred in a system service invoked by CICS).</p> <p>SRP_CICS_ERROR_REASON 4-character field containing the MVS abend reason code. It contains a value only if flag SRP_VALID_REASON is set.</p> <p>SRP_CICS_ERROR_DATA An area describing the last thing that CICS did, prior to the abend. It contains the following:</p> <p>SRP_CICS_EC_PSW 8-character field containing the extended control (EC) mode program status word (PSW)</p> <p>SRP_CICS_EC_INT 8-character field containing the interrupt code and ILC</p> <p>SRP_CICS_REGST 64-character field containing the contents of the general-purpose (GP) registers</p> <p>SRP_CICS_EXEC_KEY 1 byte containing the PSW key, in the form X'0n'.</p> <p>SRP_SYSTEM_ERROR_DATA An area describing the last thing "the system" did, prior to the abend. It contains the following:</p> <p>SRP_SYSTEM_EC_PSW 8-character field containing the EC mode PSW</p> <p>SRP_SYSTEM_EC_INT 8-character field containing the interrupt code and ILC</p> <p>SRP_SYSTEM_REGST 64-character field containing the contents of the GP registers</p> <p>SRP_SYSTEM_EXEC_KEY 1 byte containing the PSW key, in the form X'0n'.</p>
--

Table 57 (Page 3 of 3). Exit XSRAB

<p>SRP_ERROR_FP_REGS An area describing the contents of the floating point registers at the time of the abend. It contains:</p> <p>SRP_FP_REG_0 FP register 0</p> <p>SRP_FP_REG_2 FP register 2</p> <p>SRP_FP_REG_4 FP register 4</p> <p>SRP_FP_REG_6 FP register 6.</p> <p>Notes:</p> <ol style="list-style-type: none"> 1. If flag SRP_NOT_CICS_RB is set, SRP_CICS_ERROR_DATA describes the last thing that CICS did, prior to the abend; SRP_SYSTEM_ERROR_DATA describes the last thing that the system service (for example, VTAM, VSAM, or MVS) did. 2. The format of SRP_ERROR_DATA is shown in the <i>CICS/ESA Data Areas</i> manual. 	
Return codes	<p>UERCNOCA Abnormally terminate the task with abend code 'ASRB'. Do not cancel any program-level abend exits that are associated with this task.</p> <p>UERCNCANC Abnormally terminate the task with abend code 'ASRB'. Cancel any program-level abend exits that are associated with this task.</p> <p>UERCNCICS Abnormally terminate CICS.</p>
XPI calls	<p>Because CICS invokes the exit XSRAB in an error environment, you can only use a subset of the XPI calls.</p> <p>Only TRACE_PUT is available for general use.</p> <p>You can use WAIT_MVS, but only after the exit program has determined (from the SRP_CICS_CODE and SRP_USER_CODE fields) that the abend has occurred in user application code, and not in CICS code.</p>

Important

Notes:

1. Take care when coding a program to run at the XSRAB exit point. If your exit program causes the system recovery program to be reentered (if, for example, a program check occurs) then CICS terminates abnormally, with a DFHSR06xx message.
2. The default return code is 'UERCNOCA'. This ensures that the task abends if the exit is in error.
3. There is no 'UERCNORM' return code at this exit point, because the exit is invoked after a failure.
4. The exit should **not** set the return code 'UERCPURG'.

System termination program exit XSTERM

The XSTERM exit could be used to output final statistics to your statistics SMF data sets, and to close them. (Note that CICS VSAM and BDAM data sets have already been closed by CICS file control before the exit is invoked.)

When invoked	During the second quiesce stage of a normal system shutdown, immediately before the transient data and temporary storage buffers are cleared. The exit is not invoked during an IMMEDIATE shutdown.
Exit-specific parameters	None.
Return codes	UERCNORM Continue processing.
XPI calls	All XPI calls except WRITE_JOURNAL_DATA can be used. However, their use is not recommended, because they could cause the task to lose control, thus allowing another task to write more monitoring data.

Task control program exit XKCREQ

Table 59. Exit XKCREQ

When invoked	Before ENQUEUE and DEQUEUE requests are actioned.	
Exit-specific parameters	UEPENQFN	A 1-byte field indicating the type of function—either ENQUEUE or DEQUEUE—which caused the exit program to be invoked. The equated values are: UEPENQ ENQUEUE UEPDEQ DEQUEUE.
	UEPENQA	The address of the resource name that is the object of the ENQUEUE or DEQUEUE request.
	UEPENQL	The length of the resource name.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

Note: Some requests that caused this exit to be driven in previous releases of CICS are passed directly to the dispatcher domain in CICS/ESA 4.1, and therefore do not cause the exit to be driven.

Temporary storage control program exits XTSREQ, XTSIN, and XTSOUT

XTSREQ

<i>Table 60. Exit XTSREQ</i>		
When invoked		Before analysis of a temporary storage request.
Exit-specific parameters	UEPTSIOA	Address of the TSIOAVRL field in the temporary storage input/output area (TSIOA). The TSIOA can be mapped using the DSECT DFHTSIOA. Note that UEPTSIOA does not point to the start of the DFHTSIOA DSECT.
	UEPTSDI	Name of the temporary storage queue ID.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls		All can be used.

XTSIN

<i>Table 61. Exit XTSIN</i>		
When invoked		After an input event.
Exit-specific parameters	UEPTSIOA	Address of the TSIOAVRL field in the temporary storage input/output area (TSIOA). The TSIOA can be mapped using the DSECT DFHTSIOA. Note that UEPTSIOA does not point to the start of the DFHTSIOA DSECT.
	UEPTSGID	Address of the first temporary storage group identification control block (TSGID). The TSGID is mapped by the DSECT DFHTSGID. However, field TSGIDSA is not a programming interface.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls		All can be used.

XTSOUT

<i>Table 62. Exit XTSOUT</i>		
When invoked	Before an output event.	
Exit-specific parameters	UEPTSIOA	Address of the TSIOAVRL field in the temporary storage input/output area (TSIOA). The TSIOA can be mapped using the DSECT DFHTSIOA. Note that UEPTSIOA does not point to the start of the DFHTSIOA DSECT.
	UEPTSGID	Address of the first temporary storage group identification control block (TSGID). The TSGID is mapped by the DSECT DFHTSGID. However, field TSGIDSA is not a programming interface.
	UEPTSREC	Address of the record prefix. The record prefix is mapped by the DSECT DFHTSCI. However, field TSCISA is not a programming interface.
Return codes	UERCNORM	Continue processing.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

Temporary storage EXEC interface program exits XTSEREQ and XTSEREQC

The XTSEREQ exit allows you to intercept temporary storage API requests before any action has been taken on the request. The XTSEREQC exit allows you to intercept the response after a temporary storage API request has completed.

The API requests affected are:

- EXEC CICS WRITEQ TS
- EXEC CICS READQ TS
- EXEC CICS DELETEQ TS

Using **XTSEREQ**, you can:

- Analyze the API parameter list (function, keywords, argument values, and responses)
- Modify any input parameter value prior to execution of a request
- Prevent execution of a request.

Using **XTSEREQC**, you can:

- Analyze the API parameter list
- Modify any output parameter value after request completion.

You can also:

- Pass data between your XTSEREQ and XTSEREQC exit programs when they are invoked for the same request
- Pass data between your temporary storage exit programs when they are invoked within the same task.

It is possible that programs invoked from the exits in the temporary storage control program (XTSOUT, XTSIN, and XTSREQ) could modify situations set up by XTSEREQ; therefore you must consider the order in which the exits are invoked.

If all five exits are enabled, the order of invocation is as follows:

- For the WRITEQ temporary storage command:
 1. XTSEREQ
 2. XTSREQ
 3. XTSOUT
 4. XTSEREQC
- For the READQ temporary storage command:
 1. XTSEREQ
 2. XTSREQ
 3. XTSIN
 4. XTSEREQC
- For the DELETEQ temporary storage command:
 1. XTSEREQ
 2. XTSREQ
 3. XTSEREQC

XTSEREQ

<i>Table 63. Exit XTSEREQ</i>		
When invoked	Before CICS processes a temporary storage API request.	
Exit-specific parameters	UEPCLPS	Address of a copy of the command parameter list. See "The command-level parameter structure" on page 145.
	UEPTQTOK	Address of a 4-byte area which can be used to pass information between XTSEREQ and XTSEREQC for a single temporary storage request.
	UEPRCODE	Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see the <i>CICS/ESA Application Programming Reference</i> manual.
	UEPRES P	Address of a 4-byte binary copy of the EIB response code EIBRESP.
	UEPRES P2	Address of a 4-byte binary copy of the EIB response code EIBRESP2.
	UEPTSTOK	Address of a 4-byte token which can be used to pass information between successive temporary storage requests within the same task (for example, between successive invocations of the XTSEREQ exit).
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERC B Y P	Bypass this request
	UERC N O R M	Continue processing.
	UERC P U R G	Task purged during XPI call.
XPI calls	All can be used.	
API commands	All can be used.	

Warning: Care should be taken when issuing recursive commands. For example, you must avoid entering a loop when issuing a temporary storage request from the XTSEREQ exit. Use of the recursion counter UEPRECUR is recommended.

XTSEREQC

<i>Table 64. Exit XTSEREQC</i>	
When invoked	After a temporary storage API request has completed, before return from the temporary storage EXEC interface program.
Exit-specific parameters	<p>UEPCLPS Address of a copy of the command parameter list. See “The command-level parameter structure” on page 145.</p> <p>UEPTQTOK Address of a 4-byte area which can be used to pass information between XTSEREQ and XTSEREQC for a single temporary storage request.</p> <p>UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code EIBRCODE. For details of EIB return codes, see the <i>CICS/ESA Application Programming Reference</i> manual.</p> <p>UEPRES P Address of a 4-byte binary copy of the EIB response code EIBRESP.</p> <p>UEPRES P2 Address of a 4-byte binary copy of the EIB response code EIBRESP2.</p> <p>UEPTSTOK Address of a 4-byte token which can be used to pass information between successive temporary storage requests within the same task (for example, between successive invocations of the XTSEREQC exit).</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	All can be used.
API commands	All can be used.

Warning: Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when issuing a temporary storage request from the XTSEREQC exit. Use of the recursion counter UEPRECUR is recommended.

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. If you update the values, temporary storage copies the new values into the application program’s EIB after the completion of XTSEREQC or if you specify a return code of UERCBYP in XTSEREQ.

You must set valid temporary storage responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by temporary storage to describe a valid completion. CICS does not check the consistency of EIBRCODE, EIBRESP, and EIBRESP2. If EIBRCODE is set to a non-zero value and EIBRESP is set to zero, CICS will override EIBRESP with a non-zero value. To help you set values for EIBRCODE, EIBRESP, and EIBRESP2, the values used by temporary storage are specified in DSECT DFHTSUED.

The command-level parameter structure

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of a bit string that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request.

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request.

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

The UEPCLPS exit-specific parameter: The UEPCLPS exit-specific parameter is included in both exit XTSEREQ and exit XTSEREQC. It is the address of the command-level parameter structure. The command-level parameter structure contains 8 addresses, TS_ADDR0 through TS_ADDR7. It is defined in the DSECT TS_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHTSUED.

The command-level parameter list is made up as follows.

Note: The relationship between arguments, keywords, data types, and input/output types is summarized for the temporary storage commands in the following tables:

Command	See
WRITEQ TS	Table 65 on page 149
READQ TS	Table 66 on page 149
DELETEQ TS	Table 67 on page 150

TS_ADDR0

is the address of an 8-byte area called the EID, which is made up as follows:

TS_GROUP
TS_FUNCT
TS_BITS1
TS_BITS2
TS_EIDOPT5
TS_EIDOPT6
TS_EIDOPT7

TS_GROUP Always X'0A', indicating that this is a temporary storage request.

TS_FUNCT One byte that defines the type of request:

X'02' WRITEQ
X'04' READQ

temporary storage EXEC interface program exits

	X'06'	DELETEQ
TS_BITS1		Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.
	X'80'	Set if the request contains an argument for the QUEUE keyword. If set, TS_ADDR1 is meaningful.
	X'40'	Set if the request contains an argument for any of the FROM, INTO, or SET keywords. If set, TS_ADDR2 is meaningful.
	X'20'	Set if the request contains an argument for the LENGTH keyword. If set, TS_ADDR3 is meaningful.
	X'10'	Set if the request contains an argument for the NUMITEMS keyword. If set, TS_ADDR4 is meaningful.
	X'08'	Set if the request contains an argument for the NUMITEMS or ITEM keyword. If set, TS_ADDR5 is meaningful.
	X'02'	Set if the request contains an argument for the SYSID keyword. If set, TS_ADDR7 is meaningful.
TS_BITS2		Two bytes not used by temporary storage.
TS_EIDOPT5		Indicates whether certain keywords were specified on the request.
	X'01'	SET was specified (otherwise INTO). You cannot modify this bit from your user exit.
TS_EIDOPT6		One byte not used by temporary storage.
TS_EIDOPT7		Indicates whether certain functions and/or keywords were specified on the request.
	X'10'	WRITEQ NOSUSPEND specified.
	X'80'	WRITEQ MAIN or READQ ITEM specified.
	X'04'	WRITEQ REWRITE or READQ NUMITEMS specified.
TS_EIDOPT8		Indicates whether certain keywords were specified on the request.
	X'80'	ITEM was specified (otherwise NUMITEMS).

TS_ADDR1

is the address of an 8-byte area containing the name from QUEUE.

TS_ADDR2

is the address of one of the following:

- A 4-byte address from SET (if the request is READQ and **TS_EIDOPT5** indicates that this is SET).
- Data from INTO (if the request is READQ and **TS_EIDOPT5** indicates that this is not SET).
- Data from FROM (if the request is WRITEQ).

TS_ADDR3

is the address of the halfword value of LENGTH (if the request is READQ or WRITEQ).

Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

TS_ADDR4

is the address of the halfword value of NUMITEMS (if the request is READQ).

TS_ADDR5

is the address of one of the following:

- The halfword value of NUMITEMS (if the request is WRITEQ).
- The halfword value of ITEM (if the request is READQ or WRITEQ).

TS_ADDR6

is the address of a value intended for CICS internal use only. It must not be used.

TS_ADDR7

is the address of an area containing the value of SYSID.

Modifying fields in the command-level parameter structure: Some fields that are passed to temporary storage are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

QUEUE
FROM
SYSID

The following are always output fields:

INTO
NUMITEMS
SET

LENGTH is an input field on a WRITEQ request, and an output field on a READQ request that specifies SET. It is both an input and an output field on a READQ request that specifies INTO.

temporary storage EXEC interface program exits

ITEM is an input field on a READQ request, and on a WRITEQ request that specifies REWRITE. It is both an input and an output field on a WRITEQ request that does not specify REWRITE.

Modifying input fields: The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Warning: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in “Modifying input fields” is not suitable for modifying output fields. (The results would be returned to the new area instead of the application’s area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application’s data in place, because the application is expecting the field to be modified anyway.

Modifying fields used for both input and output: An example of a field that is used for both input and output is LENGTH on a READQ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID: It is not possible to modify the EID to make major changes to requests. It is not possible, for example, to change a READQ request to a WRITEQ request.

However, you can make minor changes to requests, such as to turn on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that can be modified. Any attempt to modify any other part of the EID is ignored.

TS_BITS1

X'02' The existence bit for SYSID.

TS_EIDOPT7

A user exit program at XTHEREQ can set the following on or off for all WRITEQ TS commands:

X'10' The existence bit for NOSUSPEND.

X'08' The existence bit for MAIN.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the temporary storage request only.

Warning: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that are permitted.

Use of the task token UEPTSTOK: UEPTSTOK provides the address of a 4-byte area that you can use to pass information between successive temporary storage requests in the same task. (By contrast, UEPTQTOK is usable only for the duration of a single temporary storage request, because its contents may be destroyed at the end of the request.) For example, if you need to pass information between successive invocations of the XTSEREQ exit, UEPTSTOK provides a means of doing this.

Table 65. WRITEQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg2	FROM	DATA-AREA	input
Arg3	LENGTH	BIN(15)	input
Arg4	*	*	*
Arg5	ITEM	BIN(15)	input/output
Arg5	NUMITEMS	BIN(15)	output
Arg6	*	*	*
Arg7	SYSID	CHAR(4)	input

Note: The different uses of Arg5 are shown, because Arg5 is used by the ITEM and NUMITEMS keywords which are alternatives and the argument to the ITEM keyword is an input field when REWRITE is specified.

Table 66. READQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg2	SET	DATA-AREA, PTR	output
Arg2	INTO	DATA-AREA	output
Arg3	LENGTH	BIN(15)	input/output
Arg4	NUMITEMS	BIN(15)	output
Arg5	ITEM	BIN(15)	input
Arg6	*	*	
Arg7	SYSID	CHAR(4)	input

Table 67. DELETEQ TS: User arguments and associated keywords, data types, and input/output types

Argument	Keyword	Data type	Input/output type
Arg1	QUEUE	CHAR(8)	input
Arg2	*	*	*
Arg3	*	*	*
Arg4	*	*	*
Arg5	*	*	*
Arg6	*	*	*
Arg7	SYSID	CHAR(4)	input

Modifying user arguments: User exit programs can modify user arguments, as follows:

For input arguments, the user exit program should obtain sufficient storage to hold the modified argument, set up that storage to the required value, and set the associated pointer in the parameter list to the address of the newly acquired area.

For output arguments, and for input/output arguments, the user exit program can update the argument in place, because the area of storage is represented by a variable in the application which is expected to receive a value from CICS.

Notes:

1. CICS does not check changes to argument values, so any changes must be verified by the user exit program making the changes.
2. It is not advisable for XTSEREQ to modify output arguments or for XTSEREQC to modify input arguments.

Adding user arguments: Global user exit programs can add arguments associated with the SYSID keyword. You must ensure that the arguments you specify or modify in your exit programs are valid.

Assuming that the argument to be added does not already exist, the user exit program must:

1. Obtain storage for the argument to be added
2. Initialize the storage to the required value
3. Select and set up the appropriate pointer from the parameter list
4. Select and set up the appropriate argument existence bit in the EID
5. Modify the parameter list to reflect the new end of list indicator.

Removing user arguments: User exit programs can remove arguments (for which the program is totally responsible) associated with the SYSID keyword:

Assuming that the argument to be removed exists, the user exit program must:

1. Switch the corresponding argument existence bit to '0'b in the EID
2. Modify the parameter list to reflect the new end of list indicator.

Example program

CICS supplies—in hardcopy only—an example program, DFH\$XTSE, that shows how temporary storage requests can be modified. See Appendix E, “The example program for the XTSEREQ global user exit, DFH\$XTSE” on page 717.

Terminal allocation program exit XALCAID

XALCAID is driven when an AID with data is canceled in one of the following ways:

- By means of the CEMT transaction
- During execution of a SET TERMINAL or SET CONNECTION command
- During reinstallation of a terminal or connection.

XALCAID is invoked only if there is data associated with the AID.

Table 68 (Page 1 of 2). Exit XALCAID

When invoked	Whenever an AID with data is canceled.	
	Note:	It is not possible for the exit to prevent the request from being canceled.
Exit-specific parameters	UEPALTSD	Address of a 4-byte field containing the symbolic identifier of the transaction which was to be started by this request.
	UEPALTRM	Address of a 4-byte field containing the identifier of the terminal or connection to which this request was directed.
	UEPALDAT	Address of an area of storage containing the data specified in the FROM option; or hexadecimal zeros, if the AID was created by a START request without a FROM option.
	UEPALLEN	Address of a fullword binary field containing the length of the FROM data; or hexadecimal zeros, if the FROM option was not specified.
	UEPALRQD	Address of an 8-byte field containing the value of the REQID associated with the FROM data. The data was stored in a temporary storage queue with this name. This value was either specified explicitly using the REQID option on the START command, or created internally by CICS.
	UEPALQUE	Address of an 8-byte field containing the value specified in the QUEUE option on the START command, or hexadecimal zeros if QUEUE was not specified.
	UEPALRTE	Address of a 4-byte field containing the value specified in the RTERMID option on the START command, or hexadecimal zeros if RTERMID was not specified.
	UEPALRTA	Address of a 4-byte field containing the value specified in the RTRANSID option on the START command, or hexadecimal zeros if RTRANSID was not specified.
	UEPALFMH	Address of a 1-byte field containing the value X'FF' if the data contains FMHs, as specified by the FMH option on the associated START command; or hexadecimal zeros otherwise.
	UEPALSTC	Address of a 2-byte field containing the start code. This is 'SZ' for FEPI starts; otherwise it is 'SD'.
Return codes	UERCNORM	No other return codes are supplied. The value of the return code is not inspected.

Table 68 (Page 2 of 2). Exit XALCAID

XPI calls	You can use: INQ_APPLICATION_DATA INQUIRE_SYSTEM No other XPI calls should be used.
API and SPI commands	No EXEC CICS commands can be used.

Note: The XALTENF exit, used to handle the “terminal not known” condition, is also invoked from the terminal allocation program. XALTENF is described on page 157.

Terminal control program exits XTCIN, XTCOUT, XTCATT, XTCTIN, and XTCTOUT

XTCIN

<i>Table 69. Exit XTCIN</i>		
When invoked	After an input event for a sequential device.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTCTLE	Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. However, note that you cannot use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA.	

XTCOUT

<i>Table 70. Exit XTCOUT</i>		
When invoked	Before an output event for a sequential device.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTCTLE	Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. However, note that you cannot use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA.	

XTCATT

<i>Table 71. Exit XTCATT</i>		
When invoked	Before task attach.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTCTLE	Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.
	UEPTRAN	Address of the 4-byte transaction id.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used.	

XTCTIN

<i>Table 72. Exit XTCTIN</i>		
When invoked	After a TCAM input event.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTCTLE	Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.
Return codes	UERCNORM	Continue processing – format the TCAM header.
	UERCBYP	Suppress formatting of the TCAM header.
XPI calls	All can be used.	

XTCTOUT

<i>Table 73. Exit XTCTOUT</i>		
When invoked		Before a TCAM output event.
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTCTLE	Address of the terminal control table line entry (TCTLE). The TCTLE can be mapped using the DSECT DFHTCTLE.
Return codes	UERCNORM	Continue processing – format the TCAM header.
	UERCBYB	Suppress formatting of the TCAM header.
XPI calls		All can be used.

'Terminal not known' condition exits XALTENF and XICTENF

The 'terminal not known' condition can occur when intercommunicating CICS regions use both SHIPPABLE terminal definitions and automatic transaction initiation (ATI). The condition is especially likely to arise if autoinstall is used.

SHIPPABLE attribute

Terminals defined with the SHIPPABLE attribute in a terminal-owning region (TOR) do not need a definition in a connected application-owning region (AOR). If necessary to support transaction routing, CICS ships a copy of the definition from the TOR to the AOR. For further information, refer to the *CICS/ESA Resource Definition Guide*.

Automatic transaction initiation (ATI)

ATI occurs when an internally generated request leads to the initiation of a transaction. For example, when:

- An application issues an EXEC CICS START command, or
- The transient data trigger level is reached.

Two CICS modules handle ATI requests:

The **interval control program** processes a START command, checks that the terminal is known in the local system, and (when any START time interval elapses) calls the terminal allocation program.

The **terminal allocation program** is called by the interval control program or by the transient data triggering mechanism, and checks that the terminal is known in the local system. If the requested terminal is remote, the terminal allocation program ships an ATI request to the remote system, which initiates transaction routing back to the local system.

For guidance information about ATI, refer to the *CICS/ESA Intercommunication Guide*.

'Terminal not known' condition

The 'terminal not known' condition arises when an ATI request is made for a terminal not known in the region. An ATI request can occur in the AOR for a SHIPPABLE terminal before any transaction routing has taken place for the terminal, and so before the definition of the terminal can have been shipped from the TOR to the AOR.

If the 'terminal not known' condition occurs, both the interval control program and the terminal allocation program reject the transaction-initiation request as 'TERMIDERR'.

The exits: To deal with the 'terminal not known' condition, CICS provides global user exits in the interval control and terminal allocation programs:

XICTENF In the interval control program

XALTENF In the terminal allocation program.

CICS drives the XICTENF exit when the 'terminal not known' condition occurs after the interval control program has been invoked by an EXEC CICS START command. CICS drives the XALTENF exit when the 'terminal not known' condition occurs after the terminal allocation program has been invoked by the transient data trigger level or the interval control program. Note that an EXEC CICS START command could result in both exits being invoked.

'terminal not known' condition exits

The exit program must indicate whether the terminal exists on another system and, if so, on which one. CICS passes data to the exit program to help establish this information. You can use the same exit program at both exit points. CICS supplies a sample exit program, DFHXTENF (see Figure 2 on page 163), that can be used at both exits and that can deal unchanged with some typical situations.

The exits are designed to deal with 'terminal not known' conditions that occur in CICS regions other than the TOR. For a TOR/AOR pair, enable the exit program in the AOR. The exits cannot deal with a 'terminal not known' condition in the TOR and the exit program should not normally be enabled there. However, if more than one TOR exists, you may need to enable the exit program in each TOR to deal with requests for terminals owned by other TORs. In this case, the exit program must recognize terminals that should be owned by this system and reject the requests ('UERCTEUN'). Although the exit provides as much data as possible, the logic of your program depends entirely on your system design. A simple solution to the most complex case would be to make the name of each terminal reflect the netname or sysid of its owning region.

Data returned by exit: The exit program must set a return code in register 15 as follows:

UERCTEUN Terminal does not exist
UERCNETN Netname returned
UERCYSI Sysid returned.

For return codes 4 and 8, the program must place the netname or sysid in fields UEPxxNTO or UEPxxSYO (where xx is AL or IC).

XALTENF

<i>Table 74 (Page 1 of 2). Exit XALTENF</i>	
When invoked	By the terminal allocation program when the terminal that an ATI request from transient data or interval control requires is unknown in this system. The exit program is expected to give a return code indicating whether the terminal exists on another connected CICS system and, if so, on which one.
Exit-specific parameters	<p>UEPALEVT Address of 2 bytes containing the type of request. The equated values of the types are:</p> <p>UEPALESD START command with data</p> <p>UEPALES START command without data</p> <p>UEPALETD Transient data trigger level reached.</p> <p>UEPALTR Address of 1 byte containing an indication of whether the task issuing the START command was started by transaction routing. The equated values are:</p> <p>UEPALTY A START command was being processed and the task issuing the command was transaction routed to.</p> <p>UEPALTN A START command was not being processed or a START command was being processed but the task issuing the command was not transaction routed to.</p> <p>UEPALFS Address of 1 byte containing an indication of whether the START command was function shipped. The equated values are:</p> <p>UEPALFY A START command was being processed and the START was function shipped.</p> <p>UEPALFN A START command was not being processed or a START was being processed but it was not function shipped.</p> <p>UEPALTRN Address of 4 bytes containing the name of the transaction to be run.</p> <p>UEPALRTR Address of 4 bytes containing the name of the terminal on which the transaction should run. (If a transient data trigger level was reached and the DCT entry specified DESTFAC= (SYSTEM,sysidnt), then this would contain a sysid).</p>

Table 74 (Page 2 of 2). Exit XALTENF		
	UEPALCTR	Address of 4 bytes containing, for START commands, the name of the current terminal if the command was transaction routed, or the name of the session if the command was function shipped. For other START commands and for transient data trigger events, the field pointed to contains blanks.
	UEPALNTI	Address of 8 bytes containing, for function-shipped START commands, the netname of the last system from which the request came. For START commands issued in this system by transaction routing to a task, the netname of the last system from which the task was routed. For other START command situations and for transient data trigger level events, the field pointed to contains blanks.
	UEPALSUI	Address of 4 bytes containing, if UEPALNTI contains a netname, the corresponding sysid. If UEPALNTI does not contain a netname, the field pointed to is blank.
	UEPALNTO	Address of 8 bytes containing the contents of UEPALNTI. If it sets a return code of 'UERCNETN', your exit program must place in this field the netname of the system to which the ATI request should be sent.
	UEPALSUI	Address of 4 bytes containing the contents of UEPALSUI. If it sets a return code of 'UERCSYSI', your exit program must place in this field the sysid of the system to which the ATI request should be sent.
Return codes	UERCTEUN	Terminal unknown, reject request
	UERCNETN	Terminal known, netname returned in UEPALNTO
	UERCSYSI	Terminal known, sysid returned in UEPALSUI.
XPI calls	You can use: INQ_APPLICATION_DATA INQUIRE_SYSTEM No other XPI calls should be used.	

XICTENF

<i>Table 75 (Page 1 of 2). Exit XICTENF</i>	
When invoked	By the interval control program when the terminal that an EXEC CICS START command requires is unknown in this system. The exit program is expected to give a return code indicating whether the terminal exists on another connected CICS system and, if so, on which one.
Exit-specific parameters	<p>UEPICEVT Address of 2 bytes containing the type of request. The equated values of the types are:</p> <p>UEPICESD START command with data</p> <p>UEPICES START command without data.</p> <p>UEPICTR Address of 1 byte containing an indication of whether the task issuing the START command was started by transaction routing. The equated values are:</p> <p>UEPICTY A START command was being processed and the task issuing the command was transaction routed to.</p> <p>UEPICTN A START command was not being processed or a START command was being processed but the task issuing the command was not transaction routed to.</p> <p>UEPICFS Address of 1 byte containing an indication of whether the START command was function shipped. The equated values are:</p> <p>UEPICFY A START command was being processed and the START was function shipped.</p> <p>UEPICFN A START command was not being processed or a START was being processed but it was not function shipped.</p> <p>UEPICTRN Address of 4 bytes containing the name of the transaction to be run.</p> <p>UEPICRTR Address of 4 bytes containing the name of the terminal on which the transaction should run.</p> <p>UEPICCTR Address of 4 bytes containing, for START commands, the name of the current terminal if the command was transaction routed, or the name of the session if the command was function shipped.</p> <p>For other START commands, the field pointed to contains blanks.</p>

<i>Table 75 (Page 2 of 2). Exit XICTENF</i>	
UEPICNTI	<p>Address of 8 bytes containing, for function-shipped START commands, the netname of the last system from which the request came.</p> <p>For START commands issued in this system by transaction routing to a task, the netname of the last system from which the task was routed.</p> <p>For other START command situations, the field pointed to contains blanks.</p>
UEPICSYI	<p>Address of 4 bytes containing, if UEPICNTI contains a netname, the corresponding SYSID.</p> <p>If UEPICNTI does not contain a netname, the field pointed to is blank.</p>
UEPICNTO	<p>Address of 8 bytes containing the contents of UEPICNTI.</p> <p>If it sets a return code of 'UERCNETN', your exit program must place in this field the netname of the system to which the ATI request should be sent.</p>
UEPICSYO	<p>Address of 4 bytes containing the contents of UEPICSYI.</p> <p>If it sets a return code of 'UERCYSI', your exit program must place in this field the sysid of the system to which the ATI request should be sent.</p>
Return codes	<p>UERCTEUN Terminal unknown, reject request.</p> <p>UERCNETN Terminal known, netname returned in UEPICNTO.</p> <p>UERCYSI Terminal known, sysid returned in UEPICSYO.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>The following must not be used:</p> <p>ADD_SUSPEND DELETE_SUSPEND RESUME SUSPEND WAIT_MVS.</p>

The sample program for the XALTENF and XICTENF exits, DFHXTENF

One program can be used for both exits, or a separate program can be written for each. Figure 2 shows the executable code from the supplied sample program DFHXTENF, which can be used for both exits. DFHXTENF rejects transient data requests, because the action in this case is very much installation-dependent.

Important

The example is intended purely as a demonstration of some of the possibilities available, and would be impractical in a production environment.

```
DFHXTENF CSECT
          DFHVM XTENF
          ENTRY DFHXTENA
DFHXTENA DS    0H
          STM   R14,R12,12(R13)    save registers
          BALR  R11,0              set up base register
          USING *,R11
*
          USING DFHUEPAR,R1        DFHUEH parameter list
*
*      Could check the terminal ID at this point. In this
*      program we assume it is valid. We also choose to accept
*      START requests and reject Transient Data trigger level
*      events.
*
          L     R2,UEPICEVT         access type of request
          CLC   0(2,R2),START       START command?
          BE    STARTCMD           yes
*
          CLC   0(2,R2),STARTDAT    START command with data?
          BNE   NOTSTART           no, must be Transient Data
*
STARTCMD DS    0H
*
*      Accept the default netname if we are Function Shipping.
*      Otherwise build a netname.
*
          L     R2,UEPICFS         access FS information
          CLI   0(R2),UEPICFY      Function Shipping?
          BNE   BLDNETNM          no, build a netname
*
          LH   R15,NETNAME        accept the default netname
          B    EXIT
*
```

Figure 2 (Part 1 of 2). Sample program for XALTENF and XICTENF exits

'terminal not known' condition exits

```
BLDNETNM DS    0H
*
*      Build a netname by taking the first character of the
*      terminal ID and appending it to the characters 'CICS'.
*
      L    R2,UEPICNTO      access the output netname field
      L    R3,UEPICRTR      access ID of requested terminal
      MVC  0(8,R2),=C'CICS  '
      MVC  4(1,R2),0(R3)    first character of terminal ID
      LH   R15,NETNAME      netname returned
      B    EXIT
*
NOTSTART DS    0H
      LH   R15,UNKNOWN      reject Transient Data trigger      *
                          level events
*
EXIT      DS    0H
      L    R14,12(R13)      restore registers except 15
      LM   R0,R12,20(R13)   which contains the return code
      BR   R14
*
*****
*      Local constants
*****
START     DC    AL2(UEPICES)
STARTDAT  DC    AL2(UEPICESD)
NETNAME   DC    AL2(UERCNETN)
UNKNOWN   DC    AL2(UERCTEUN)
*
      DFHEND DFHXTENF
```

Figure 2 (Part 2 of 2). Sample program for XALTENF and XICTENF exits

Transaction backout programs exits XRCINIT, XRCINPT, XRCOPER, and XRCFCER

At emergency restart, updates made to recoverable CICS resources that were not committed when the system failed must be backed out. Six programs are involved in the backout process, and they run in parallel under their own CICS tasks. There are global user exit points in four of the backout programs. These backout programs are:

- The DL/I backout program
- The file control backout program
- The message and ISC state recovery program
- The user backout program, for backing out user-written system log entries.

Four global user exits can be invoked from these backout programs. These are XRCINIT, XRCINPT, XRCOPER, and XRCFCER. XRCINIT is invoked at both initialization and termination of each of the four recovery programs. XRCINPT is invoked from the file control backout program, the user backout program, and the message and ISC state recovery program. XRCOPER and XRCFCER are invoked from the file control backout program only.

For further guidance information about exits in the transaction backout programs, refer to the *CICS/ESA Recovery and Restart Guide*.

Coding the exit programs

CICS services can be used in exit programs invoked from these exits using the XPI or EXEC CICS commands. However, you need to consider the following:

- There is a restriction on using the XPI early during initialization: do not invoke exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR and INQUIRE_MONITOR_DATA until the second phase of the PLTPI.
- There are also restrictions on the use of EXEC CICS commands in these exits:
 - You cannot use EXEC CICS commands to access terminal control services.
 - You are strongly advised not to use temporary storage, transient data, file control, journal control, or DL/I services, because the resources that you try to access may also be in a state of recovery and therefore “not open for business”. Attempting to access resources in these circumstances causes, at best, serialization of the recovery tasks and, at worst, a deadlock.

If you do issue file control requests in programs invoked from these exits, note that:

- If an exit program acquires an area as a result of a file control request, it is the responsibility of the program to release that area.
- An exit program must not attempt to make any file control requests to a file referring to a VSAM data set with a string number of 1, unless no action is specified for that file during the initialization exit.
- An XRCFCER exit program must not disable or close files, because this could cause further error conditions.
- Your exit program must not issue EXEC CICS commands if the recovery is as the result of an EXEC CICS SYNCPOINT ROLLBACK request.

transaction backout programs exits

- Exit programs that issue EXEC CICS commands must first address the EIB. See “Using CICS services” on page 5.
- Exit programs that issue EXEC CICS commands, and that use the DFHEIENT macro, should use the DFHEIRET macro to set a return code and return to CICS. See “Returning values to CICS” on page 10.
- Exit programs invoked from these exits must be translated with the NOEDF option, if they issue EXEC CICS commands. See “Using EDF with global user exits” on page 6.
- Task-chained storage acquired in an exit program is released at the completion of emergency restart processing. However, the exit program should attempt to release the storage as soon as its contents are no longer needed.
- No exit program should reset either the absent or no-action indicators set by the file control backout program.

Warning: Care should be taken when issuing recursive commands not to cause a loop. For example, it is your responsibility to avoid entering a loop when an RC request is issued from any of the transaction backout program exits. Use of the recursion counter UEPRECUR is recommended.

Enabling the exit programs

To enable these exits, you must either specify the system initialization parameter TBEXITS=(name1,name2,name3,name4), where name1, name2, name3, and name4 are the names of your user exit programs for XRCINIT, XRCINPT, XRCFCER, and XRCOPER, or enable them during the first stage of initialization using a PLTPI program. If you use the TBEXITS parameter to enable these exits, a global work area of 4 bytes is provided. If you use a PLTPI program, you can select the size of the global work area. You can also enable more than one exit program for use at each exit point; the TBEXITS parameter allows only one exit program at each exit point. PLTPI processing is described in Chapter 4, “Writing initialization and shutdown programs” on page 335.

XRCINIT from the DL/I backout program

<i>Table 76. Exit XRCINIT invoked from the DL/I backout program</i>	
When invoked	At the beginning and at the end of the DL/I backout program.
Exit-specific parameters	<p>UEPDBOTE Address of the DL/1 backout (DBO) table. The DBO table can be mapped using the DSECT DFHDBODS.</p> <p>UEPTREQ Address of a 1-byte flag indicating the reason for the call.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p> <p>When UEPTREQ has a value of UEDLINIT, the exit has been invoked at the start of recovery, and when UEPTREQ has a value of UEDLTERM, the exit has been invoked at the end of recovery.</p>
Return codes	UERCNORM Continue processing. No other return codes are supported.
XPI calls	All can be used. See page 165 for details of restrictions.

XRCINIT from the file control backout program

<i>Table 77. Exit XRCINIT invoked from the file control backout program</i>		
When invoked	At the beginning and at the end of the file control backout program.	
Exit-specific parameters	UEPFBOTE	Address of the start of the file backout (FBO) table. The FBO table can be mapped using DSECT DFHFBODS. All of the fields in DSECT DFHFBODS are product-sensitive programming interfaces, except for bit TBFBFAIL of byte TBFBFLG1, which is not a programming interface.
	UEPTREQ	Address of a 1-byte flag indicating the reason for the call. When UEPTREQ has a value of UEFCINIT, the exit has been invoked at the start of file recovery, and when UEPTREQ has a value of UEFCTERM, the exit has been invoked at the end of file recovery.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing. No other return codes are supported.
XPI calls	All can be used. See page 165 for details of restrictions.	

XRCINIT from the message and ISC SRP

<i>Table 78. Exit XRCINIT invoked from the message and ISC SRP</i>		
When invoked	At the beginning and at the end of the message and ISC state recovery program.	
Exit-specific parameters	UEPMBOTE	Address of the message backout (MBO) table. The MBO table can be mapped using the DSECT DFHMBODS.
	UEPTREQ	Address of a 1-byte flag indicating the reason for the call. When UEPTREQ has a value of UEMEINIT, the exit has been invoked at the start of message recovery, and when UEPTREQ has a value of UEMETERM, the exit has been invoked at the end of message recovery.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing. No other return codes are supported.
XPI calls	All can be used. See page 165 for details of restrictions.	

XRCINIT from the user backout program

<i>Table 79. Exit XRCINIT invoked from the user backout program</i>	
When invoked	At the beginning and at the end of the user backout program.
Exit-specific parameters	<p>UEPTBOTE Address of the transaction backout (TBO) table. The TBO table can be mapped using the DSECT DFHTBODS.</p> <p>UEPTREQ Address of a 1-byte flag indicating the reason for the call. When UEPTREQ has a value of UEUSINIT, the exit has been invoked at the start of user recovery, and when UEPTREQ has a value of UEUSTERM, the exit has been invoked at the end of user recovery.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	UERCNORM Continue processing. No other return codes are supported.
XPI calls	All can be used. See page 165 for details of restrictions.

XRCINPT from the file control backout program

<i>Table 80. Exit XRCINPT invoked from the file control backout program</i>	
When invoked	From the file control backout program whenever a file control record has been read from the restart data set.
Exit-specific parameters	<p>UEPFBOTE Address of the entry in the file backout (FBO) table corresponding to the CICS file involved. The FBO table can be mapped using the DSECT DFHFBODS. All of the fields in DSECT DFHFBODS are product-sensitive programming interfaces, except for bit TBFBFAIL of byte TBFBFLG1, which is not a programming interface.</p> <p>UEPLGREC Address of the log record just read. The journal control record can be mapped using the DSECT DFHJCRDS, which is described in Chapter 18, "CICS journaling" on page 521. The type of file control record read from the restart data set can be determined by testing field JCRSTRID (defined in DFHJCRDS) with values provided by copy book DFHFMIDS. Copy book DFHJFCDS defines the file control layout of the variable part of the system prefix section of the log record.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYB Bypass this record.</p>
XPI calls	All can be used. See page 165 for details of restrictions.

XRCINPT from the message and ISC SRP

<i>Table 81. Exit XRCINPT invoked from the message and ISC SRP</i>	
When invoked	From the message and ISC state recovery program whenever a terminal control record has been read from the restart data set.
Exit-specific parameters	<p>UEPMBOTE Address of the message backout (MBO) table. The MBO table can be mapped using the DSECT DFHMBODS.</p> <p>UEPLGREC Address of the log record just read. The journal control record can be mapped using the information supplied in Chapter 18, "CICS journaling" on page 521.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYB Bypass this record.</p>
XPI calls	All can be used. See page 165 for details of restrictions.

XRCINPT from the user backout program

<i>Table 82. Exit XRCINPT invoked from the user backout program</i>	
When invoked	From the user backout program whenever a user-journaled record has been read from the restart data set.
Exit-specific parameters	<p>This parameter is preceded by a 4-byte reserved field in the parameter list.</p> <p>UEPLGREC Address of the log record just read. The journal control record can be mapped using the information supplied in Chapter 18, "CICS journaling" on page 521.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYB Bypass this record.</p>
XPI calls	All can be used. See page 165 for details of restrictions.

XRCOPER from the file control backout program

<i>Table 83. Exit XRCOPER invoked from the file control backout program</i>		
When invoked	From the file control backout program if an error occurs while opening a file control data set.	
Exit-specific parameters	UEPFBOTE	Address of the entry in the file backout (FBO) table corresponding to the CICS file involved. The FBO table can be mapped using the DSECT DFHFBODS. All of the fields in DSECT DFHFBODS are product-sensitive programming interfaces, except for bit TBFBFAIL of byte TBFBFLG1, which is not a programming interface.
	UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. See page 165 for details of restrictions.	

XRCFCER from the file control backout program

<i>Table 84 (Page 1 of 2). Exit XRCFCER invoked from the file control backout program</i>		
When invoked	From the file control backout program when the file control program returns an error during backout processing, or when the file control backout program itself detects an error.	
Exit-specific parameters	UEPFBOTE	Address of the entry in the file backout (FBO) table corresponding to the CICS file involved. The FBO table can be mapped using the DSECT DFHFBODS. All of the fields in DSECT DFHFBODS are product-sensitive programming interfaces, except for bit TBFBFAIL of byte TBFBFLG1, which is not a programming interface.
	UEPLGREC	Address of the log record just read. The journal control record can be mapped using the information supplied in Chapter 18, "CICS journaling" on page 521.
	UEPFCTE	Address of the entry in the file control table (FCT), if any (see note below). The FCT entry can be mapped using the DSECT DFHFCTDS.

Table 84 (Page 2 of 2). Exit XRCFCER invoked from the file control backout program

UEPFCRSP	The address of the file control response byte. This can have one of the following values:
UENORESP	Normal response
UEFILERR	File not found
UEINVREQ	Invalid request
UEILOGIC	VSAM illogical error
UENOTOPN	Not open
UEDISAB	Disabled
UEENDFIL	End of file
UEIOEROR	I/O error
UENOTFND	Record not found
UEDUPREC	Duplicate record
UENOSPAC	No space available
UELENGER	Length error
UEDUPKEY	Duplicate key in alternate index (AIX)
UEPURGED	Transaction canceled
UEDISAST	Disastrous error detected.
Note:	A value of UEPFCRSP other than one of those listed above indicates that File Control detected an invalid request.
UEPERR	Address of a one-byte field containing the error type. The system log record can be mapped using the information supplied in Chapter 18, "CICS journaling" on page 521. The values of the error byte and their meanings are described in "Values of the error byte pointed to by UEPERR" on page 172, and are defined in the DSECT DFHFBODS. All of the fields in DFHFBODS are product-sensitive programming interfaces, except for bit TBFBFAIL of byte TBFBFLG1, which is not a programming interface.
UEPFDATA	Address of a variable-length field containing the data in the file control request.
UEPFLEN	Address of a fullword containing the length of the data in the file control request.
UEPRECUR	Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.
Note:	UEPFCTE, UEPFDATA, and UEPFLEN contain valid values, depending on the type of error reported by UEPFCRSP. For example, if the byte addressed by UEPFCRSP contains 'UEFILERR' (file not found), then UEPFCTE and UEPFDATA are zero.
Return codes	<p data-bbox="688 1617 1438 1675">UERCNORM Continue processing and invoke CICS backout failure control.</p> <p data-bbox="688 1690 1438 1749">UERCBYP Ignore the error (do not invoke CICS backout failure control) and continue.</p> <p data-bbox="688 1764 1438 1818">UERCRTY Retry the request, or reapply the record if the error is 'TBFEWA'.</p>
XPI calls	All can be used. See page 165 for details of restrictions.

Values of the error byte pointed to by UEPERR

Field UEPERR in the XRCFCER parameter list points to an error byte which contains one of the following values:

TBFEGU

An error response has been returned from the file control program (DFHFCVS) while servicing a GET UPDATE request. DFHFBCBP has attempted to retrieve the existing copy of the record before backing it out. Use parameter UEPFCRSP in combination with the type of record pointed to by parameter UEPLGREC (before copy of a “read for update” record or new copy of a “write add” to be deleted) to determine the specific problem.

TBFELE

The area acquired in DFHFCVS is not large enough to receive the before-copy data picked up from the restart data set to perform the backout. Parameter UEPFDATA points to the data. UEPFLEN points to a field containing the length. Parameter UEPFCRSP does not apply to this error.

TBFEPU

An error response has been returned from DFHFCVS while servicing a PUT UPDATE request. DFHFBCBP has attempted to replace the existing copy of the record on the file with the “before-copy” held in the log record pointed to by UEPLGREC. Use parameter UEPFCRSP to determine which error occurred.

TBFEPN

An error response has been returned from DFHFCVS while servicing a PUT NEW request. DFHFBCBP has attempted to add the “before-copy” of a deleted VSAM KSDS record. Use parameter UEPFCRSP to determine which error occurred.

TBFEWA

If the record read from the restart data set is a WRITE ADD, the record is also read in from the file through a GET UPDATE. For BDAM and VSAM-ESDS data sets, no delete function exists. Because DFHFBCBP does not attempt to delete the record that has been added, you are given the opportunity in your exit program to “mark” the existing record on the file as deleted according to application-dependent logic. Mark the record at the address pointed to by UEPFDATA. If you want this version to be reapplied, specify return code 'UERCRTY'.

Parameter UEPFCRSP does not apply to this sort of error, and will contain a normal response from the READ UPDATE. Parameter UEPFDATA points to the unmatched record. UEPFLEN points to the length of this record.

TBFEVD

An error response has been returned from DFHFCVS while servicing a VSAM DELETE request. DFHFBCBP has attempted to delete a new record added to a VSAM key-sequenced data set. Use parameter UEPFCRSP to determine which error occurred.

Transaction manager domain exit **XXMATT**Table 85. Exit *XXMATT*

When invoked	During transaction attach. This exit is able to change some of the attributes of the transaction that is being attached.
Exit-specific parameters	<p>UEPTRANID The address of transaction id (see Notes).</p> <p>UEPUSER The address of the userid associated with the transaction if the current task is a user task (see Notes).</p> <p>UEPTERM The address of the terminal id associated with the transaction, if any (see Notes).</p> <p>UEPPROG The address of the application program name for this transaction, if any (see Notes).</p> <p>UEPATPTI The address of a 4-byte field containing the primary transaction id. You can change the primary transaction id by modifying the addressed field.</p> <p>UEPATOTI The address of the 4-byte attach transaction id. A transid of X'00000000' indicates that a transid was not supplied on the attach.</p> <p>UEPATTPL The address of an area containing the length of the attach TPName. A length of zero indicates that a TPName was not supplied on the attach.</p> <p>UEPATTPA The address of a fullword containing the address of the attach TPName. The attach TPName can be 1 through 64 bytes long, as defined by UEPTTPL.</p> <p>UEPATLOC The address of a 1-byte field indicating whether the transaction was found. Equated values are: UEATFND The transaction was found UEATNFND The transaction was not found.</p> <p>UEPATTST The address of a one-byte transaction definition state. Equated values for the definition state are: UEATENAB Transaction is enabled UEATDISA Transaction is disabled.</p> <p>UEPATTTK The address of a doubleword containing a transaction token. Note that some of the transaction manager XPI calls require this token to identify the transaction that is being attached.</p>
Return codes	UERCNORM Continue attach processing.
XPI calls	<p>The user exit can inquire on the transaction being attached using the UEPATTTK transaction token as input to the XMIQ INQUIRE_TRANSACTION XPI call.</p> <p>The exit can also set the total priority and tclass using the XMIQ SET_TRANSACTION XPI call.</p> <p>Most of the XPI calls can be used, but with caution since typically this exit is invoked under the TCP task. Thus it is advisable not to issue any XPI calls that might cause the TCP task to wait.</p>

Notes:

1. The following XPI calls can be useful for obtaining information that could be used to modify the attach of a transaction:

- INQUIRE_TRANSACTION
- INQUIRE_MXT
- INQUIRE_TCLASS
- INQUIRE_TRANDEF
- INQUIRE_SYSTEM

2. The fields UEPTRANID, UEPUSER, UEPTERM, and UEPPROG are common to many of the domain global user exit points, and normally return values associated with the current user task. In the case of XXMATT, however, the user task that is being attached is **not** the current task when the exit is invoked. Until task attach is complete, the current task is the CICS task that is performing the attach.

When the task being attached is for a task started by an immediate START command; that is, a START without an interval, the current task is the task that issues the START command, and the fields contain values associated with that task.

Transient data program exits XTDREQ, XTDIN, and XTDOUT

XTDREQ

<i>Table 86. Exit XTDREQ</i>	
When invoked	Before request analysis.
Exit-specific parameters	<p>UEPTDQUE Address of 4-byte TD queue name.</p> <p>UEPTDTYP Address of 1-byte TD request type. Values are:</p> <p>UEPTDPUT PUT request</p> <p>UEPTDGET GET request</p> <p>UEPTDPUR PURGE request.</p>
Return codes	<p>UERCNORM Continue TD processing.</p> <p>UERCTDOK Quit TD processing – returning 'NORMAL' to the caller.</p> <p>UERCTDNA Quit TD processing – returning 'NOTAUTH' to the caller.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>You can use:</p> <p> INQ_APPLICATION_DATA</p> <p> INQUIRE_SYSTEM</p> <p> WAIT_MVS</p> <p>Do not use any other calls.</p>

XTDIN

<i>Table 87. Exit XTDIN</i>	
When invoked	After receiving data from QSAM (for extrapartition) or VSAM (for intrapartition).
Exit-specific parameters	<p>UEPTDQUE Address of the 4-byte TD queue name.</p> <p>UEPTDAUD Address of the unmodified TD data.</p> <p>UEPTDLUD Address of the fullword length of the unmodified TD data.</p> <p>UEPTDAMD Address of the TD data modified by the exit program.</p> <p>UEPTDLMD Address of the fullword length of the TD data modified by the exit program.</p>
Return codes	<p>UERCNORM Continue TD processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>You can use:</p> <p> INQ_APPLICATION_DATA</p> <p> INQUIRE_SYSTEM</p> <p> WAIT_MVS</p> <p>Do not use any other calls.</p>

XTDOUT

<i>Table 88. Exit XTDOUT</i>	
When invoked	Before passing the data to a QSAM (for extrapartition) or VSAM (for intrapartition) user-defined transient data queue.
Exit-specific parameters	<p>UEPTDQUE Address of the 4-byte TD queue name.</p> <p>UEPTDAUD Address of the unmodified TD data.</p> <p>UEPTDLUD Address of the fullword length of the unmodified TD data.</p> <p>UEPTDAMD Address of the TD data modified by the exit program.</p> <p>UEPTDLMD Address of the fullword length of TD data modified by the exit program.</p> <p>UEPTDNUM Address of the fullword containing the number of items in the list.</p> <p>UEPTDCUR Address of the fullword containing the number of the current item.</p>
Return codes	<p>UERCNORM Continue TD processing.</p> <p>UERCTDOK Quit TD processing – returning 'NORMAL' to the caller.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>You can use:</p> <p style="padding-left: 40px;">INQ_APPLICATION_DATA</p> <p style="padding-left: 40px;">INQUIRE_SYSTEM</p> <p style="padding-left: 40px;">WAIT_MVS</p> <p>Do not use any other calls.</p>

Transient data EXEC interface program exits XTDEREQ and XTDEREQC

The XTDEREQ exit allows you to intercept a transient data request before any action has been taken on it by transient data. The XTDEREQC exit allows you to intercept a transient data request after transient data has completed its processing.

Using XTDEREQ, you can:

- Analyze the request to determine its type, the keywords specified, and their values.
- Modify any value specified by the request before the command is executed.
- Set return codes to specify that either:
 - CICS should continue with the (possibly modified) request.
 - CICS should bypass the request. (Note that if you set this return code, you must also set up return codes for the EXEC interface block (EIB), as if you had processed the request yourself.)

Using XTDEREQC, you can:

- Analyze the request, to determine its type, the keywords specified, and their values.
- Set return codes for the EIB.

Both exits are passed six parameters as follows:

- The address of the command-level parameter structure
- The address of a token (UEPTDTOK) used to pass 4 bytes of data from XTDEREQ to XTDEREQC
- The addresses of copies of three pieces of return code information from the EIB
- The address of a token (UEPTSTOK) that is valid throughout the life of a task
- The address of an exit recursion count (UEPRECUR).

Example program

CICS supplies—in hardcopy only—an example program, DFH\$XTSE, that shows how to modify fields in the command-level parameter structure passed to EXEC interface exits. DFH\$XTSE is listed on page 717.

XTDEREQ

<i>Table 89. Exit XTDEREQ</i>	
When invoked	Before CICS processes a transient data API request.
Exit-specific parameters	<p>UEPCLPS Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 181.</p> <p>UEPTDOK Address of the 4-byte token to be passed to XTDEREQC. This allows you, for example, to pass a work area to exit XTDEREQC.</p> <p>UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.</p> <p>UEPRES P Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.</p> <p>UEPRES P2 Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.</p> <p>UEPTSTOK Address of a 4-byte token that is valid throughout the life of a task. See “Use of the task token UEPTSTOK” on page 184.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCBYP The transient data EXEC interface program should ignore this request.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>All can be used.</p> <p>Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.</p>
API commands	All can be used.

Warning: Care should be taken when issuing recursive commands. For example, you must avoid entering a loop when issuing a transient data request from the XTDEREQ exit. Use of the recursion counter UEPRECUR is recommended.

XTDEREQC

<i>Table 90. Exit XTDEREQC</i>	
When invoked	After a transient data API request has completed, and before return from the transient data EXEC interface program.
Exit-specific parameters	<p>UEPCLPS Address of the command-level parameter structure. See “The UEPCLPS exit-specific parameter” on page 181.</p> <p>UEPTDOK Address of the 4 byte token to be passed to XTDEREQC. This allows you, for example, to pass a work area to exit XTDEREQC.</p> <p>UEPRCODE Address of a 6-byte hexadecimal copy of the EIB return code ‘EIBRCODE’. For details of EIB return codes, refer to the <i>CICS/ESA Application Programming Reference</i> manual.</p> <p>UEPRES Address of a 4-byte binary copy of the EIB response code ‘EIBRESP’.</p> <p>UEPRES2 Address of a 4-byte binary copy of the EIB response code ‘EIBRESP2’.</p> <p>UEPTSTOK Address of a 4-byte token that is valid throughout the life of a task. See “Use of the task token UEPTSTOK” on page 184.</p> <p>UEPRECUR Address of a halfword recursion counter. The counter is set to 0 when the exit is first invoked, and is incremented for each recursive call.</p>
Return codes	<p>UERCNORM Continue processing.</p> <p>UERCPURG Task purged during XPI call.</p>
XPI calls	<p>All can be used.</p> <p>Although the exit permits the use of XPI GETMAIN and FREEMAIN calls, we recommend that you use the EXEC CICS GETMAIN and FREEMAIN commands instead.</p>
API commands	All can be used.

Warning: Care should be taken when issuing recursive commands. For example, you must avoid entering a loop when issuing a transient data request from the XTDEREQC exit. Use of the recursion counter UEPRECUR is recommended.

The command-level parameter structure

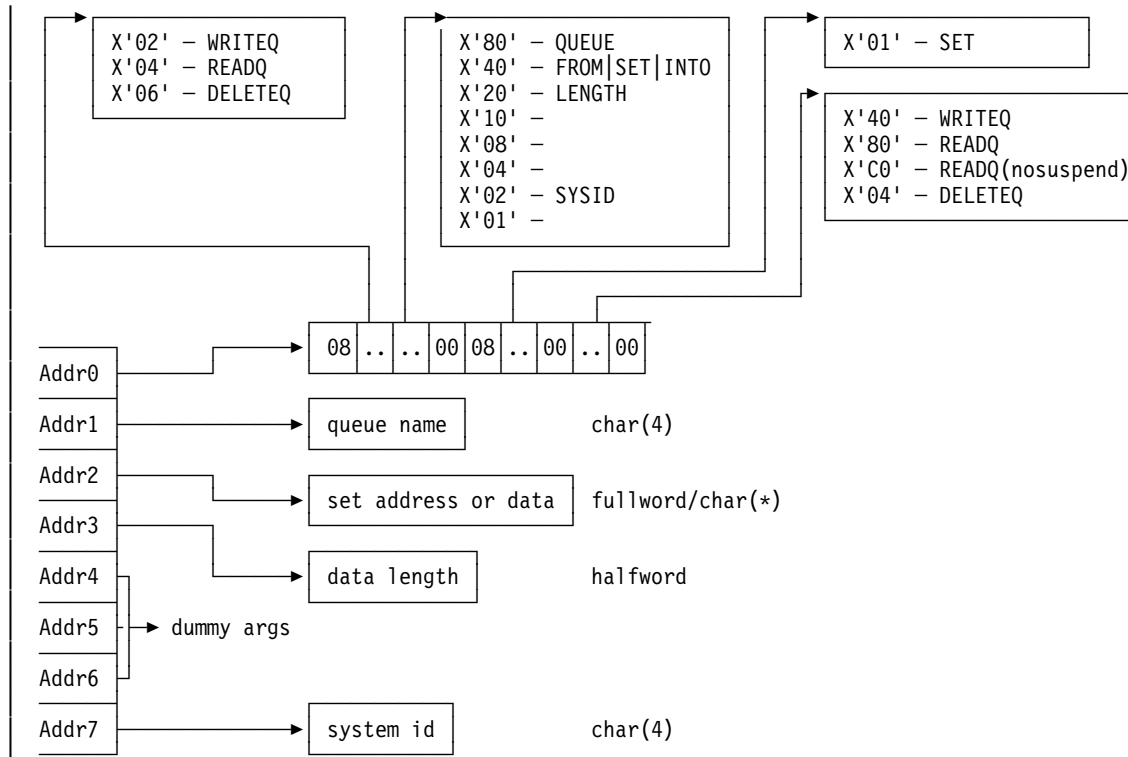


Figure 3. The command-level parameter structure for transient data

The command-level parameter structure consists of a series of addresses. The first address points to the EXEC interface descriptor (EID), which consists of an 8-byte area that describes the type of request and identifies each keyword specified with the request. The remaining addresses point to pieces of data associated with the request. (For example, the second address points to the queue name.)

You can examine the EID to determine the type of request and the keywords specified. You can examine the other parameters in the list to determine the values of the keywords. You can also modify values of keywords specified on the request. (For example, you could change the sysid specified in the request.)

End of parameter list indicator

The high-order bit is set on in the last address set in the parameter list to indicate that it is the last one in the list. On return from your user exit program, CICS scans the parameter list for the high-order bit to find the last parameter. Therefore, if you modify the length of the parameter list, you must also reset the high-order bit to indicate which is the new last address.

For example, if the parameter list specifies only the first two addresses (TD_ADDR0, the address of the EID, and TD_ADDR1, the address of the name of the queue named in a DELETEQ request), the high-order bit is set on in TD_ADDR1. If you extend the parameter list by setting the address of a SYSID in TD_ADDR7, you must reset the high-order bit in TD_ADDR1 and set it on in TD_ADDR7 instead.

The maximum size of parameter list is supplied to the exit, thus allowing your exit program to add any parameters not already specified without needing to first obtain more storage.

The original parameter list, as it was before XTDEREQ was invoked, is restored after the completion of XTDEREQC. It follows that the execution diagnostic facility (EDF) displays the original command before **and** after execution. **EDF does not display any changes made by the exit.**

The UEPCLPS exit-specific parameter: The UEPCLPS exit-specific parameter is included in both exit XTDEREQ and exit XTDEREQC. It is the address of the command-level parameter structure. The command-level parameter structure contains 8 addresses, TD_ADDR0 through TD_ADDR7. It is defined in the DSECT TD_ADDR_LIST, which you should copy into your exit program by including the statement COPY DFHTDUED.

The command-level parameter list is made up as follows:

TD_ADDR0

is the address of an 8-byte area called the EID, which is made up as follows:

TD_GROUP
TD_FUNCT
TD_BITS1
TD_BITS2
TD_EIDOPT5
TD_EIDOPT6
TD_EIDOPT7

TD_GROUP Always X'08', indicating that this is a transient data request.

TD_FUNCT One byte that defines the type of request:

X'02' WRITEQ
X'04' READQ
X'06' DELETEQ

TD_BITS1 Existence bits that define which arguments were specified. To obtain the argument associated with a keyword, you need to use the appropriate address from the command-level parameter structure. Before using this address, you must check the associated existence bit. If the existence bit is set off, the argument was not specified in the request and the address should not be used.

X'80' Set if the request contains an argument for the QUEUE keyword. If set, **TD_ADDR1** is meaningful.

X'40' Set if the request contains an argument for any of the INTO, SET, or FROM keywords. If set, **TD_ADDR2** is meaningful.

X'20' Set if the request contains an argument for the LENGTH keyword. If set, **TD_ADDR3** is meaningful.

transient data EXEC interface program exits

X'02' Set if the request contains an argument for the SYSID keyword. If set, **TD_ADDR7** is meaningful.

TD_BITS2 Two bytes not used by transient data.

TD_EIDOPT5 Indicates whether certain keywords were specified on the request.

X'01' SET (and not INTO) was specified.

TD_EIDOPT6 One byte not used by transient data.

TD_EIDOPT7 Indicates whether certain functions and/or keywords were specified on the request:

X'40' WRITEQ specified

X'80' READQ specified

X'C0' READQ(nosuspend) specified

X'04' DELETEQ specified

TD_ADDR1

is the address of a 4-byte area containing the name from QUEUE.

TD_ADDR2

is the address of one of the following:

- A 4-byte address from SET (if the request is READQ and **TD_EIDOPT5** indicates that this is SET).
- Data from INTO (if the request is READQ and **TD_EIDOPT5** indicates that this is not SET). You cannot modify this bit in your user exit.
- Data from FROM (if the request is WRITEQ).

TD_ADDR3

is the address of one of the following:

- The halfword value of LENGTH (if the request is READQ or WRITEQ).
Warning: For requests that specify INTO, do not change the value of LENGTH to a value greater than that specified by the application. To do so causes a storage overlay in the application.

TD_ADDR4

is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR5

is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR6

is the address of a value intended for CICS internal use only. It must not be used.

TD_ADDR7

is the address of an area containing the value of SYSID.

TD_ADDR8

is the address of a value intended for CICS internal use only. It must not be used.

Modifying fields in the command-level parameter structure: Some fields that are passed to transient data are used as input to the request, some are used as output fields, and some are used for both input and output. The method your user exit program uses to modify a field depends on the usage of the field.

The following are always input fields:

QUEUE
FROM
SYSID

The following are always output fields:

INTO
SET

LENGTH is an input field on a WRITEQ request, and an output field on a READQ request that specifies SET. It is both an input and an output field on a READQ request that specifies INTO.

Modifying input fields: The correct method of modifying an input field is to create a new copy of it, and to change the address in the command-level parameter list to point to your new data.

Warning: You must never modify an input field by altering the data that is pointed to by the command-level parameter list. To do so would corrupt storage belonging to the application program and would cause a failure when the program attempted to reuse the field.

Modifying output fields: The technique described in “Modifying input fields” is not suitable for modifying output fields. (The results would be returned to the new area instead of the application’s area, and would be invisible to the application.)

An output field is modified by altering the data that is pointed to by the command-level parameter list. In the case of an output field, you can modify the application’s data in place, because the application is expecting the field to be modified.

Modifying fields used for both input and output: An example of a field that is used for both input and output is LENGTH on a READQ request that specifies INTO. You can treat such fields in the same way as output fields, and they are considered to be the same.

Modifying the EID: It is not possible to modify the EID to make major changes to requests, such as changing a READQ request to a WRITEQ request.

However, you can make minor changes to requests, such as turning on the existence bit for SYSID so that the request can be changed into one that is shipped to a remote system.

The list that follows shows the bits in the EID that **can** be modified. Any attempt to modify any other part of the EID is ignored.

TD_BITS1

X'20' The existence bit for LENGTH.

transient data EXEC interface program exits

X'02' The existence bit for SYSID.

TD_EIDOPT5

X'01' Existence bit for SET keyword. You cannot modify this bit from your user exit.

TD_EIDOPT7

Changes to TD_EIDOPT7 are limited to READQ requests. X'80'–READQ is interchangeable with X'C0'–READQ(nosuspend). No other changes may be made to this byte.

The EID is reset to its original value before return to the application program. That is, changes made to the EID are retained for the duration of the transient data request only.

Warning: Your user exit program is prevented from making major changes to the EID. However, you must take great care when making the minor modifications that are permitted.

Use of the task token UEPTSTOK: UEPTSTOK provides the address of a 4-byte area that you can use to pass information between successive transient data requests in the same task. (By contrast, UEPTDTOK is usable only for the duration of a single transient data request, because its contents may be destroyed at the end of the request.) For example, if you need to pass information between successive invocations of the XTDEREQ exit, UEPTSTOK provides a means of doing this.

The EIB: Copies of EIBRCODE, EIBRESP, and EIBRESP2 are passed to the exit, so that you can:

- Modify or set completion information in XTDEREQ and XTDEREQC
- Examine completion information in XTDEREQC.

You can update the copies of EIBRCODE, EIBRESP, and EIBRESP2 that you are given in the parameter list. Transient data copies your values into the real EIB after the completion of XTDEREQC; or if you specify a return code of 'bypass' in XTDEREQ.

You must set valid transient data responses. You must set all three of EIBRCODE, EIBRESP, and EIBRESP2 to a consistent set of values, such as would be set by CICS transient data to describe a valid completion. **CICS does not police the consistency of EIBRCODE, EIBRESP, and EIBRESP2.** However, if EIBRCODE is set to a non-zero value and EIBRESP is set to zero then CICS will override EIBRESP with a non-zero value. To aid you in setting the values of EIBRCODE, EIBRESP, and EIBRESP2, the values used by transient data are specified in DFHTDUED.

VTAM terminal management program exit XZCATT

Table 91. Exit XZCATT

When invoked	Before task attach for terminal tasks.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
	UEPTPN	Address of the APPC transaction process name (TPN), or the LU6.1 process name (DPN), whose length is addressed by the parameter UEPTPNL.
	UEPTPNL	Address of a 1-byte field containing the length of the TPN or DPN.
	UEPTRAN	Address of the 4-byte transaction ID. Note: The exit program must not change the TRANSID of tasks started by automatic transaction initiation (ATI). (This is because CICS needs to match the TRANSID in its program control table with the TRANSID in the automatic initiate descriptor (AID) that was created in the AOR.)
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used.	

VTAM working-set module exits XZCIN, XZCOUT, XZCOUT1, and XZIQUE

Note: None of the exits in the VTAM working-set module is available for advanced program-to-program communication (APPC, or LUTYPE6.2) links.

XZCIN

<i>Table 92. Exit XZCIN</i>		
When invoked	After an input event.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. However, we do not recommend that you use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application may experience problems.	

XZCOUT

<i>Table 93. Exit XZCOUT</i>		
When invoked	Before an output event.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces. Note: APAR PQ09804, documented 19th January 1998: In certain circumstances—for example, when XZCOUT is invoked before the send of a NULL RU—UEPTIOA contains zeroes.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. However, we do not recommend that you use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application may experience problems.	

#

XZCOUT1

<i>Table 94. Exit XZCOUT1</i>		
When invoked	Before a message is broken into RUs.	
Exit-specific parameters	UEPTCTTE	Address of the terminal control table terminal entry (TCTTE). The TCTTE can be mapped using the DSECT DFHTCTTE.
	UEPTIOA	Address of the terminal input/output area (TIOA). Your exit program should not change the address. The TIOA can be mapped using the DSECT DFHTIOA. However, fields TIOASAL and TIOASCA are not programming interfaces.
Return codes	UERCNORM	Continue processing.
XPI calls	All can be used. However, we do not recommend that you use a GETMAIN call to obtain terminal-class storage for use as a replacement TIOA. This is because there are several internal pointers to the TIOA, and if any one of these is not updated the application may experience problems.	

XZIQUE exit for managing intersystem queues

You can use the XZIQUE exit to control the number of queued requests for sessions on intersystem links (allocate queues).

Note: There are several methods that you can use to control the length of intersystem queues. For a description of the various methods, see the *CICS/ESA Intercommunication Guide*.

The XZIQUE exit enables you detect queuing problems (bottlenecks) early. It extends the function provided by the XISCONA global user exit (introduced in CICS/ESA 3.3 and described on page 77), which is invoked only for function shipping requests. XZIQUE is invoked for transaction routing, DPL, asynchronous processing, and distributed transaction processing requests, as well as for function shipping. Compared with XISCONA, it receives more detailed information on which to base its decisions.

XZIQUE enables allocate requests to be queued or rejected, depending on the length of the queue. It also allows a connection on which there is a bottleneck to be terminated and then re-established.

Interaction with the XISCONA exit

There is no interaction between the XZIQUE and XISCONA global user exits. If you enable both exits, XISCONA and XZIQUE could both be invoked for function shipping requests, which is not recommended. You should ensure that only one of these exits is enabled. Because of it provides more function and greater flexibility, it is recommended that you use XZIQUE rather than XISCONA.

If you already have an XISCONA global user exit program, you could possibly modify it for use at the XZIQUE exit point.

When the XZIQUE exit is invoked

The XZIQUE global user exit is invoked, if it is enabled, at the following times:

- Whenever CICS tries to acquire a session with a remote system and there is no free session available. It is invoked whether or not you have specified the QUEUELIMIT option on the CONNECTION definition, and whether or not the limit has been exceeded. It is not invoked if the allocate request specifies NOQUEUE or NOSUSPEND.

Requests for sessions can arise in a number of ways, such as explicit EXEC CICS ALLOCATE commands issued by DTP programs, or by transaction routing or function shipping requests.

- Whenever an allocate request succeeds in finding a free session, after the queue on the connection has been purged by a previous invocation of the exit program. In this case, your exit program can indicate that CICS is to continue processing normally, resuming queuing when necessary.

Using an XZIQUE global user exit program

When the exit is enabled, your XZIQUE global user exit program is able to check on the state of the allocate queue for a particular connection in the local system. Information is passed to the exit program in a parameter list, that is structured to provide data about non-specific allocate requests, or requests for specific modegroups, depending on the session request. Non-specific allocate requests are for MRO, LU6.1, and APPC sessions that do not specify a modegroup.

Using the information passed in the parameter list, your global user exit program can decide (based on queue length, for example) whether CICS is to queue the allocate request. Your program communicates its decision to CICS by means of one of the return codes CICS provides. These are:

UERCAQUE

This return code indicates that CICS is to queue the allocate request.

The total number of allocate requests queued against the connection is provided in field A14ESTAQ of the system entry statistics (for all non-specific allocates) or A20ESTAQ of the mode entry statistics (for specific modegroup allocates). See DSECTs DFHA14DS or DFHA20DS for details. CICS passes to the exit program, in the exit specific parameter UEPQUELIM, the QUEUELIMIT parameter from the connection definition.

If the limit has not been reached, you can return control to CICS with return code UERCAQUE.

UERCAPUR

This return code indicates that CICS is to reject the allocate request and return SYSIDERR to the application program, but leave the existing queue unchanged.

If the number of queued allocate requests has reached the limit set on the QUEUELIMIT parameter for the connection, you can request that CICS rejects the request. However, you should first check whether the state of the link is satisfactory. This means checking that the rate of allocation of sessions is acceptable. Use the time the queue was started, the current time, and the total number of allocates processed since the queue began, to determine the rate at which CICS is processing requests. The relevant fields are: UEPSAQTS and UEPSACNT for non-specific allocate requests; and UEPMAQTS and UEPMACNT for specific modegroup requests.

You can compare the calculated time with either:

1. The parameter from the connection definition, MAXQTIME, which is passed in the exit specific parameter UEPQMXQT, or
2. Some other preset time value

to determine whether CICS is allocating requests for sessions on this connection at an acceptable rate. If the processing time using this kind of formula is acceptable, return control to CICS with return code UERCAPUR to purge only this request.

UERCAKLL or UERCAKLM

These return codes indicate that you want CICS to deal with the request as follows:

- UERCAKLL—reject this request, purge all other queued allocate requests on this connection, and send an information message to the operator console.
- UERCAKLM—reject this request, purge all other queued modegroup allocate requests on this connection, and send an information message to the operator console.

If the queue limit has been reached but the performance of allocate processing against the queue is below the acceptable limits defined in your user exit program, you can return control to CICS as follows:

- For non-specific allocate requests, use return code UERCAKLL. UERCAKLL also returns SYSIDERR to all application programs waiting on the purged allocate requests. CICS sets the UEPFLAG parameter to UEPRC8 on subsequent calls to your XZIQUE exit program to indicate that UERCAKLL was returned previously to purge the queue.
- For specific modegroup allocate requests, use return code UERCAKLM. UERCAKLM also returns SYSIDERR to all application programs waiting on the purged allocate requests. CICS sets the UEPFLAG parameter to UEPRC12 on subsequent calls to your XZIQUE exit program to indicate that UERCAKLM was returned previously to purge the queue.

Purging a queue that is causing congestion in the flow of tasks frees task slots that are needed to prevent the system becoming clogged. The more you allow a session queue to grow, the more likely you are to reach the task ceiling set by the MAXT parameter, and then cause a queue of incoming tasks in the local region that cannot be attached. Note that some internal CICS requests (such as those for the LU services model transactions CLS1, CLS2, and CLS3) are not purged by return codes UERCAKLL and UERCAKLM.

If a queue has been purged previously (with UERCAKLL or UERCAKLM) but there are no queued requests currently, check the number of successful allocates since the queue was last purged. For non-specific allocate requests, this number is in UEPSARC8, and for specific modegroup requests, this number is in UEPMAR12. If no requests of this type have been allocated on this connection since the queue was last purged, the problem that caused the purge previously has not been resolved, and this request should be rejected with UERCAPUR.

If the UEPSARC8 or UEPMAR12 parameters show that allocates are being processed, you should use UERCAQUE to resume queuing of requests. If you return with UERCAQUE in this case, CICS issues an information message to the console to signal that queuing has been resumed.

Note: The address of the system entry statistics record, UEPCONST, is supplied for both non-specific and specific modegroup allocate requests.

The address of the modegroup statistics record, UEPMODST, is set to zeros for non-specific allocate requests. This address is supplied only if the request is for a specific modegroup.

If the exit is invoked after a successful allocate following the suppression of queuing, you can use the following return code:

UERCNORM

This return code indicates that CICS is to resume normal processing on the link, including queuing of requests.

Statistics fields in DFHA14DS and DFHA20DS

There are some statistics fields that your XZIQUE global user exit program can use to control queues.

A14EALRJ: Each time an XZIQUE global user exit program returns with a request to reject a request, CICS increments a new field in the system entry connection statistics. This is A14EALRJ (allocate rejected) in DSECT DFHA14DS. This field is provided to help you to tune the queue limit. Normally, if the number of sessions and the queue limit defined for a link are correctly balanced, and there has been no abnormal congestion on the link, the A14EALRJ should be zero. If the rejected allocates field is non-zero it probably indicates that some action is needed.

A14EQPCT and A20EQPCT: Each time an XZIQUE global user exit program returns with a request to purge a queue, CICS increments a new field in either the system entry or mode entry connection statistics. These fields are:

A14EQPCT The count of the number of times the queue has been purged for the connection as a whole.

A20EQPCT The count of the number of times the mode group queue has been purged.

For detailed information about statistics fields, what they contain and how they are updated, see the *CICS/ESA Performance Guide*.

<i>Table 95 (Page 1 of 3). Exit XZIQUE</i>			
When invoked	Whenever:	<ol style="list-style-type: none"> 1. An allocate request for a session is about to be queued 2. An allocate request succeeds following previous suppression of queuing. 	
Exit-specific parameters	UEPZDATA	Address of the 70-byte area containing the information listed below. This area is mapped by the DSECT in copybook DFHXZIDS.	
Area addressed by UEPZDATA	UEPSYSID	The 4-byte SYSID of the connection.	
	UEPREQ	A 2-byte origin-of-request code, which can have the following values: <ul style="list-style-type: none"> TR Transaction routing FS Function shipping (includes distributed program link) AL Other kinds of intercommunication (for example, distributed transaction processing (DTP) or CPI Communications). 	
	UEPREQTR	The 4-byte identifier of the requesting transaction (applicable only when the origin-of-request code is FS or AL).	
	UEPTRAN	The 4-byte identifier of the transaction being routed (applicable only when origin of request is TR).	
	UEPFLAG	A 1-byte flag indicating whether a return code 8 or return code 12 was issued last time the exit was invoked. <ul style="list-style-type: none"> UEPRC8 The exit program returned control to CICS on the previous invocation with return code 8. UEPRC12 The exit program returned control to CICS on the previous invocation with return code 12. 	
	UEPPAD	A 1-byte padding field.	
	UEPFSPL	Address of the 10-byte function shipping parameter list.	
	UEPCONST	Address of the 98-byte system entry statistics record (this can be mapped using DSECT DFHA14DS).	
	UEPMODST	Address of the 70-byte modegroup statistics record for the modegroup specified in the relevant CICS profile. This field applies only to APPC connections for a specific allocate. For LU61, IRC, or non-specific APPC allocates, it contains zero.	
			The statistics record can be mapped using DSECT DFHA20DS. The modegroup name field (A20MODE) may contain blanks. The record is followed by a fullword of X'FFFFFFFF'.

Table 95 (Page 2 of 3). Exit XZIQUE

Area addressed by UEPZDATA (continued)	UEPSTEX	<p>A 6-byte area containing additional current statistics for APPC that are not already in the modegroup statistics record (DFHA20DS). For specific allocates, the numbers refer to the specified modegroup only. For non-specific allocates, they refer to the whole connection—that is, they are the totals of each modegroup.</p> <p>The 6-byte area contains:</p> <p>UEPEBND A halfword binary field containing the number of bound sessions</p> <p>UEPEWWT A halfword binary field containing the number of contention winners with tasks</p> <p>UEPELWT A halfword binary field containing the number of contention losers with tasks.</p>
	UEPEMXQT	A halfword binary field containing the maximum queuing time specified for the connection (MAXQTIME on the CONNECTION resource definition).
	UEPMDGST	<p>Address of a set of 70-byte modegroup statistics records—one for each user modegroup for the connection. This field applies only to APPC connections for a non-specific allocate. For LU61, IRC, and APPC specific allocates, it contains zero.</p> <p>Each statistics record can be mapped using DSECT DFHA20DS. The modegroup name field (A20MODE) may contain blanks. The end of the set of records is indicated by a fullword of X'FFFFFFFF'.</p>
	Non-specific allocates data:	
	The following three fields contain data relating to MRO, LU6.1, and non-specific APPC allocates:	
	UEPSAQTS	A double-word binary field containing the time stamp from the TCT system entry indicating the time the queue of non-specific requests was started.
	UEPSACNT	A half-word binary field containing the number of all non-specific allocates processed since the queue was started (see UEPSAQTS for the start time).
	UEPSARC8	A half-word binary field containing the number of sessions freed since the queue was last purged as a result of a UEPCAKLL return code to CICS.

<i>Table 95 (Page 3 of 3). Exit XZIQUE</i>	
<p>Specific allocates data: The following three fields contain data relating to specific modegroup allocates. They are applicable only when UEPMODST is non-zero (that is, it contains the address of the relevant modegroup statistics).</p>	
UEPMAQTS	A double-word binary field containing the time stamp from the TCT mode entry indicating the time that the modegroup queue was started for this specific modegroup.
UEPMACNT	A half-word binary field containing the number of all specific allocates for this modegroup processed since the queue was started (see UEPMAQTS for the start time).
UEPMAR12	A half-word binary field containing the number of modegroup sessions freed since the queue was last purged as a result of a UEPCAKLL return code to CICS.
UEPQUELM	A half-word binary field containing the queue limit specified for this connection (QUEUELIMIT on the CONNECTION definition).
Return codes	<p>In the case of an allocate that is about to be queued, use one of the following:</p> <p>UERCAQUE Queue the allocate request.</p> <p>UERCAPUR Reject the allocate request with SYSIDERR.</p> <p>UERC AKLL Reject this allocate request with SYSIDERR. Purge all other queued allocate requests and send an information message to the operator console. CICS also returns SYSIDERR to all application programs waiting on the purged allocate requests.</p> <p>UERC AKLM Reject this allocate request for the modegroup and return SYSIDERR. Purge all other queued allocate requests for the modegroup specified on this allocate request and send an information message to the operator console. Retry the modegroup after an interval.</p> <p>UERC PURG Task purged during XPI call.</p> <p>In the case of a successful allocate following the use of UERC AKLL or UERC AKLM, on a previous invocation of the exit, use one of the following:</p> <p>UERC NORM Resume normal operation of the link or modegroup.</p> <p>UERC PUR Reject the allocate request with SYSIDERR.</p>
XPI calls	All can be used.

Designing an XZIQUE global user exit program

The functions of your XZIQUE exit should be designed:

1. To control of the number of tasks (and the amount of associated resource) that are waiting in a queue for a free intersystem session. Waiting tasks can degrade the performance of the local system.

2. To detect poor response from the receiving (remote) system and to notify the operator (or automatic operations program).
3. To cause CICS to issue a message when the link resumes normal operation.

The XZIQUE global user exit parameter list is designed to support these objectives.

Design considerations

The information passed at XZIQUE is designed to enable your XZIQUE global user exit program to:

- Avoid false diagnosis of problems on the connection by distinguishing poor response times from a complete bottleneck
- Ensure that a link resumes normal operation quickly and without operator intervention once any problem in a remote system is resolved.

Some guidance on the use of IRC/ISC statistics

CICS adds an entry for unsatisfied allocate requests to the following queues:

Non-specific (generic) allocate queue

All non-specific allocate requests are queued in this single queue. CICS makes the total number of entries in this queue available in the system entry statistics field A14ESTAQ, to which your global user exit program has access by means of the address of the system entry statistics, which is passed in UEPCONST.

Specific modegroup allocate queues

Specific allocate requests are queued in the appropriate modegroup queue—one queue for each specific modegroup name. CICS makes the total number of entries in all these queues available, as a single total, in the mode entry statistics field A20ESTAQ, to which your global user exit program has access by means of the address of the mode entry statistics, which is passed in UEPMODST.

Sample exit program design

A sample XZIQUE exit program is provided with CICS/ESA 4.1 as a base for you to design your own global user exit program. It is called DFH\$XZIQ, and is supplied in the CICS410.SDFHSAMP library. The DSECT used by the sample program to map the area addressed by UEPZDATA is called DFHXZIDS, and this is supplied in the CICS410.SDFHMAC library.

As supplied, the sample exit program implements the same basic function as described for the QUEUELIMIT and MAXQTIME parameters on the connection resource definition. If the XZIQUE exit is not enabled, CICS uses these parameters to control the existence and length of the queue of allocate requests. If you enable the exit, the parameters from the connection definition are passed to your XZIQUE global user exit program, which can change the way in which these parameters are used.

The exit program also demonstrates how to control allocate requests for a particular modegroup, based on the same QUEUELIMIT and MAXQTIME parameters.

Overview of the sample exit program: The program uses the exit-specific parameters passed by CICS to determine the state of the connection, and to request the appropriate action, as follows:

1. The connection is operating normally; a queue may exist, but is of short length.

In this case, the exit program returns with UERCAQUE to indicate that CICS is to **queue the request**.

2. The response from the partner system is slower than the rate of requests demands, and the queue length has grown to the limit specified on the QUEUELIMIT parameter. The partner system is still operating normally, but is overloaded.

In this case, the exit program returns with UERCAPUR to indicate that CICS is to **purge the request**.

3. The queue has reached the limit specified by the QUEUELIMIT parameter, **and** requests that join the queue are expected to take longer to be satisfied than the time defined by the MAXQTIME parameter. (The estimated time for a request to complete is calculated by dividing the number of successful requests since the queue first formed by the time elapsed since it formed. These statistics are passed to the exit in the parameter list.)

These criteria are used to determine that the connection is not operating correctly, and that continued queuing of tasks is not helpful. In this case:

- The exit returns with UERCAKLL requesting CICS to **purge all queued** user requests from the connection. The SYSIDERR condition is returned to the application program.
 - CICS issues message DFHZC2300 to warn that a connection is not performing as expected.
4. The queue has been purged as a result of a previous invocation of the global user exit program, there are still no free sessions, and the request is about to be queued.

In this case, the exit program returns with UERCAPUR to indicate that CICS is to **purge the request**. This also leaves the UEPRC8 flag set.

5. The queue has been purged as a result of a previous invocation of the global user exit program. A new allocate request has been received and is about to be allocated because a session has become free.

CICS invokes the exit program to enable it to indicate that normal processing can continue.

In this case, the exit program returns with UERCNORM to indicate that CICS is to **continue processing normally**. This also causes the UEPRC8 flag to be unset following this invocation, and CICS to issue message DFHZC2301.

The sample program also monitors the length of queues for modegroup-specific allocate requests and controls these—in the same way as the queue for the whole connection—using the QUEUELIMIT parameter and MAXQTIME parameters.

If both UEPRC8 and UEPRC12 are set, UERCNORM is required twice to resume normal operation. The UEPRC8 condition is reset first in this case.

Extensions to the sample program: The sample exit program does not attempt to control the queue length, or detect poor response for a particular modegroup differently from the whole connection. This kind of enhancement is something you might want to add to your own exit program if your applications request specific modegroups via the allocate command (or via a transaction profile) and you think it would be useful to control the modegroups individually.

| You can also use more complex decisions (such as adding time delays to lessen
| the risk of false diagnosis) to decide when to issue the return codes that purge the
| queue, and allow queuing to restart.

XRF request-processing program exit XXRSTAT

XXRSTAT enables you to decide whether to terminate CICS when either of the following occurs:

- CICS is notified of a VTAM failure by the TPEND exit.
- A **predatory takeover**. A “predatory takeover” can occur, if you are using VTAM Release 3.4.0 or above, and a VTAM application with the same APPLID as that of the executing CICS system assumes control of all the sessions of the executing CICS system.

XXRSTAT gives you the choice of allowing the system which has suffered the takeover to continue or to terminate.

To avoid potential integrity exposures, CICS default action after a predatory takeover is to terminate without a dump. If you want CICS to terminate with a dump, your exit program should return UERCABDU. CICS terminates with the abend code specified by your exit program.

If you want CICS to continue after a predatory takeover, your exit program must return UERCCOIG. Message DFHZC0101 is issued and CICS continues processing without VTAM support. The predatory application assumes control of all VTAM sessions (all TCAM sessions remain bound to CICS).

Warning: Allowing CICS to continue after a predatory takeover could cause integrity problems and is not recommended. You are also recommended to use RACF to protect your CICS APPLIDs.

For more information about this exit and the circumstances in which you can use it, refer to the *CICS/ESA 3.3 XRF Guide*.

Table 96 (Page 1 of 2). Exit XXRSTAT											
When invoked	After either of the following: <ul style="list-style-type: none"> • CICS is notified of a VTAM failure by the TPEND exit. • A predatory takeover. 										
Exit-specific parameters	<table> <tr> <td>UEPERRA</td> <td>Address of parameter list containing:</td> </tr> <tr> <td>UEPGAPLD</td> <td>Address of the 8-byte generic applid</td> </tr> <tr> <td>UEPSAPLD</td> <td>Address of the 8-byte specific applid</td> </tr> <tr> <td>UEPDOMID</td> <td>Address of the 4-byte domain ID</td> </tr> <tr> <td>UEPERRID</td> <td>Address of the 4-byte error ID.</td> </tr> </table> <p>Notes:</p> <ol style="list-style-type: none"> 1. No DSECT is provided for the above parameter list. You need to code your own DSECT to access the named fields. 2. When VTAM has failed, the domain ID is 'ZC ' (uppercase Z, uppercase C, and two blanks), and the error ID is the character string '3443'. 	UEPERRA	Address of parameter list containing:	UEPGAPLD	Address of the 8-byte generic applid	UEPSAPLD	Address of the 8-byte specific applid	UEPDOMID	Address of the 4-byte domain ID	UEPERRID	Address of the 4-byte error ID.
UEPERRA	Address of parameter list containing:										
UEPGAPLD	Address of the 8-byte generic applid										
UEPSAPLD	Address of the 8-byte specific applid										
UEPDOMID	Address of the 4-byte domain ID										
UEPERRID	Address of the 4-byte error ID.										

Table 96 (Page 2 of 2). Exit XXRSTAT

Return codes	UERCNORM	Take the system action. The system action depends on the reason why the exit was invoked: <ul style="list-style-type: none"> • For XRF, in the event of a VTAM failure: CICS continues processing as if the exit program had not been invoked. • For VTAM persistent sessions, in the event of a predatory takeover: CICS abends without a dump.
	UERCCOIG	Ignore.
	UERCABNO	Abend CICS without a dump.
	UERCABDU	Abend CICS with a dump.
	UERCPURG	Task purged during XPI call.
XPI calls	All can be used.	

XRF request-processing program exit

Chapter 2. Task-related user exit programs

General-use programming interface

This chapter describes a special kind of user exit called a **task-related user exit**. A task-related user exit allows you to write your own program to access a resource, such as a database, that would not otherwise be available to your CICS system. Such a resource is known as a non-CICS resource. The exit is said to be task-related because it becomes part of the task that invoked it and because, unlike a global user exit, it is not associated with an exit point. You do not have to use any of the task-related user exits, but you can use them to extend and customize the function of your CICS system according to your own requirements.

The most common use of a task-related user exit is to communicate with a resource manager external to CICS, for example, a file or database manager. The CICS interface modules that handle the communication between the task-related user exit and the resource manager are usually referred to as the resource manager interface (RMI) or the task-related user exit interface.

Introduction to the task-related user exit mechanism (the adapter)

The task-related user exit mechanism is known as an **adapter** because it provides the connection between an application program that needs to access a non-CICS resource and the manager of that resource. Figure 4 on page 202 illustrates the adapter concept.

The adapter is made up of three or more locally-written programs. These are a “stub” program, a task-related user exit program, and one or more administration routines or programs.

The **stub program** intercepts a request (for example, to access data held on an external database manager) issued by the calling application program. The stub can be used to resolve a locally-defined high-level language command into a task-related user exit macro call, DFHRMCAL, which then causes CICS to pass control to the task-related user exit program.

The **task-related user exit program** translates commands for accessing a non-CICS resource into a form acceptable to the resource manager. The program must be written in assembler language, and can reside above or below the 16MB line. For more guidance information about addressing and residency modes, refer to “Addressing-mode implications” on page 217. The program must not alter the contents of any access registers. It is executed in response to a specific application program request, for example, to read data from an external database. In this instance, it may be passed application data, such as a search argument for a required record. Responses from the resource manager are passed back to the calling program by the task-related user exit program.

the stub program

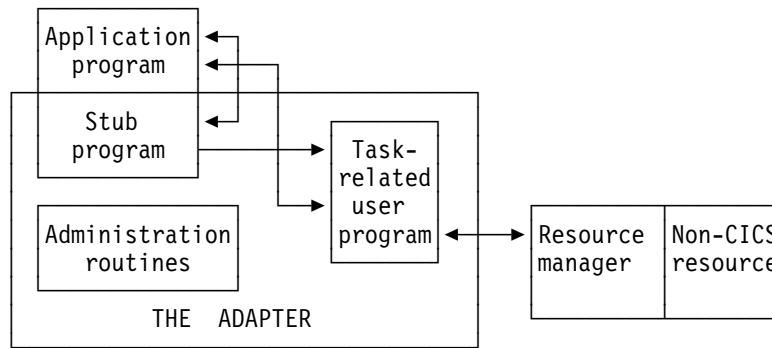


Figure 4. The adapter concept

The task-related user exit program is provided with a parameter list (DFHUEPAR) by the CICS management module that handles task-related user exits. This parameter list gives the task-related user exit access to information such as the addresses and sizes of its own work areas.

The task-related user exit program may be invoked by the CICS task manager and the CICS syncpoint manager, as well as by an application program. It may also be invoked at CICS termination or by the Execution Diagnostic Facility (EDF). The parameter list serves to distinguish between these various callers, and gives access to a register save area containing the caller's registers.

The **administration routines** contain the EXEC CICS ENABLE and DISABLE commands that you use to install and withdraw the task-related user exit program. The administration routines may also contain commands to retrieve information about one of the exit program's work areas (the EXEC CICS EXTRACT EXIT command), and to resolve any inconsistency between CICS and a non-CICS resource manager after a system failure (the EXEC CICS RESYNC command). For programming information about the EXEC CICS RESYNC command, refer to the *CICS/ESA System Programming Reference* manual.

The remainder of this chapter discusses each of these parts of the adapter in turn.

The stub program

The purpose of the stub program is to shield your application programmers from the mechanics of non-CICS resource managers. It is written in assembler language. After assembly, the stub is link-edited to each application program that wants to use it. See Figure 5 on page 203.

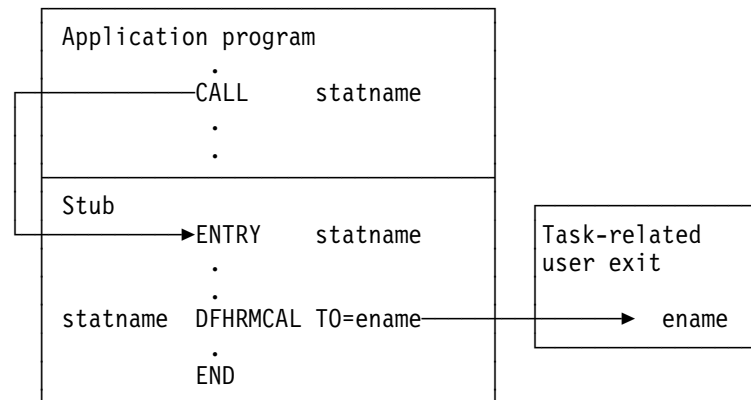


Figure 5. The stub concept

statname is a label that can be referenced externally. It should conform to the requirements of an assembler-language ENTRY statement, and typically resolves a V-type address constant, or the target of a high-level language CALL. A single stub may contain several such labels.

ename is the entry name (specified on the EXEC CICS ENABLE command) of the task-related user exit program that you want to handle resource manager requests.

You can define high-level language commands for your programmers to use when they want to access a non-CICS resource. If you do this, you must use a translator to convert a locally-defined high-level language command into a conventional CALL to the required entry point of the stub program. Alternatively, the application program can issue a CALL naming the stub entry point, as shown in Figure 5. For example, to read a record from a non-CICS resource, an application program can use the COBOL statement:

```
CALL 'XYZ' USING PARM1 PARM2...
```

XYZ is an entry point (the statname) in your stub program. The stub converts the command into a macro call (DFHRMCAL) to the task-related user exit program, specified in the TO= operand. Return from the task-related user exit program is to the calling application program, not to the stub program.

The application can use a parameter to determine whether the resource manager was called. For example, if the application sets a parameter to zero and the resource manager sets it to nonzero, the parameter value on return indicates whether the resource manager was invoked.

Returning control to the application program: If you specify RTNABND=YES in the DFHRMCAL macro, control returns to the application program when the task-related user exit is not available, for example, because it is not enabled or started. Note that for assembler-language application programs, a negative value in register 15 signals to the application program that control has returned because the exit is not available. The task-related user exit program can use positive values (including zero) in register 15 to pass resource manager response codes to the application program.

If you do not specify RTNABND=YES and the task-related user exit is not available, the application program terminates abnormally with the abend code 'AEY9'.

the task-related user exit program

Task-related user exits and EDF: When a task-related user exit (TRUE) is invoked for a call to a non-CICS resource manager from an application that is being monitored by EDF, EDF's default action is to display the parameters that are addressed by the parameter list passed by the DFHRMCAL macro. However, the parameter list can be transformed into a more meaningful display by the TRUE itself. This is done by specifying FORMATEDF on the EXEC CICS ENABLE command that enables the TRUE. The latter is then invoked several times, before and after the invocation to satisfy the call to the resource manager, to format the data to be displayed by EDF and to deal with any changes made by the user to the data on the EDF screen.

For more information about how to format screens for EDF, refer to "CICS EDF build parameters" on page 212 and "Using EDF with your task-related user exit program" on page 228.

If a task-related user exit program contains EXEC CICS commands, EDF may be useful in debugging the TRUE itself. If you want EDF to display commands from the TRUE, you must specify the EDF option when the TRUE program is translated. The standard EDF screens for the CICS commands are then displayed between the "About to Execute" and "Command Execution Complete" screens for the call to the resource manager. However, as EDF is primarily an application debugging tool and the CICS commands within the TRUE would not generally be of interest to the application programmer, the TRUE program is normally translated with the "NOEDF" option; in this case, screens for CICS commands within the TRUE are suppressed.

Note: If you specify SUPPEDF=YES on the DFHRMCAL macro, the "About to Execute" and "Command Execution Complete" screens relating to DFHRMCAL's invocation of the TRUE are suppressed; in other words, DFHRMCAL becomes "invisible" to EDF. (Specifying SUPPEDF=YES has no effect in determining whether EDF displays EXEC CICS commands within the TRUE—the factors governing this are as described above—but it **does** suppress the display of parameters passed to the TRUE.)

The task-related user exit program

The main function of the task-related user exit program is to translate the calling program's parameters into a form acceptable to your non-CICS resource manager, and then to pass control to the resource manager. You therefore need to be familiar with your resource manager's syntax requirements. The calling program's parameters are described on page 208.

This section describes the user exit parameter lists, the schedule flag word, which is used by the exit program to register its need to be invoked by CICS management services, and register-handling in the task-related user exit program. This section also discusses the use of the CICS syncpoint manager and the CICS task manager.

User exit parameter lists

When a task-related user exit is invoked, the CICS management module that handles task-related user exits provides the exit with a parameter list. The address of this parameter list is passed in register 1. The list contains the following information:

- The identity of the caller
- Addresses and sizes of any work areas that are available to the task-related user exit program
- The address of the register save area of the caller
- The address of an EXEC interface block (EIB) that is for use by the task-related user exit program during this invocation
- The address of the identifier of the current unit of recovery
- The address of the schedule flag word
- The address of the kernel stack entry
- The address of the APPC unit of work (UOW) identifier
- The address of the user security block flag
- The address of the user security block
- The address of the resource manager qualifier name
- The address of the resource manager's "single-update" and "read-only" indicator byte
- The address of the caller's AMODE indicator byte
- The address of the application's DATALOC and TASKDATAKEY indicator byte.

To enable your exit program to access this parameter list, you must include in it the macro:

```
DFHUEXIT TYPE=RM
```

The DFHUEXIT TYPE=RM macro causes the assembler to create the storage definitions (DSECTs) DFHUEPAR and DFHUERTR. If you want your task-related user exit to be able to format screens for EDF, you must include in it the macro:

```
DFHUEXIT TYPE=RM,DSECT=EDF
```

This causes the assembler to create the UEPEDFRM DSECT, which is described in "CICS EDF build parameters" on page 212. All of the user exit parameter lists are summarized in Figure 7 on page 214.

The format and the purpose of these definitions are described below.

DFHUEPAR

DFHUEPAR gives you the following symbolic names for address parameters:

UEPEXN Address of the function definition, which tells the task-related user exit program why it is being called. See "DFHUERTR (the function definition)" on page 208 for more details.

the task-related user exit program

UEPGAA	Address of the global work area requested in the EXEC CICS ENABLE command. The global work area is described on page 219. CICS initializes this work area to X'00' when the task-related user exit program is enabled.
UEPGAL	Address of a halfword containing the length (binary value) of the global work area.
UEPTCA	This field is retained for historical reasons. It should not be referenced by your exit program.
UEPCSA	This field is retained for historical reasons. It should not be referenced by your exit program.
UEPHMSA	Address of the register save area (RSA) of the caller. It is an 18-word save area, with the contents of registers 14 through 12 stored in the fourth and subsequent words. Its fifth word, representing the calling program's register 15, is cleared by CICS before the task-related user exit program is invoked, so that it can be used to convey response codes from the resource manager to the calling program. For this reason you cannot use register 15 to send data to the task-related user exit program. The seventh word of the save area contains the caller's register 1. Register 1 addresses the caller's parameter list if the exit program is being invoked by the CICS task manager or the CICS syncpoint manager, by EDF, or at CICS termination. When the caller is an application program, the contents of register 1 are determined by the linkage conventions of the adapter's language interface.
UEPTAA	Address of the local work area requested in the EXEC CICS ENABLE command. The local work area is described on page 219. CICS initializes the work area to X'00' throughout on first acquiring the area; that is, when the task first invokes the task-related user exit program.
UEPTAL	Address of a halfword containing the binary length of the local work area.
UEPEIB	Address of the EXEC interface block (EIB) created by CICS for the task-related user exit program. The EIB exists only for the duration of the call and it allows the task-related user exit program to request CICS services through the command-level interface. This is not the same EIB that is available to the calling program, so you cannot access the calling program's environment other than by UEPHMSA (see above), which provides the address of the calling program's register save area (RSA).
UEPURID	Address of CICS unit of recovery identifier. This field contains the 8-byte date and time value that is generated by an STCK instruction, and it identifies the current logical unit of work.
UEPFLAGS	Address of the schedule flag word. This is a fullword that the task-related user exit program uses to register its need for CICS management programs' services. For more information, see "The schedule flag word" on page 214.
UEPRMSTK	Address of the kernel stack entry.
UEPUOWDS	Address of the APPC unit of work (UOW) identifier.

UEPSECFLG Address of the user security flag. The user security flag is a 1-byte field that can take the following values:

UEPNOSEC (X'80')

Security is not active for this CICS system.

UEPSEC (X'20')

Security is active for this CICS system. Only in this case is the address of the “user security block” set.

UEPSECBLK Address of a fullword that addresses the “user security block”—that is, the ACEE.

UEPRMQUA Address of an 8-byte field into which the task-related user exit can move the qualifier name of the resource manager on each API request. Where different resource manager qualifiers are returned on the responses to various API requests within an LUW, it is the resource manager qualifier returned on the final API request immediately before a prepare or backout invocation that is used when recording any in-doubt information. For programming information about the purpose of the qualifier name, refer to the *CICS/ESA System Programming Reference* manual.

UEPCALAM Address of caller’s AMODE indication byte.

X'80' Indicates that the original caller was in AMODE 31. If the top bit is not set, then the caller was in AMODE 24.

UEPSYNCA Address of the single-update and read-only indication byte. This field contains flags that your exit program can set to indicate that the resource manager “understands” the single-update protocol, and to record the status of the current logical unit of work (LUW). See “Increasing efficiency – single-update and read-only protocols” on page 220.

UEPSUPDR (X'80')

The resource manager understands the single-update protocol. That is, your exit program can instruct the resource manager to perform a single-phase commit, in appropriate circumstances.

UEPREADO (X'40')

The resource manager understands the read-only protocol, and has been in read-only mode for this LUW so far. (If this flag is not set, it means either that the LUW contains updates for this resource manager, or that the LUW may be read-only but the resource manager does not understand the read-only protocol.)

UEPTIND Address of a 1-byte field containing indicator bits. The indicators are represented by two symbolic values, UEPTANY and UEPTCICS, which you can test to determine whether data locations can be above or below the 16MB line, and whether the application’s storage is in CICS-key or user-key storage.

UEPTANY (X'80')

The application can accept addresses above the 16MB line. If the symbolic value is not UEPTANY, the application must be returned an address below the 16MB line.

UEPTCICS (X'40')

The application's working storage and task life-time storage are in CICS-key storage (TASKDATAKEY=CICS). If the symbolic value is not UEPTCICS, the application's working storage and the task's life-time storage are in user-key storage (TASKDATAKEY=USER).

UEPPBTOK Address of the performance block token used for workload management, to enable resource managers to relate their own performance blocks for the work request with the original CICS performance block. For example, DBCTL and DB2 need to correlate the work they do on behalf of CICS with the originating CICS task, so that MVS workload manager can measure the performance of the whole CICS task.

DFHUERTR (the function definition)

The function definition identifies the caller of the task-related user exit program. The DSECT contains two symbolic definitions (fields).

UERTFGP A single byte that is set to X'00'. The zero setting shows that this is a task-related user exit invocation and that the parameter list therefore includes the fields UEPTAA, UEPTAL, UEPEIB, UEPURID, and UEPFLAGS.

UERTFID A single-byte identifier that shows whether this call has been made by an application program, the CICS syncpoint manager, the CICS task manager, or EDF, or whether this is a CICS termination call. It can have one of the following five settings:

- UERTAPPL** (X'02') The calling program is an application program.
- UERTSYNC** (X'04') The calling program is the syncpoint manager.
- UERTTASK** (X'08') The calling program is the CICS task manager.
- UERTCTER** (X'0A') This is a CICS termination call.
- UERTFEDF** (X'0C') The calling program is EDF.

It is important to know which type of program has made the call because it affects how the calling program's parameter list is interpreted by the task-related user exit program.

Caller parameter lists

In addition to the DSECTs DFHUEPAR and DFHUERTR, the inclusion of DFHUEXIT TYPE=RM in the task-related user exit program provides some field definitions that are specific to the caller of the task-related user exit. The calling program's parameter list is normally addressed by R1 in the calling program's RSA. This RSA is addressed by field UEPHMSA of DFHUEPAR. These parameters are described below.

Application program parameters: If the caller is an application program, the format and addressing of its parameter list are decided locally.

CICS syncpoint manager parameters: The CICS syncpoint manager's parameter list contains ten entries, although on most invocations only parameters 1 and 10 contain values. The operation bytes pointed to by parameters 1 and 10 contain flags which, when combined, form an operation code that tells the TRUE why it has been invoked.

Parameters 2 through 9 contain values only when the syncpoint manager makes a “Commit Unconditionally” or “Backout” call to the TRUE, for resynchronization purposes after a session or system failure. These extra parameters point to fields that identify the task, the transaction that started the task, the terminal from which it was initiated, the identity of the terminal operator, the date and time of the failing syncpoint, and (if there are no further units of recovery associated with the task) the next transaction code. Typically, you would use these values to create meaningful messages for resource recovery. They are presented explicitly because, after a system failure, the task driving the exit is not the task that originally scheduled the recoverable work. These additional parameters describe the **original** task’s environment and are accessed directly.

The full parameter list is as follows:

Parameter 1

The address of operation byte 1, which contains the following flags:

UERTPREP

(X'80') Prepare to commit (that is, perform the first phase of a two-phase commit).

UERTCOMM

(X'40') Commit unconditionally (perform the second phase of a two-phase commit).

UERTBACK

(X'20') Backout.

UERTDGCS

(X'10') Unit of recovery is lost to CICS cold start.

UERTDGNK

(X'08') Resource manager should not be in doubt about this unit of recovery.

UERTRSYN

(X'02') This syncpoint request was generated as the result of an EXEC CICS RESYNC command.

UERTLAST

(X'01') There are no further units of recovery associated with this task. Note that when this bit is **not** set, there may or may not be further units of recovery. For this reason, it is not recommended that you rely on this bit to signal end-of-task. You should instead schedule the CICS task manager to drive you at end-of-task by setting the task manager bit in the schedule flag word. If you do use UERTLAST to signal end-of-task, and if at that stage you can complete your clean-up process, you can set the task manager bit off in the schedule flag word when the clean-up process is finished, to avoid an unnecessary invocation by the CICS task manager.

The only **valid bit combinations** are those produced by combining one of UERTPREP, UERTCOMM, UERTBACK, UERTDGCS, and UERTDGNK with either UERTLAST or UERTRSYN, or both.

Your exit program should examine the flags set both in this byte and in operation byte 2 (see parameter 10), to determine what action is expected of it.

the task-related user exit program

Parameter 2

If not zero, the address of a 4-byte, packed-decimal field identifying the original task. But note that, on many invocations of the exit program, parameters 2 through 9 do not contain values. See note 1.

Parameter 3

Address of a 4-character field identifying the transaction that started the original task. See note 1.

Parameter 4

Address of a 4-character field identifying the terminal from which the original task was initiated. See note 1.

Parameter 5

Address of a 4-character field containing the identity of the terminal operator (OPID) who initiated the original task. See note 1.

Parameter 6

Address of a 4-byte, packed-decimal field containing the date of the failing syncpoint, in the format 00yyddd+. See note 1.

Parameter 7

Address of a 4-byte, packed-decimal field containing the time of the failing syncpoint, in the format 0hhmmss+. See note 1.

Parameter 8

Address of an 8-byte field containing the resource manager qualifier. See note 1.

Parameter 9

Address of a 4-character field containing the next transaction code. If the transaction ended with an EXEC CICS RETURN without specifying the next transaction code, the addressed field is set to nulls; otherwise, it is set to the value specified by the application. See note 2.

Parameter 10

The address of operation byte 2, which contains the following flags:

UERTONLY

(X'80') Perform a single-phase commit. (No recoverable resources other than those owned by the resource manager being invoked have been updated during the current LUW.)

UERTELUW

(X'40') Perform a single-phase commit. (The resource manager was in read-only mode throughout the current LUW.)

Your exit program should examine the flags set both in this byte and in operation byte 1 (see parameter 1), to determine what action is expected of it.

Notes:

1. Parameters 2 through 8 contain values only if the CICS syncpoint manager call is prompted by the issue of an EXEC CICS RESYNC command after a session or system failure, and operation byte 1 contains the bit settings UERTCOMM or UERTBACK. Otherwise, they are set to X'00' (hexadecimal zero). For programming information about the EXEC CICS RESYNC command and about the completion of the syncpointing procedure following a system failure, refer to the *CICS/ESA System Programming Reference* manual.

Note that parameters 2 through 8 describe the environment of the **original** task (not of the task that is currently driving the TRUE).

2. Unless the UERTLAST bit is set in operation byte 1, parameter 9 is a zero address. Although for a call prompted by an EXEC CICS RESYNC call, the UERTLAST bit will be set on, in this case the next transaction code does not apply and so Parameter 9 addresses a field set to nulls.

CICS task manager parameters: There are either one or two entries in the CICS task manager's parameter list, depending on the reason for the call to the TRUE: on start-of-task calls, the parameter list contains one entry, while on end-of-task calls, it contains two. Each entry consists of an address, and the end of the parameter list is indicated by the top bit of the address being set.

The significance of the parameters is as follows:

Parameter 1

The address of a single byte with bit definitions indicating the reason for the call:

- UERTSOTR** (X'40') Start of CICS task
- UERTEOTR** (X'80') End of CICS task

Parameter 2

This parameter is passed only on end-of-task calls. It is the address of a 4-character field which contains the next transaction code specified on the EXEC CICS RETURN command. If the transaction ends with an EXEC CICS RETURN without specifying a next transaction, this field is set to nulls.

The schedule flag word should be set during the start-of-task call if you want your task-related user exit program to be invoked unconditionally by the CICS syncpoint manager.

CICS termination manager parameters: All task-related user exit programs that have been enabled with the SHUTDOWN option of the EXEC CICS ENABLE command, and started, are invoked at CICS termination to allow them to do the clean-up processing that is appropriate to the type of termination. At CICS termination, the address of a one-byte termination code is passed to your exit program. The code may consist of any of the following bit settings:

- UERTCORD** (X'80') CICS orderly shutdown
- UERTCIMM** (X'40') CICS immediate shutdown
- UERTCABY** (X'20') CICS abend, retry possible, TCBs dispatchable
- UERTCABN** (X'10') CICS abend, retry not possible, TCBs dispatchable
- UERTOPCA** (X'01') CICS abend, retry not possible, TCBs not dispatchable.

For further information about shutdown TRUEs, see "Coding a program to be invoked at CICS termination" on page 224.

the task-related user exit program

CICS EDF build parameters: On EDF invocations, the address contained in register 1 of the calling program's RSA points to the UEPEDFRM DSECT. This contains the following fields:

UEPEDFR1 The address of the application's R1 parameter list.

UEPEDFFI The input flag byte. When a task-related user exit is invoked by EDF, UEPEDFFI can take the following bit settings:

UEPEDFRQ (X'80') "About to Execute" invocation.

UEPEDFRS (X'40') "Command Execution Complete" invocation.

UEPEDFRA (X'20') About to display command to EDF.

UEPEDFRC (X'10') Command has been displayed to EDF.

UEPEDFSC (X'08') EDF user has changed the screen.

UEPEDFWS (X'04') EDF user has changed working storage.

UEPEDFNO (X'01') EDF user has requested NOOP.

UEPEDFFO The output flag byte. If the task-related user exit requires, it can set the UEPEDFFO flag byte to indicate to EDF what action the task-related user exit wants EDF to take. It can take the following values:

UEPEDDFD (X'80') Take default CICS action. (EDF screen contains the uninterpreted caller's R1 parameter list.)

UEPEDFND (X'40') Do not display command to EDF.

UEPEDFRD (X'20') Redisplay command to EDF.

UEPEDFDL EDF screen attributes. These are for information only: the task-related user exit program cannot change these fields.

UEPEDFPS (halfword binary) Page size (number of lines).

UEPEDFLS (halfword binary) Line size.

UEPEDFMP (halfword binary) Maximum number of pages.

UEPEDFPA The address of the EDF display data parameter list, supplied by the task-related user exit. The display data parameter list is composed of alternating pairs of attribute-byte addresses and data-field addresses. Attribute bytes refer to the line of display data pointed to by the data-field addresses. The data field must be the same size as the value specified in UEPEDFLS. The display data is in the format shown in Figure 6 on page 213.

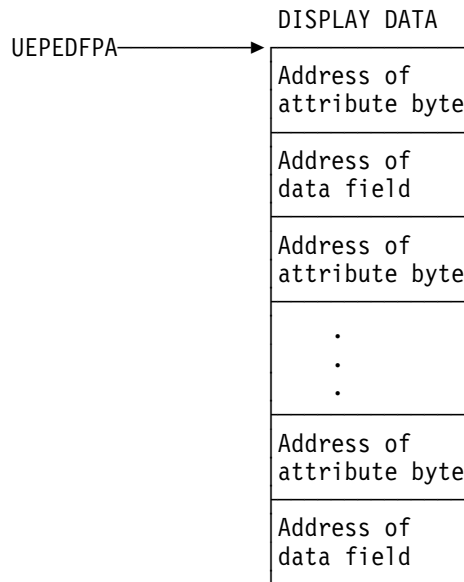


Figure 6. Display data parameter list

Notes:

1. CICS provides a list of named standard attribute bytes that you may want to use. These standard attribute bytes are contained within DFHBMSCA, which must be copied into your program. For programming information, including a list of the attribute bytes and their meanings, refer to the *CICS/ESA Application Programming Reference* manual.
2. The high-order bit must be set **on** in the last address, to indicate to EDF that this is the last address.

Summary of the task-related user exit parameter lists

Figure 7 on page 214 shows, in diagrammatic form, the relationships between the parameter lists that are discussed in the preceding sections.

the task-related user exit program

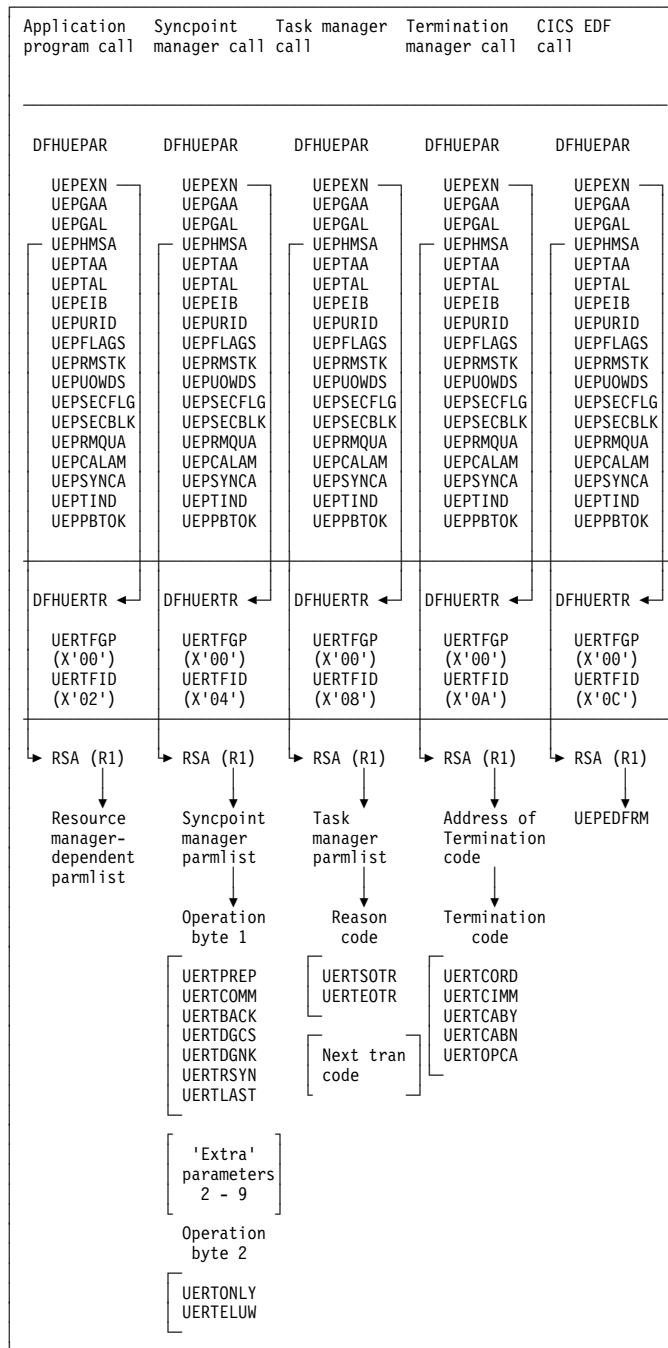


Figure 7. Task-related user exit parameter lists

The schedule flag word

The schedule flag word is a fullword indicator that the task-related user exit program uses to control its own invocation. It is also used by CICS to schedule the first invocation of a task-related user exit program. The schedule flag word is accessed by the address parameter UEPFLAGS of DFHUEPAR. There is a unique schedule flag word for each association between a CICS task and the ENTRYNAME specified when a task-related user exit program is enabled.

The default setting of the schedule flag word is for application program requests (that is, the last two bytes are set to X'0004').

The format of the schedule flag word is shown in Table 97.

Byte	Setting	Comments
0	—	Reserved.
1	—	Reserved.
2		
UEFDFEDF	UEFMFEDF (X'10')	Bit mask for EDF invocation.
UEFDTASK	UEFMTASK (X'01')	Bit mask for task manager exit.
3		
UEFDSYNC	UEFMSYNC (X'10')	Bit mask for syncpoint manager exit.
UEFDAPPL	UEFMAPPL (X'04')	Bit mask for application program exit.

The bit settings of the schedule flag word show which programs invoke your task-related user exit program. For example, if an exit program is to be invoked by the CICS task manager, the CICS syncpoint manager, **and** an application program, then the last two bytes of the schedule flag word are set to X'0114'. If an exit program is to be called by the CICS task manager and an application program only, the last two bytes of the flag word are set to X'0104'. Before the exit program is first called by a task, CICS sets the API flag bit on.

Before returning from any call, the task-related user exit can change the bit settings of the flag word to register its need to be invoked by a different CICS management service, or to register lack of interest in a service by setting the relevant flag bit to zero.

For example, a task-related user exit may be called by an application program that needs to access a non-CICS recoverable resource. When the exit program is first called, the API bit is set on by CICS. If the calling program then issues a request to update a record, the exit program sets the syncpoint manager bit on in the schedule flag word. When the calling application program subsequently issues a syncpoint command, or when end-of-task is reached, the CICS syncpoint manager calls the exit program.

Note: CICS sets the syncpoint manager bit off after every call to the syncpoint manager. This is to avoid the CICS syncpoint manager invoking the task-related user exit program for a unit of recovery during which the exit program did no recoverable work. The syncpoint manager bit must therefore be set on whenever the exit program performs any recoverable work.

If you set the task manager bit in the schedule flag word on, CICS invokes your task-related exit program at the end of this task. (Note that, if you want your exit program to be called at the **start** as well as at the end of a task, you must specify TASKSTART on the EXEC CICS ENABLE command for the TRUE. This causes the TRUE to be invoked at the start and end of **every** task.)

the task-related user exit program

If the last two bytes of the schedule flag word are set to X'1000', this indicates that the task-related user exit is interested in being invoked by EDF to format requests for display. This schedule flag bit UEFD FEDF is set **on** either by the EXEC CICS ENABLE FORMATEDF command, or by the task-related user exit. Unlike other schedule flag bits, there are restrictions on when the task-related user exit can register a lack of interest in EDF (that is, restrictions on when UEFD FEDF can be set off). Once a task-related user exit has formatted the initial screen for EDF to display on "About to Execute" or "Command Execution Complete", CICS does not allow it to set the EDF bit UEFD FEDF off until the screen build cycle is complete.

Register handling in the task-related user exit program

In this section, two sets of registers are discussed:

1. The registers belonging to the CICS management module that handles task-related user exits. These are referred to as the **CICS registers**.
2. The registers belonging to the calling program and that are addressed by parameter UEPHMSA of DFHUEPAR. These are referred to as the **calling program's registers**.

Saving CICS registers

Your task-related user exit program should begin by saving the contents of the CICS registers. Register 13 addresses an 18-word area into whose 4th and subsequent words your exit program should store registers 14 through 12. Three of the saved values have significance, as follows:

- The saved contents of register 14 contain the address within CICS to which the task-related user exit program returns control.
- The saved contents of register 15 contain the address at which the task-related user exit program has just been entered.
- The saved contents of register 1 address the parameter list (DFHUEPAR) that is provided by CICS for the task-related user exit program.

Note: As a general rule, if you fail to understand the origin or the purpose of a call, you should:

1. Restore any registers that you have used to the state they were in on entry to your code
2. Return to the address contained in CICS register 14.

The calling program's registers

The calling program's registers are stored at the address specified by UEPHMSA of DFHUEPAR. Where the calling program is a CICS management program, for example the syncpoint manager, the only caller registers that have significance are registers 1 and 15. Register 1 addresses the calling program's parameter list. CICS sets the calling program's register 15 to zero before the task-related user exit program is invoked. The calling program's register 15 can sometimes be used to pass responses back to the calling program from the task-related user exit program, depending on the identity of the caller. If the calling program is a CICS management program, and the register is still zero on return, CICS assumes that its call was not understood. If the calling program is an application program, the significance of register settings on return are either described in your resource manager's documentation, or defined locally.

Addressing-mode implications

The task-related user exit is invoked in the AMODE of the caller, unless the exit has been enabled with the LINKEDITMODE option of the EXEC CICS ENABLE command. This option enables the task-related user exit in its link-edit AMODE. Therefore, if the TRUE has been link-edited AMODE 31 and is enabled with the LINKEDITMODE option, it can be placed above the 16MB line. For programming information about the LINKEDITMODE option of the EXEC CICS ENABLE command, refer to the *CICS/ESA System Programming Reference* manual.

Important

You should avoid the use of the LINKEDITMODE option where the TRUE has been link-edited AMODE 24. This combination forces the TRUE *always* to run AMODE 24, which is unwise because:

- An AMODE 24 TRUE cannot be invoked from a transaction running with TASKDATALOC(ANY). This results in an 'AEZB' abend.
- Enabling an AMODE 24 TRUE for task start causes CICS to force all transactions to run with TASKDATALOC(BELOW).
- On a CICS termination call, CICS ignores LINKEDITMODE and invokes the TRUE in AMODE 31, if it detects that the TCA it is running under is above the 16MB line. (This is because, for some types of termination, such as a cancel, the TCA under which the TRUE will run is not predetermined.)

It is recommended that TRUEs are:

- Written so that they can always run AMODE 31
- Link-edited AMODE 31
- Enabled with the LINKEDITMODE option.

If the task-related user exit has not been enabled with the LINKEDITMODE option of EXEC CICS ENABLE, it is invoked in the AMODE of the caller. For example, in the case of an application request, if the application is AMODE 24 at the time of the DFHRMCAL, the task-related user exit is invoked in AMODE 24. For this reason, task-related user exits which have been enabled without the LINKEDITMODE option must reside below the 16MB line.

Exit programs and the CICS storage protection facility

When you are running CICS with the storage protection facility, there are two points you need to consider for task-related user exits:

1. The execution key in which your task-related user exit programs run
2. The storage key of data storage obtained for your exit programs.

Execution key for task-related user exit programs

When you are running with storage protection active, CICS always invokes task-related user exit programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to the TRUE. However, if a task-related user exit program itself passes control to another program (via a link or transfer-control command), the program thus invoked executes according to the execution key (EXECKEY) defined in its program resource definition.

the task-related user exit program

Important

You are strongly recommended to specify EXECKEY(CICS) when defining both task-related user exit programs, and programs to which an exit program passes control.

Data storage key for task-related user exit programs

The storage key of storage used by task-related user exit programs depends on how the storage is obtained:

- Global or local work areas specified when an exit program is enabled, are always in CICS key.
- Any working storage obtained for the exit program is in the key set by the TASKDATAKEY of the transaction under which the exit program is invoked.
- Task-related user exit programs can use EXEC CICS commands to obtain storage by issuing:
 - Explicit EXEC CICS GETMAIN commands
 - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

The default storage key for storage obtained by EXEC CICS commands is
CICSDATAKEY. However, on an EXEC CICS GETMAIN command, the exit
program can override the CICSDATAKEY by specifying USERDATAKEY.

Recursion within a task-related user exit program

The task-related user exit has the ability to invoke itself recursively. It can do this, for example, by issuing a DFHRMCAL call to its own entry name (as specified on the EXEC CICS ENABLE command). It can also be entered recursively if it performs an EXEC CICS SYNCPOINT when it is interested in SYNCPOINT invocations.

Using CICS services in your task-related user exit program

You might find some CICS services useful in your exit program. These can be invoked using EXEC CICS commands. However, you should take note of the following:

- If your program is invoked because of a CICS abend, it must not use any CICS services. See “Coding a program to be invoked at CICS termination” on page 224.
- DFHEIENT and DFHEIRET must be in your program. *But see the note about not using DFHEIENT in abend invocations, on page 225.* For further details of the DFHEIENT and DFHEIRET macros, see the *CICS/ESA Application Programming Reference* manual.
- If your exit program entry point is immediately followed by an occurrence of a DFHEIENT macro, inserted either implicitly by CICS or explicitly in the program, then the expansion of the DFHEIENT macro stores incorrect values at DFHEIBP and DFHEICAP. Your code can subsequently correct this by copying UEPEIB into DFHEIBP, reloading the EIB base register (DFHEIBR) from UEPEIB, and setting DFHEICAP to X'80000000'. For example,

```

TESTPROG DFHEIENT CODEREG=2,EIBREG=11,DATAREG=10
          USING DFHUEPAR,1
          MVC   DFHEIBP,UEPEIB           Get correct EIB address
          L     DFHEIBR,UEPEIB           Reload EIB base register
          MVC   DFHEICAP,=X'80000000'
```

+

Note that the entry point of a program does not have to be at the start of the program and can be positioned after the DFHEIENT macro.

- The DFHEIENT macro allocates dynamic storage to be mapped by the DFHEISTG DSECT. You must return to CICS by means of the DFHEIRET macro, which frees the dynamic storage.
- Command-level calls use registers 0, 1, 14, and 15.
- Do not issue a syncpoint in start-of-task, end-of-task, or syncpoint invocations.
- On each invocation of a task-related user exit program, a new EXEC environment is created, even when the program is being invoked from the same task. This means that CICS operations, such as browse of a resource definition table, cannot be continued from one invocation of the exit program to the next.

Work areas

When you use the EXEC CICS ENABLE command to identify a task-related user exit program to CICS, you may specify that the program must have access to one local and one global work area. The EXEC CICS ENABLE command allows you to specify the size, in bytes, of the work areas to be acquired for your task-related user exit program. CICS acquires storage for the areas and initializes pointers to them. The user exit parameter list, DFHUEPAR, gives you access to the pointers. For more information, see the description of DFHUEPAR under “User exit parameter lists” on page 205.

The global work area

A global work area is associated with an exit program. Whenever the exit program is invoked, it has access to the area through the parameter UEPGAA of DFHUEPAR. The global work area may be shared by a number of exit programs. You must have specified the size of the global work area using the GALENGTH parameter or the GAENTRYNAME parameter of the EXEC CICS ENABLE command.

The local work area

A local work area is associated with a single task and lasts only for the duration of the task. It is for the use of a single task-related user exit program. It can be thought of as a logical extension to the transaction work area (TWA, TWACOB) that is exclusively for the exit program’s use. It is specified using the TALENGTH option of the EXEC CICS ENABLE command and is accessed using the UEPTAA parameter of DFHUEPAR.

Coding a program to be invoked by the CICS syncpoint manager

All task-related user exit programs can be invoked by the CICS syncpoint manager. An exit program must “schedule” the syncpoint manager by setting the syncpoint manager bit-mask in the schedule flag word (addressed by the UEPFLAGS parameter). The bit-mask must be set after every piece of recoverable work to ensure that the CICS syncpoint manager calls the exit program during syncpoint

the task-related user exit program

processing. (The identification of the current unit of recovery—or logical unit of work—is addressed by the 8-byte field UEPURID. This is available on all invocations of your exit program in which recoverable actions are possible, for example, application calls and subsequent syncpoint manager calls.)

Increasing efficiency – single-update and read-only protocols

If your resource manager is capable of performing a single-phase commit, you can increase the efficiency of your system by means of CICS single-update and read-only protocols.

Single-update protocol: Many CICS transactions use only one external resource manager. In this case, a single-phase commit is in order. The benefits of a single-phase commit are:

- The resource manager can reduce from two to one the number of log forces required for transactions.
- The number of transaction-related log records written by CICS is reduced.
- A path length reduction is achieved, because the TRUE is invoked only once at the syncpoint, rather than twice.

To take advantage of these benefits, your task-related user exit program must indicate to CICS that the resource manager understands the single-update protocol, and that it (the TRUE) can process a syncpoint call to perform a single-phase commit. It indicates this by setting the UEPSUPDR flag in the field pointed to by UEPSYNCA in the DFHUEPAR parameter list. It must do this every time it sets the syncpoint manager bit in the schedule flag word.

If the exit program has set the UEPSUPDR flag, then, when the syncpoint manager next invokes the TRUE, it informs it whether the resource manager is the only one to have updated resources in the current LUW. It does this by means of the UERTONLY bit (in operation byte 2 of the syncpoint manager's parameter list); if this is set on, then the resource manager can be asked to perform a single-phase commit.

Read-only protocol: Similar gains in efficiency can be made if the resource manager is in read-only mode throughout the current LUW. Again, a single-phase commit is in order. To benefit, the resource manager must return to the TRUE a flag indicating whether the LUW is read-only or not. The flag may show either the "history" of the LUW so far (for example, so far it is read-only), or simply whether the current request is read-only. In turn, the TRUE must update the UEPREADO flag in the DFHUEPAR parameter list with the history of the LUW so far. That is, it must set UEPREADO initially, but unset it as soon as the LUW contains updates. (Once UEPREADO has been unset, CICS ignores any subsequent setting of the flag during the current LUW, and treats the LUW as containing updates.)

At the end of the LUW, if the UEPREADO flag is still set, the syncpoint manager invokes the TRUE with instructions to issue a single-phase commit to the resource manager (by setting the UERTELUW bit on).

Return codes

When a task-related user exit program is invoked by the CICS syncpoint manager, the return codes it is able to set depend on the reason for the invocation. Table 98 shows the relationship between the request flags in the syncpoint manager's parameter list and the TRUE return codes. (The CICS syncpoint manager parameters are described on page 208.)

Table 98. Valid return codes for a TRUE invoked by the CICS syncpoint manager

Request-type	Return codes	Meaning
UERTPREP	UERFPREP	Phase 1 of 2-phase commit successful
	UERFBACK	Phase 1 of 2-phase commit unsuccessful
UERTCOMM	UERFDONE	Phase 2 of 2-phase commit successful
	UERFHOLD	Phase 2 of 2-phase commit unsuccessful
UERTBACK	UERFDONE	Backout successful
	UERFHOLD	Backout unsuccessful
UERTONLY	UERFOK	Single-phase commit successful
	UERFBOUT	Single-phase commit failed and backed out
UERTELUW	None	Not applicable

What is expected of your resource manager

If every request from the syncpoint manager prompts a meaningful response from the resource manager, CICS ensures that changes to recoverable resources (such as databases) can be synchronized. That is, either all the changes take effect or all are backed out, even across system failures.

Limitations

Do not update a recoverable CICS resource during a syncpoint call because any changes will not be seen by the CICS syncpoint manager.

Sample code for a TRUE invoked by the CICS syncpoint manager

The pseudocode given in Figure 8 on page 222 is only an example. It is not complete, and includes only some of the conditions that a task-related user exit invoked at a syncpoint might be required to check.

#

the task-related user exit program

```
if UERTFID = UERTSYNC then      /* Caller is CICS syncpoint manager */
select;                          /* Type of syncpoint manager request */
  when (UERTONLY)                /* ONLY resource manager */
    invoke RM for single-phase commit /* Single-phase commit */
    if RM single-phase commit succeeded then
      give CICS syncpoint manager 'YES' vote (UERFOK)
    else
      /* Single-phase commit failed */
      /* If RM completed backout */
      if RM single-phase commit failed and backed out
        give CICS syncpoint manager 'NO' vote (UERFBOUT)
      else
        /* Don't know what happened */
        put out message and issue transaction abend
      endif
    endif
  when (UERTEL UW)                /* RM read-only for current LUW */
    invoke RM for single-phase commit /* Single-phase commit */
  when (UERTPREP)                /* Not ONLY resource manager, nor read-only */
    invoke RM for PREPARE /* Prepare - phase 1 of 2-phase commit */
    select (resource manager vote)
      when (YES)                  /* Phase 1 completed */
        give CICS syncpoint manager 'YES' vote (UERFPREP)
      otherwise
        give CICS syncpoint manager 'NO' vote (UERFBACK)
    endselect
  when (UERTCOMM)                /* Commit - phase 2 of 2-phase commit */
    invoke RM for commit phase 2
    if RM commit succeeded then
      tell CICS sync manager OK (UERFDONE)
    else
      tell CICS sync manager remember could not commit (UERFHOLD)
    endif
  when (UERTBACK)                /* Backout request */
    invoke RM for backout
    if RM backout succeeded then
      tell CICS sync manager OK (UERFDONE)
    else
      tell CICS sync manager remember could not backout (UERFHOLD)
    endif
endselect
endif
```

Figure 8. Sample pseudocode for a task-related user exit program to be invoked by the CICS syncpoint manager

As described in “Increasing efficiency – single-update and read-only protocols” on page 220, if the UERTONLY bit is set (indicating that the resource manager is the only one to have updated resources) the exit program should cause the resource manager to perform a single-phase commit. If the commit is successful, the exit program should return ‘UERFOK’ in register 15; if not, it should return ‘UERFBOUT’, meaning that the commit was unsuccessful and the resources were backed out. If the exit program is unsure about the outcome of a single-phase commit, it must abend the task, having saved or displayed any diagnostic information that it considers necessary.

Note that “register 15” in this section refers to the syncpoint manager’s register 15, the fifth word of the area addressed by UEPHMSA.

Similarly, when the UERTELUW bit is set (indicating that the resource manager was in read-only mode throughout this LUW), the exit program should cause the resource manager to perform a single-phase commit. There are no return codes for a UERTELUW call. Because no updates were made, data integrity is not at risk, and therefore no action is taken if the commit fails.

On receiving a request to perform the first phase of a two-phase commit, the resource manager is expected to get into a state where recoverable changes made since the last syncpoint can be either committed or backed out on demand, even if there is an intervening system failure. For example, buffer contents must be moved to nonvolatile storage. If the resource manager is unable to get into this state, the exit program should return 'UERFBACK' in register 15, to request backout. Normally, it should return 'UERFPREP', to indicate a successful phase 1 of a 2-phase commit.

On receiving a request to perform the second phase of a two-phase commit, or a request to back out, your resource manager should take the corresponding irreversible step, and have the exit program send the syncpoint manager a return code: either 'UERFDONE', meaning that the commit or abend process is complete; or 'UERFHOLD', meaning that the commit or abend must be resolved later. These return code constants are available to you when you code the macro DFHUEXIT TYPE=RM in your exit program.

If a resource manager cannot understand a call, it should not change the contents of the caller's register 15 before returning to the caller, because it cannot anticipate how the caller will interpret the change.

Resynchronization after failure

If a failure occurs between returning from the exit after performing phase 1 of a 2-phase commit and the subsequent phase 2 or back out call, the resource manager must be ready, on restart, to discover the state of the unit of recovery, and to act accordingly. For programming information about restart resynchronization using the EXEC CICS RESYNC command, see the *CICS/ESA System Programming Reference* manual.

Information about in-doubt units of work is retained across CICS warm starts. CICS initialization and keypoint management routines recover from the system log all information associating resource managers with outstanding units of recovery, which are resolved automatically when CICS reconnects to the resource managers concerned.

However, if for any reason CICS has to be cold started after a failure, information about in-doubt units of work is lost. In these circumstances, the outstanding units of recovery have to be resolved manually. You can use the CICS-supplied sample program, DFH\$INDB, to help you do this. DFH\$INDB produces a list of in-doubt external resource records, with the action needed to resolve each one. Each list relates to a single, specified resource manager, so you may need to execute DFH\$INDB more than once. You can write your own version of DFH\$INDB by modifying the source code supplied in the CICS410.SDFHSAMP library. For more information about DFH\$INDB, see the *CICS/ESA Operations and Utilities Guide*.

Coding a program to be invoked by the CICS task manager

If your exit program sets the task manager bit in the schedule flag word, it is invoked at end-of-task. If you specify TASKSTART on the EXEC CICS ENABLE command for the TRUE, it is invoked at start-of-task, and (providing it does not unset the task manager bit), at end-of-task too. To determine whether a particular invocation is at start- or end-of-task, you can examine the CICS task manager parameters described in “CICS task manager parameters” on page 211. Typically, your program shows interest in task manager events if it needs to save task-related information, such as performance or accounting data, before the task ends.

Limitations

If your exit program is invoked at end-of-task, you must be alert to possible limitations on exit program activity at task-detach. For example:

- Do not update any CICS resources at all during a task-detach exit call, because the CICS syncpoint manager is not invoked again for that task. Note also that all resources (terminals, and so on) except task-storage have been released by end-of-task.

Note: You should also be aware that transactions with resource security or command security defined may not run successfully after the terminal has been released. See the *CICS/ESA CICS-RACF Security Guide* to determine which resources and commands are subject to security checking. Failure to observe these limitations can result in an ABENDAEY7 - NOTAUTH condition arising.

- It is possible to schedule a new CICS task from your exit program using the EXEC CICS START command and to pass data to a new task. However, you should note that EXEC CICS START uses a temporary storage queue to pass data to the new transaction. If this queue is recoverable (DFHTST TYPE=RECOVERY), it is locked to the detaching task. It is never unlocked, because, when the task-detach exit call is made, the resources of the detaching task have **already** been freed. Use of the PROTECT option would cause a different problem: the new task could not be scheduled until the next syncpoint of the detaching task, but there will be no such syncpoint.
- It is recommended that you do not access remote resources using a task-related user exit program. However, if you do so, then you must understand fully the circumstances in which the function shipping conversation may be terminated.

Coding a program to be invoked at CICS termination

If you specify the SHUTDOWN option when enabling your task-related user exit program, it is invoked at system termination. The CICS system termination manager passes the exit program the address of a one-byte code that identifies the type of termination (see “CICS termination manager parameters” on page 211). You can use this invocation of your program to do processing appropriate to the type of termination. For example, at an orderly shutdown you could use it, rather than a PLT program, to shut down the adapter; at a CICS abend you could use it to take special actions, related to the seriousness of the abend.

Limitations

Note that, due to the nature of CICS abends and operator cancels, there is no guarantee that CICS will be able to invoke your exit program at system termination, even if you have specified SHUTDOWN; it may be too impaired to do so.

The limitations on what your program can do, if invoked, depend on the type of termination:

Orderly shutdown (UERTCORD)

Your exit program must follow the rules for programs that execute during the first quiesce stage of CICS shutdown—that is, all CICS services are available, but programs must not start any new tasks.

Immediate shutdown (UERTCIMM)

As for orderly shutdown, except that your exit program should do the minimum required and return control, so that shutdown can proceed.

CICS abend, retry possible, TCBs dispatchable (UERTCABY)

MVS has flagged the failure as being “eligible for retry”. Your exit program must follow the MVS rules for this type of failure, documented in the *MVS/ESA Assembler Programming Guide*.

Subtasks in the region (that is, task control blocks (TCBs) in addition to the CICS job-step TCB) are still dispatchable, and your exit program can execute code under them.

You must not use any CICS services.

CICS abend, retry not possible, TCBs dispatchable (UERTCABN)

MVS has flagged the failure as “not eligible for retry”. Your exit program must follow the MVS rules for this type of failure. Note that your exit program is invoked from code within the CICS extended subtask abend exit (ESTAE). MVS imposes more restrictions on ESTAE code than on non-ESTAE code.

Subtasks in the region are still dispatchable, and your exit program can execute code under them.

You must not use any CICS services.

CICS abend, retry not possible, TCBs not dispatchable (UERTOPCA)

As for UERTCABN, except that subtasks in the region are **not** dispatchable; your exit program must not try to execute code under any TCBs that it may have attached.

Important

In the abend invocations (UERTCABY through UERTOPCA), your exit program must not use any CICS services. This includes the DFHEIENT call, which performs a CICS GETMAIN. To prevent a DFHEIENT call being issued automatically on each invocation of your program, specify the NOPROLOG translator option; but include in the program source your own DFHEIENT call to be issued on non-abend invocations only. An example of how to code a task-related user exit program to be invoked at CICS termination is given in Figure 9 on page 226. For further information about coding a DFHEIENT call, see the *CICS/ESA Application Programming Reference* manual.


```

MVC DFHEIBP,UEPEIB
MVC DFHEICAP,=X'80000000'
*
* .....
* Insert code here for all types of call other than when CICS
* is abending
* (CICS services can be used)
* .....
EXIT DS 0H
DFHEIRET
*
LTORG
*
DFHCTERM DSECT
CTERML DS A
*
CTERMLIST DSECT
CTERMTYPE DS CL1
*
DFHEISTG DSECT
*
* Local working storage for CSECT JTRUE1B
*
RSA DS 18F Register save area
SA DSECT Register save area DSECT
DS F
*
RSACB DS F +004
RSACF DS F +008
RSAR14 DS F +00C
RSAR15 DS F +010
RSAR0 DS F +014
RSAR1 DS F +018
RSAR2 DS F
RSAR3 DS F
RSAR4 DS F
RSAR5 DS F
RSAR6 DS F
RSAR7 DS F
RSAR8 DS F
RSAR9 DS F
RSAR10 DS F
RSAR11 DS F
RSAR12 DS F
DFHREGS
DFHUEXIT TYPE=RM
DFHEISTG
DFHEIEND
PRINT NOGEN
PRINT GEN
END

```

Figure 9 (Part 2 of 2). Sample code for a task-related user exit program to be invoked at CICS termination

Using EDF with your task-related user exit program

If your exit program sets the EDF bit in the schedule flag word and EDF is active, the exit program is invoked before and after each API request to format screens for EDF to display.

Communication between the task-related user exit and EDF is controlled by the task-related user exit interface. The command flow between this interface, EDF, and the task-related user exit is summarized in Figure 10.

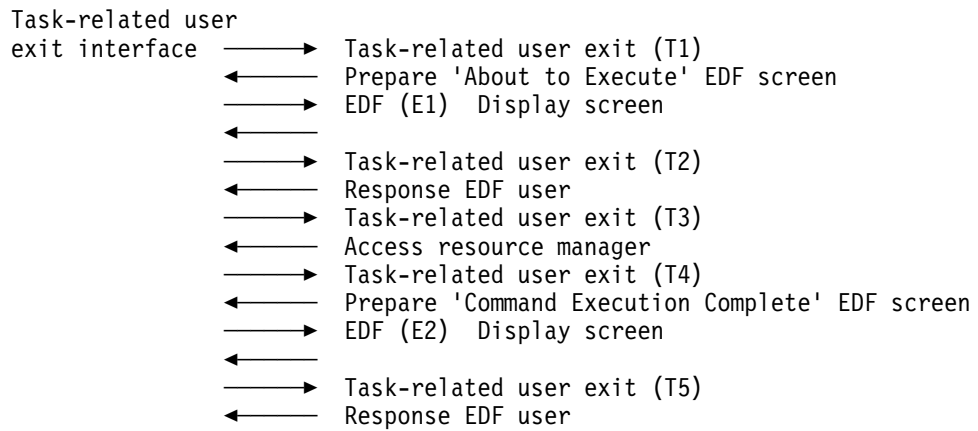


Figure 10. Interface between the task-related user exit and EDF

Table 99 describes each stage of the interface between the task-related user exit and EDF, relating the descriptions to the (Tn) and (En) expressions in Figure 10.

Invocation	Description
(T1)	Task-related user exit invoked to set up its EDF requirements. At this stage the task-related user exit prepares the “About to Execute” screen based on the application request.
(E1)	Using information passed back from the task-related user exit at invocation T1, the task-related user exit interface invokes EDF to display the “About to Execute” screen. EDF sets up the EDF user response, for example, if the user has changed the screen.
(T2)	Task-related user exit is invoked with the EDF user response to the “About to Execute” screen.
(T3)	Task-related user exit invoked to access external resource manager for application request.
(T4)	Task-related user exit invoked to prepare a “Command Execution Complete” screen, based on the result of the application request.
(E2)	Using information passed back from the task-related user exit at invocation T4, the task-related user exit interface invokes EDF to display the “Command Execution Complete” screen. EDF sets up the EDF user response, for example, if the user has changed the screen.
(T5)	Task-related user exit is invoked with the EDF user response to the “Command Execution Complete” screen.

Important

The E1/T2 and E2/T5 cycles can be used repeatedly. This may occur, for example, if the EDF user changes the screen a number of times.

Adapter administration

Careful use of task-related user exits can allow your application programmers to be unaffected by the invocation of non-CICS resource managers from CICS application programs. Enabling and disabling task-related user exit programs for an installation should be the responsibility of one or more supervisory or master terminal operators. This section lists what you must do before you can use the adapter, and describes the commands used by the supervisor to administer task-related user exit programs.

For programming information about the use of commands in CICS application programs, see the *CICS/ESA Application Programming Reference* manual.

What you must do before using the adapter

1. A task-related user exit program must be defined to the system using the CEDA INSTALL PROGRAM command.
2. If you want to use CICS syncpoint management in task-related user exits, your CICS system must contain the supplied module DFHDBP.
3. To enable the task-related user exit program and to define its working storage needs, you must use the EXEC CICS ENABLE command. A task-related user exit program must be both enabled and started before it is available for execution. For example:

```
EXEC CICS ENABLE PROGRAM('EP9')
      TALENGTH(750) GALENGTH(200) SHUTDOWN
EXEC CICS ENABLE PROGRAM('EP9')
      START
```

The first command loads the task-related user exit program EP9, and causes a 200-byte work area to be obtained and associated with it. The first command also schedules the allocation of a further 750-byte work area for each task that subsequently invokes EP9, and the invocation of EP9 at CICS termination. The second command starts the exit program, that is, it makes its entry point capable of being invoked.

Note: If a task-related user exit program is enabled before it has been installed, CICS scans the LPA for the program and may issue message DFHLD0107I, meaning that it was unable to find the program in the LPA and is using the DFHRPL version.

For programming information about the EXEC CICS commands, refer to the *CICS/ESA System Programming Reference* manual.

The administration routines

As well as being enabled before they can be used, task-related user exit programs should be disabled when you have finished using them. You should prepare procedures (the administration routines) for enabling and disabling your task-related user exit programs, using the EXEC CICS ENABLE and DISABLE commands, and for resynchronizing between sessions or after a system failure. Your enabling routines could be PLT initialization programs or online programs. Your disabling routines could, for example, be started by a TRUE invoked at CICS termination.

The EXTRACT EXIT command obtains the address and the length of a global work area that is owned by, or shared by, a named task-related user exit program.

For programming information about these commands and the rules governing them, and also about resynchronization, refer to the *CICS/ESA System Programming Reference* manual.

Tracing a task-related user exit program

CICS outputs a trace entry just before control is passed to the task-related user exit and just after returning from the exit. You can control these trace entries using the EI option of the CETR trace control transaction or the EXEC CICS SET TRACETYPE command.

_____ End of General-use programming interface _____

Chapter 3. The user exit programming interface (XPI)

This chapter describes the user exit programming interface (XPI) of CICS/ESA 4.1. It is divided into the following sections:

- “**Overview**” is an introduction to the XPI.
- “**General form of an XPI call**” on page 234 contains information that applies to all the XPI calls.
- “**Global user exit XPI examples, showing the use of storage**” on page 240 contains two pieces of sample code.
- “**The XPI functions**” on page 247 describes the syntax of the individual XPI calls. The calls are grouped according to the type of function they perform (for example, dump control, storage control). The functional groups are ordered alphabetically.

Overview

The user exit programming interface provides global user exit programs with access to some CICS services. It consists of a set of macro function calls that you can use in your user exit programs. It provides opportunities to extend CICS functions beyond the facilities provided in the standard CICS system, but it must be used with care. Any exit programs you write that use the interface must be written following the guidance in this chapter, and they must be carefully tested to ensure that they cannot cause system errors.

The user exit programs must be in assembler language; the XPI is not provided for other languages. You should also note that programs containing XPI calls must be written to 31-bit standards, and must be reentrant.

You must be in primary-space translation mode when you invoke the XPI. (For information about translation modes, see the *IBM ESA/370 Principles of Operation* manual.)

- Using the XPI **dispatcher functions**, you can:
 - Obtain a suspend token for a task—see “The ADD_SUSPEND call” on page 251
 - Suspend execution of the issuing task—see “The SUSPEND call” on page 253
 - Resume execution of a suspended task—see “The RESUME call” on page 256
 - Release a suspend token associated with a task—see “The DELETE_SUSPEND call” on page 257
 - Request a wait on one or more MVS event control blocks (ECBs)—see “The WAIT_MVS call” on page 258
 - Change the priority of the issuing task—see “The CHANGE_PRIORITY call” on page 262.
- Using the XPI **dump control functions**, you can:
 - Request a system dump—see “The SYSTEM_DUMP call” on page 263
 - Request a transaction dump—see “The TRANSACTION_DUMP call” on page 264.

- Using the XPI **kernel domain functions**, you can:
 - Inhibit purge for the current task—see “The START_PURGE_PROTECTION function” on page 267
 - Reenable purge for the current task—see “The STOP_PURGE_PROTECTION function” on page 267.
- Using the XPI **loader functions**, you can:
 - Define a new program to the loader domain—see “The DEFINE_PROGRAM call” on page 268
 - Load a program or, if it is already loaded, obtain its load and entry-point addresses—see “The ACQUIRE_PROGRAM call” on page 271
 - Release the storage occupied by a program, or decrement its use count by one—see “The RELEASE_PROGRAM call” on page 274
 - Delete a program definition from the list of current programs—see “The DELETE_PROGRAM call” on page 275.
- Using the XPI **monitoring functions**, you can:
 - Process a user event-monitoring point—see “The MONITOR call” on page 276
 - Retrieve the current monitoring data for the issuing task—see “The INQUIRE_MONITORING_DATA call” on page 279.
- Using the XPI **program management functions**, you can:
 - Inquire about the attributes of a specified program—see “The INQUIRE_PROGRAM call” on page 280
 - Inquire about the attributes of the program that is currently running—see “The INQUIRE_CURRENT_PROGRAM call” on page 286
 - Set selected attributes in the definition of a specified program—see “The SET_PROGRAM call” on page 288
 - Browse through program definitions, optionally starting at the definition of a specified program—see “The START_BROWSE_PROGRAM call” on page 291, “The GET_NEXT_PROGRAM call” on page 292, and “The END_BROWSE_PROGRAM call” on page 294
 - Inquire about the settings of the autoinstall function for programs, mapsets, and partitionsets—see “The INQUIRE_AUTOINSTALL call” on page 295
 - Change the settings of the autoinstall function for programs, mapsets, and partitionsets—see “The SET_AUTOINSTALL call” on page 296.
- Using the XPI **state data access functions**, you can:
 - Inquire on application system data in the AP domain—see “The INQ_APPLICATION_DATA call” on page 297
 - Inquire on CICS system data in the AP domain—see “The INQUIRE_SYSTEM call” on page 300
 - Set CICS system data values in the AP domain—see “The SET_SYSTEM call” on page 304.
- Using the XPI **storage control functions**, you can:
 - Obtain and initialize storage—see “The GETMAIN call” on page 305
 - Release storage—see “The FREEMAIN call” on page 308.
 - Inquire about the access-key of an element of storage—see “The INQUIRE_ACCESS call” on page 309
 - Obtain the start address and length of an element of task-lifetime storage—see “The INQUIRE_ELEMENT_LENGTH call” on page 310

- Discover whether CICS is short on storage—see “The INQUIRE_SHORT_ON_STORAGE call” on page 310
 - Inquire about a task’s task-lifetime storage—see “The INQUIRE_TASK_STORAGE call” on page 311
 - Cause CICS to switch from a subspace to base space—see “The SWITCH_SUBSPACE call” on page 312.
- Using the XPI **trace control function**, you can:
 - Write a trace entry to the active trace destinations—see “The TRACE_PUT call” on page 313.
 - Using the XPI **transaction management functions**, you can:
 - Discover the name of the dynamic transaction routing transaction definition—see “The INQUIRE_DTRTRAN call” on page 314
 - Discover the current value of the MXT system initialization parameter—see “The INQUIRE_MXT call” on page 315
 - Inquire about a specified transaction class—see “The INQUIRE_TCLASS call” on page 317
 - Inquire about a specified transaction definition—see “The INQUIRE_TRANDEF call” on page 318
 - Inquire about an attached transaction—see “The INQUIRE_TRANSACTION call” on page 325
 - Change the task priority and transaction class of the current task—see “The SET_TRANSACTION call” on page 329.
 - Using the XPI **user journaling function**, you can:
 - Write a record to a CICS journal—see “The WRITE_JOURNAL_DATA call” on page 331.

APAR PN70222

Documentation for PN70222 added on 7 June 1995.

Note: Using the XPI feature table, you can register a CICS-supplied feature (such as ONC/RPC) to CICS. After it has been registered, you can inquire on the feature, update it, write a trace entry, and deregister it. For details of the XPI feature table, see the *ONC/RPC Feature* manual, SC33-1119.

Important

1. You cannot use all of the XPI calls at every global user exit point. You will find an indication of when these calls **cannot** be used both with the description of each function call, and in the lists of exit points in Chapter 1, “Global user exit programs” on page 3.

Warning: These XPI calls are used to invoke CICS services; using them in the wrong exits causes unpredictable errors in your CICS system.

2. There is a restriction on using the XPI early during initialization. Do not start exit programs that use the XPI functions INQUIRE_MONITOR_DATA, MONITOR, TRANSACTION_DUMP, and WRITE_JOURNAL_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 4, “Writing initialization and shutdown programs” on page 335.
3. These XPI functions are likely to cause the task executing the user exit program to lose control to another task while the XPI function is being executed. Therefore the use of XPI functions must be very carefully considered, as interrupting the flow of CICS functions could cause problems, such as lockouts, to occur.

General form of an XPI call

If you make an XPI call, it must be in the form described below. It consists of two sets of parameters:

- Input parameters, including the XPI function call and the parameters passed to the call
- Output parameters, by which CICS can return values to you, including response and reason codes that tell you whether the call was successful.

To use an XPI macro call, you must include a copy book that defines the input and output parameters. The name of the macro is always of the form DFHxxyX, and the associated copy book has the name DFHxxyY. For example, the GETMAIN call is part of the storage control XPI. The macro you would use is DFHSMMCX and the associated copy book is DFHSMMCX.

The general format (omitting the assembler-language continuation character) of all XPI calls is:

```
macro-name [CALL],
           [CLEAR],
           [IN,
           FUNCTION(call_name),
           mandin1(value),
           mandin2(value),
           ...
           [optin1(value),]
           [optin2(value),]
           ...]
           [OUT,
           mandout1(value),
           mandout2(value),
           ...
           [optout1(value),]
           [optout2(value),]
           ...
           RESPONSE,
           REASON]
```

XPI calls follow assembler-language coding conventions:

- The “macro-name” must begin before column 16.
- The continuation lines must begin in column 16.
- There must be no embedded blanks apart from the blank between the macro-name and the first keyword (usually CALL).
- Entries on lines other than the final line must end with a comma.
- Lines other than the final line must have a continuation character in column 72.
- Parentheses around the input and output values are required—and if you use a register reference as an input or output value, it must be enclosed in an inner pair of parentheses, thus: ((R6)).
- For details about how to set the values of the XPI options, refer to “The XPI functions” on page 247.

There are three uses of these XPI functions. You can:

- Clear the parameter list used by the XPI call
- Set up the input parameters
- Make the call to the CICS function.

You can code all of these individually (see “An example showing how to build a parameter list incrementally” on page 246), or include them in a single statement.

Some options are common to all uses of the XPI. They are included in all of the syntax descriptions, but their explanation is given here. The options are CALL, CLEAR, IN, FUNCTION, OUT, RESPONSE, and REASON.

CALL causes code generation that issues a call to the XPI function. If you specify CALL, IN, FUNCTION, and OUT, then code is generated to perform the whole operation of building the parameter list, invoking the function, and receiving the result. You can omit the CALL, but specify IN to build your parameter list incrementally; later you can use CALL

with that list, coding CALL, IN, FUNCTION, OUT, and all required options. You can then represent the values of the preset options by an asterisk (*) to show that the value is already present in the list.

Warning: If you build your parameter list incrementally, do not specify CLEAR when you finally issue the call, because the CLEAR option sets the parameter list to zeros, which will cause you to lose the preset values.

CLEAR sets the existence bits in the parameter list (both mandatory and optional parameters) to binary zeros. Each macro has a COPY code, which defines the parameter list by a DSECT consisting of a header section, followed by a set of existence bits, and the parameters themselves. For performance reasons, the header section and the existence bits only are cleared. The rest of the parameter list is left unchanged.

+ **Warning:** Failure to clear the parameter list can cause unpredictable
+ results, such as program checks or storage violations. If you are
+ building the parameter list incrementally, specify CLEAR before
+ specifying any parameters. If you are not building the parameter
+ incrementally, specify CLEAR when the CALL is issued.

IN tells CICS that any parameter following the IN option and preceding the OUT option is an input value. It **must** be specified when CALL is specified. If you use the function without CALL to build a parameter list, you can specify IN and some parameter values to store values into your list.

FUNCTION specifies which function of the macro you require; for instance, GETMAIN or FREEMAIN. It **must** be specified when CALL is specified, and unlike other options, it must always be explicit—you **cannot** code “FUNCTION(*)”.

mandin(value)

“mandin” represents an option that becomes mandatory if CALL is specified. “value” may be represented by an asterisk (*) to show that a previous use of the macro has already set the value in the parameter list (see above under “CALL”). For further details about how to complete “value”, refer to the specific function calls in “The XPI functions” on page 247.

OUT tells CICS that any parameter following the OUT option is a receiver field. It **must** be specified when CALL is specified.

Note: The use of the following output parameters with values other than an asterisk (*) is invalid if CALL is not specified.

mandout(value)

“mandout” represents an option that becomes mandatory if CALL is specified. The output is placed in the parameter list if an asterisk (*) is coded, or in the location that you have specified in “value”. RESPONSE is a special case of a mandout option (see below). For further details about how to complete “value”, refer to the specific function calls (see “The XPI functions” on page 247).

optin1,2...; optout1,2....

represent items that are completely optional for **all** forms of the macro; in particular, they do not have to be specified when CALL is specified.

RESPONSE

is a mandatory data area that you define to receive the response from your XPI call. You can use an asterisk (*) to indicate to CICS that the RESPONSE value is to be placed in the parameter list, or you can specify the name of a field in which you want the RESPONSE value to be placed. You need not code the RESPONSE option if you are using the macro without CALL to build a parameter list.

The response from any XPI call is always one of 'OK', 'EXCEPTION', 'DISASTER', 'INVALID', 'KERNERROR', and 'PURGED'. There are standardized names (EQU symbols) for the response code values provided by CICS:

xyyy_OK, xyyy_EXCEPTION, xyyy_DISASTER, xyyy_INVALID,
xyyy_KERNERROR, and xyyy_PURGED,

where "xyyy" is a prefix derived from the four letters of the relevant macro-name following the string 'DFH'. Thus for DFHSMCMX the prefix is SMMC; for DFHLDLDX the prefix is LDLD. Equate values with these names are generated when you include the copy book DFHxyyyY for the macro DFHxyyyX. You cannot assume that the arithmetic values of corresponding RESPONSE codes are the same for all macro calls. The meanings of the RESPONSE codes are as follows:

OK

The XPI request was completed successfully.

EXCEPTION

The function was not completed successfully for a reason which could be expected to happen, and which may be coded for by a program (for example, TRANSACTION_DUMP, EXCEPTION = SUPPRESSED_BY_DUMPTABLE). Any REASON value may provide more information.

DISASTER

The request has failed completely. You cannot recover from this failure within the user exit program. When this failure occurs, CICS takes a system dump, issues an error message, and sets a 'DISASTER' response. On receiving this, your user exit program should exit without attempting any further processing. The REASON value for this response, shown only in the trace, may provide more information. There is no REASON value returned to the calling program.

INVALID

You have omitted a mandatory value, or you have supplied an invalid value for an option. You cannot recover from this failure within the user exit program. When this failure occurs, CICS takes a system dump, issues an error message, and sets an 'INVALID' response. On receiving this response, your user exit program should return to the caller without attempting any further processing. The REASON value for this response, shown only in the trace, may provide more information. This

may help you to correct any error in your exit program. There is no REASON value returned to the calling program.

KERNERROR

The kernel has detected an error with the CICS function you are trying to invoke. Either the function you have requested is unavailable or not valid, or there is an error within CICS.

PURGED

The task has been purged, or an interval specified on your XPI call has expired. Examine the REASON code.

Note that if an XPI call other than DFHDSSRX SUSPEND or WAIT_MVS gets this RESPONSE, your exit program should set the return code to 'UERCPURG' and return to the caller.

If a DFHDSSRX SUSPEND or WAIT_MVS call specifies an INTERVAL and gets this RESPONSE with a REASON of 'TIMED_OUT', it indicates that the INTERVAL you specified has passed. It is up to you to decide what you do next.

If a DFHDSSRX SUSPEND or WAIT_MVS call specifies an INTERVAL and gets this RESPONSE with a REASON of 'TASK_CANCELLED', this indicates that the INTERVAL you specified has not passed but that the task has been purged by an operator or an application. In this case, you must set a return code of 'UERCPURG' and return.

If a DFHDSSRX SUSPEND or WAIT_MVS call does **not** specify an INTERVAL, and gets this RESPONSE with a REASON of 'TASK_CANCELLED' or 'TIMED_OUT', it indicates that the task has been purged by an operator or an application, or by the deadlock time-out facility. In this case, you must set a return code of 'UERCPURG' and return.

You **must not** return the response code 'UERCPURG' to CICS for any other reason. If you attempt to do so, your program will have unpredictable results.

REASON This is a mandatory data area that you define in order to receive more information about the RESPONSE value. You can use (*) to indicate to CICS that the REASON value is to be placed in the parameter list. On most XPI calls, standardized reason names (EQU symbols) are provided only for RESPONSE values of 'EXCEPTION' and 'PURGED'. The REASON values that accompany responses vary from one XPI function to another, so details are provided with the descriptions of the XPI calls.

REASON is not applicable where RESPONSE was 'OK'. In these circumstances, you should not test the REASON field.

Note: For examples of how to initialize the parameter list, set up parameters, make the call, and receive output parameters, refer to "Global user exit XPI examples, showing the use of storage" on page 240. That section includes both a complete example, and also an example in which each step is executed separately.

Setting up the XPI environment

The exit programming interface (XPI) does not require the usual CICS transaction environment, but you do need to set up a special exit programming environment before you can use any XPI calls. If you are going to use any of the XPI functions in an exit program, you must include in your program, at a point **before** you issue any XPI calls, the macro:

```
DFHUEXIT TYPE=XPIENV
```

The expansion of this macro provides the DSECTs that are used in all XPI calls. It also provides a list of register equates (R0 EQU 0, R1 EQU 1, and so on), that you can use in your exit program. The other fields generated by the macro are used by CICS to perform the XPI call processing. You must not use any of these fields: if you do so, your user exit program will have unpredictable results.

The user exit program should be in 31-bit addressing mode.

XPI register usage

Before you can issue an XPI call from a global user exit program, you **must** move the contents of the parameter UEPSTACK (the kernel stack entry) of DFHUEPAR to the exit program's register 13.

The XPI function expansion uses registers 0, 1, 14, and 15, so the exit program must save and restore them if necessary around an XPI call.

For an example of how to use EXEC CICS commands and XPI calls in the same exit program, see Appendix E, "The example program for the XTSEREQ global user exit, DFH\$XTSE" on page 717.

The XPI copy books

There is a copy book for each XPI function, to provide the DSECTs associated with that function. These DSECTs allow you to map the parameters and the response and reason codes of an XPI call. You must include in your exit program a COPY statement for each XPI function that you are going to use. The copy book name is the same as the macro name, except that the final letter "X" becomes a letter "Y".

For example, to include the copy book for the XPI function DFHSMMCX, you must include the statement:

```
COPY DFHSMMCY
```

Trace entries for your XPI calls show these parameter lists if you have tracing on for the function you are using.

Reentrance considerations resulting from XPI calls

During an XPI call, CICS may give control to another task while processing the XPI call. This second task could also cause the same exit program to be invoked and the same XPI call to be made, but perhaps this time with different parameter values. **It is your responsibility to ensure that lockout situations do not occur.**

While processing an XPI call, CICS may encounter another user exit point that uses the same user exit program. Therefore the XPI parameter lists must be built in storage associated with a single invocation of the exit program.

XPI examples

If your exit program is a global user exit, CICS provides it with 260 bytes of LIFO storage, which is exclusive to a single invocation of your exit program. Your exit program can access this storage using parameter UEPXSTOR of the DFHUEPAR parameter list. Use this storage to base the DSECT provided by the DFHxxyyY copy book when building the XPI parameter list. In this way, the parameters are not corrupted if the exit program is reentered.

Parameter lists for the XPI services provided here do not exceed 256 bytes. The remaining 4 bytes of the UEPXSTOR storage can be used to store the address of some additional storage that relates to the invocation of the exit program. This storage could be obtained by either a DFHSMMCX FUNCTION (GETMAIN) macro, or an MVS GETMAIN request.

A suggested standard would be to use the first 4 bytes to hold the address of the additional storage, and to keep the following 256 bytes for XPI parameter lists.

Information to be kept across invocations of an exit program can be stored in the global work area that you can define for an exit program (or group of exit programs). The 260 bytes of LIFO storage cannot be used for this purpose because it is dynamic.

Global user exit XPI examples, showing the use of storage

The example in Figure 11 on page 241 illustrates the use of the XPI and storage in a global user exit program. It is not a complete program, but merely an example of entry and exit code for any global user exit program, and the use of the XPI function.

The options of the DFHSMMCX macro used in the example are described in “Storage control functions” on page 305.

The example uses the technique of obtaining some storage for this invocation of the program using the XPI GETMAIN call, and then saving the address of this storage in the first 4 bytes of the LIFO storage addressed by UEPXSTOR. In this example, the initialization of the parameter list (using the CLEAR option), the building of the parameter list, and the GETMAIN call occur in a single macro. For details of how to build the parameter list incrementally, and how to separate the CLEAR and the GETMAIN call, refer to “An example showing how to build a parameter list incrementally” on page 246.


```

          TITLE 'GUEXPI - GLOBAL USER EXIT PROGRAM WITH XPI'
*****
*   The first three instructions set up the global user exit   *
*   environment, identify the user exit point, prepare for the use of *
*   the exit programming interface, and copy in the definitions that *
*   are to be used by the XPI function.                         *
*****
*
*           DFHUEXIT TYPE=EP,ID=XFCREQ           PROVIDE DFHUEPAR PARAMETER
*           LIST FOR XFCREQ IN THE FILE
*           CONTROL PROGRAM AND LIST
*           OF EXITID EQUATES
*
*           DFHUEXIT TYPE=XPIENV                 SET UP ENVIRONMENT FOR
*           EXIT PROGRAMMING INTERFACE --
*           MUST BE ISSUED BEFORE ANY
*           XPI MACROS ARE ISSUED
*
*           COPY DFHSMCMCY                       DEFINE PARAMETER LIST FOR
*           USE BY DFHSMCMCX MACRO
*
*****
*   The following DSECT maps a storage area you can use to make the *
*   exit program reentrant by storing the address of the storage you *
*   acquire in the first four bytes of the 260-byte area provided by *
*   the user exit handler (DFHUEH) and addressed by UEPXSTOR.      *
*****
*
*   TRANSTOR DSECT                                     DSECT FOR STORAGE OBTAINED BY
*   :                                                     GETMAIN
*   :
*   storage declarations
*   :
*   *

```

Figure 11 (Part 1 of 6). Global user exit program with XPI

XPI examples

```
*****
* The next seven instructions form the normal start of a global user *
* exit program, setting the program addressing mode to 31-bit, saving *
* the calling program's registers, establishing base addressing, and *
* establishing the addressing of the user exit parameter list. *
*****
*
GXPI      CSECT
GXPI      AMODE 31                      SET TO 31-BIT ADDRESSING
*
          SAVE (14,12)                  SAVE CALLING PROGRAM'S REGISTERS
*
          LR   R11,R15                   SET UP USER EXIT PROGRAM'S
          USING GXPI,R11                 BASE REGISTER
*
          LR   R2,R1                     SET UP ADDRESSING FOR USER
          USING DFHUEPAR,R2              EXIT PARAMETER LIST -- USE
*                                         REGISTER 2 AS XPI CALLS USE
*                                         REGISTER 1
*
*****
* Before issuing an XPI function call, set up addressing to XPI *
* parameter list. *
*****
*
          L    R5,UEPXSTOR                SET UP ADDRESSING FOR XPI
*                                         PARAMETER LIST
*
          USING DFHSMC_ARG,R5            MAP PARAMETER LIST
*
*****
* Before issuing an XPI function call, you must ensure that register *
* 13 addresses the kernel stack. *
*****
*
          L    R13,UEPSTACK               ADDRESS KERNEL STACK
*
```

Figure 11 (Part 2 of 6). Global user exit program with XPI

```

*****
* Issue the DFHSMMCX macro call, specifying:
*
* CALL --      the macro is to be called immediately
*
* CLEAR --     initialize the parameter list before inserting values.
*
* IN --        input values follow.
*
*              FUNCTION(GETMAIN) -- acquire storage
*              GET_LENGTH(120) -- 120 bytes of it
*              SUSPEND(NO) -- don't suspend if storage not available
*              INITIAL_IMAGE(X'00') -- clear acquired storage
*                               to hex zero throughout.
*              STORAGE_CLASS(USER) -- class of storage to be
*                               acquired is user storage
*                               above the 16MB line.
*
* OUT --       output values follow
*
*              ADDRESS((R6)) -- put address of acquired storage in
*                               register 6.
*              RESPONSE(*) -- put response at SMMC_RESPONSE in
*                               macro parameter list.
*              REASON(*) -- put reason at SMMC_REASON in macro
*                               parameter list.
*
*****
*
*              DFHSMMCX CALL,
*              CLEAR,
*              IN,
*              FUNCTION(GETMAIN),
*              GET_LENGTH(120),
*              SUSPEND(NO),
*              INITIAL_IMAGE(X'00'),
*              STORAGE_CLASS(USER),
*              OUT,
*              ADDRESS((R6)),
*              RESPONSE(*),
*              REASON(*)
*

```

Figure 11 (Part 3 of 6). Global user exit program with XPI

XPI examples

```
*****
* Test SMMC_RESPONSE -- if OK, then branch round error handling.      *
*****
*
      CLI   SMMC_RESPONSE,SMMC_OK   CHECK RESPONSE AND...
      BE    STOK                    ...IF OK, BYPASS ERROR ROUTINES
*
:
      error-handling routines
:
*
*****
* The next section maps TRANSTOR on the acquired storage.            *
*****
*
STOK   DS    0H
       USING TRANSTOR,R6          MAP ACQUIRED STORAGE
       ST    R6,0(R5)             SAVE STORAGE ADDRESS IN FIRST
*                                  4 BYTES OF STORAGE ADDRESSED
*                                  BY UEPXSTOR
*
       LA    R5,4(R5)            ADDRESS 4-BYTE OFFSET
       DROP  R5                  REUSE REGISTER 5 TO BASE ALL
       USING DFHxyy_ARG,R5      FOLLOWING XPI PARAMETER LISTS
*                                  AT 4-BYTE OFFSET INTO STORAGE
*                                  ADDRESSED BY UEPXSTOR
:
rest of user exit program
:
*
*****
* When the rest of the exit program is completed, free the storage
* and return.
*****
*
       DROP  R5                  REUSE REGISTER 5 TO MAP DFHSMMC
       USING DFHSMMC_ARG,R5     XPI PARAMETER LIST
*
       L     R13,UEPSTACK        ADDRESS KERNEL STACK
*
```

Figure 11 (Part 4 of 6). Global user exit program with XPI

```

*****
* Issue the DFHSMCX macro call, specifying: *
* * * * *
* CALL -- the macro is to be called immediately. *
* * * * *
* CLEAR -- initialize the parameter list before inserting values. *
* * * * *
* IN -- input values follow. *
* * * * *
* FUNCTION(FREEMAIN) -- release storage *
* ADDRESS((R6)) -- address of storage is in register 6. *
* STORAGE_CLASS(USER) -- class of acquired storage was *
* 31-bit user storage. *
* * * * *
* OUT -- output values follow *
* * * * *
* RESPONSE(*) -- put response at SMMC_RESPONSE in *
* macro parameter list. *
* REASON(*) -- put reason at SMMC_REASON in macro *
* parameter list. *
* * * * *
*****
*
* DFHSMCX CALL, +
* CLEAR, +
* IN, +
* FUNCTION(FREEMAIN), +
* ADDRESS((R6)), +
* STORAGE_CLASS(USER), +
* OUT, +
* RESPONSE(*), +
* REASON(*) +
* *

```

Figure 11 (Part 5 of 6). Global user exit program with XPI

XPI examples

```
*****
* Test SMMC_RESPONSE -- if OK, then branch round error handling. *
*****
*
      CLI   SMMC_RESPONSE,SMMC_OK   CHECK RESPONSE AND...
      BE    STEND                    ...IF OK, BYPASS ERROR ROUTINES
*
:
      error-handling routines
:
*
*****
* Restore registers, set return code, and return to user exit handler *
*****
*
STEND   DS    0H
        L     R13,UEPEPSA
        RETURN (14,12),RC=UERCNORM
        LTORG
        END   GXPI
```

Figure 11 (Part 6 of 6). Global user exit program with XPI

An example showing how to build a parameter list incrementally

In the following example, the parameter list is built incrementally. The initialization of the parameter list (using the CLEAR option), the building of the parameter list, and the GETMAIN call are separated into discrete steps.

```
      DFHSMCX CLEAR
:
      DFHSMCX GET_LENGTH(100)
:
      DFHSMCX CALL,
          IN,
          FUNCTION(GETMAIN),
          GET_LENGTH(*),
          SUSPEND(NO),
          INITIAL_IMAGE(X'00'),
          STORAGE_CLASS(USER),
          OUT,
          ADDRESS((R6)),
          RESPONSE(*),
          REASON(*)
```

Important

You must set your parameters using **only** the XPI functions.

The XPI functions

The following sections of the chapter provide details of the individual XPI function calls. The description of each function defines only the options that are specific to that call. Options that are applicable to all function calls are described in “General form of an XPI call” on page 234. The following argument types are used:

name1, name2,....

Each of these refers to the name of a field of the given size in bytes. “name1” means that the name you specify should be that of a 1-byte field.

literalconst

A number in the form of a literal, for example B'00000000', X'FF', X'FCF4', '0', or an equate symbol with a similar value.

expression

A valid assembler-language expression: a decimal integer, or any arithmetic expression (including symbolic values) valid in assembler language; for example:

20; L'AREA; L'AREA+10; L'AREA+X'22'; SYMB/3+20 .

(Rn)

A register reference. The parentheses shown here are required in addition to those that surround the argument. For example: OPTION((R5)).

block-descriptor

Represents a source of both the data address and the data length fields. A block-descriptor can be either a single value or a double value. The following is the single-value form:

OPTION(blkname)

blkname The name of a block-descriptor. A pair of contiguous fullwords, in which the first word contains the address of the data, and the second word contains the length in bytes of the data, as a fullword binary value. Register notation is not accepted for this single-value form.

The following is the double-value form:

OPTION(addr, len)

addr

The data address as {namea | (Ra) | aliteral}:

namea The name of a location containing the data address

(Ra) A register containing the data address

aliteral An address constant literal; for example: A(data).

len The data length as {name1 | (Rn) | expression}:
name1 The name of a location containing a binary fullword giving the data length in bytes
(Rn) A register, the contents of which specify in fullword binary the number of bytes of data
expression A decimal integer, or any arithmetic expression, including symbolic values, valid in assembler language; for example:
L'AREA ; L'AREA+10 ; L'AREA+X'22' ; SYMB/3+20 .

buffer-descriptor

Represents a source of both the data address and the maximum data length fields. Parts of the buffer-descriptor are also reserved to act as receiving fields for output information. A buffer-descriptor can be either a single value or a multiple value. The following is the single-value form:

OPTION(bufdname)

bufdname The name of a buffer-descriptor. A group of four contiguous fullwords, in which:

- The first word contains the address of the data (input).
- The second word is reserved to receive the current length in bytes of the data, as a fullword binary value (output).
- The third word contains the maximum length in bytes of the data, as a fullword binary value (input).
- The fourth word is reserved for use by the XPI.

Register notation is not accepted for this single-value form.

The following is the multiple-value form:

OPTION(addr,maxlen,*)

addr The data address as {namea | (Ra) | aliteral}:
namea The name of a location containing the data address
(Ra) A register containing the data address
aliteral An address constant literal, for example, A(data).
maxlen The maximum data length as {name1 | (Rn) | expression}:
name1 The name of a location containing a binary fullword giving the maximum data length in bytes
(Rn) A register, the contents of which specify in fullword binary the maximum number of bytes of data

expression A decimal integer, or any arithmetic expression, including symbolic values, valid in assembler language; for example:

L'AREA ; L'AREA+10 ; L'AREA+X'22' ; SYMB/3+20 .

* A required parameter to indicate that the parameter list is to be used for the reserved fields.

Dispatcher functions

There are six XPI dispatcher functions. These are the DFHDSSRX calls ADD_SUSPEND, SUSPEND, RESUME, DELETE_SUSPEND, and WAIT_MVS, and the DFHDSATX call CHANGE_PRIORITY.

Usage of these dispatcher calls is very limited. Check the details supplied for each exit in Chapter 1, "Global user exit programs" on page 3 before using any functions.

Notes:

1. You must issue an ADD_SUSPEND call to create a suspend token **before** you issue a SUSPEND or RESUME call.
2. If a suspended task is canceled, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TASK_CANCELLED'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TASK_CANCELLED'.
3. If a suspended task is timed out, the SUSPEND fails with a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. A corresponding RESUME call returns with a RESPONSE value of 'EXCEPTION' and a REASON value of 'TIMED_OUT'.
4. Dispatcher protocols require that you issue a RESUME even if the SUSPEND was purged (due to task cancel or time-out). You must issue one and only one RESUME for each SUSPEND call.

Synchronization protocols for SUSPEND and RESUME processing

This section describes the protocols that must be observed by users of XPI SUSPEND and RESUME processing, so that task purging can be handled effectively.

The normal synchronization protocol

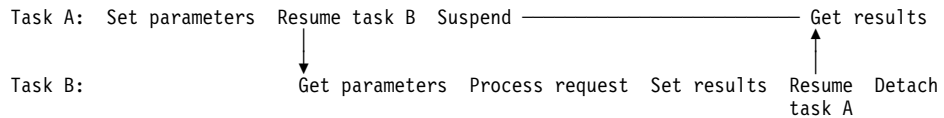
In the normal case, synchronization involves two tasks and three operations. In the following sample operations, the tasks are A (the task that requests a service) and B (the task that processes a request from task A).

1. Task A starts the request by:
 - Setting the parameters to be used by task B
 - Resuming task B
 - Issuing the SUSPEND call.

dispatcher functions

2. Task B performs the action by:
 - Getting the parameters
 - Performing the action
 - Setting the results
 - Terminating (or waiting for new work).
3. Task A ends the interaction by:
 - Getting the results left by task B.

This sequence looks like:



Ignoring the Resume and Suspend, the execution amounts to:

Set parameters; Get parameters; Process request; Set results; Get results

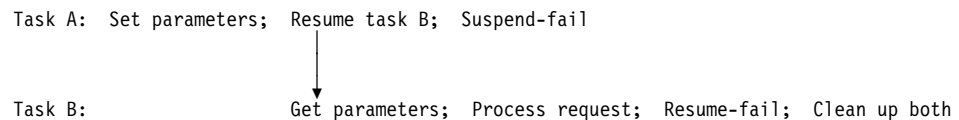
where these actions are always **sequential**.

The synchronization protocol and task purge

If one of the tasks is to be purged, it is task A, because task A is the one suspended. In this case, execution of task A after the failed SUSPEND would be in parallel with task B; the proper serialization would be lost. If the program were left unchanged, Process request and Set results would be taking place at the same time as Get results, with unpredictable results.

One way of preventing this problem is to ensure that **task A, if it is to be purged, does not do anything that could interfere with task B.** (This may well mean that A must not detach, if doing so releases storage that B needs to access.) Because the only task that is now involved is task B, B is left with the responsibility of cleaning up for both tasks.

The sequence is:



Because task-purging is effective only if performed between SUSPEND and RESUME, Suspend-fail precedes Resume-fail. This means that, with the same constraints on serialization as in the normal synchronization protocol, the task-purge protocol can be logically reduced to:

Set parameters; Get parameters; Process request; Clean up

The difference is that Set results and Get results are replaced by Clean up. It is vital that only these two sequences can happen; this means that both programs must be coded correctly. CICS ensures that both tasks are told either that SUSPEND and RESUME processing worked, or that it failed.

The following shows the programming steps that conform to these rules:

<pre> Program for Task A SET PARAMETERS; RESUME B; SUSPEND A; if RESPONSE = OK then GET RESULTS; endif </pre>	<pre> Program for Task B GET PARAMETERS; PROCESS REQUEST; RESUME A; if RESPONSE ≠ OK then CLEAN UP endif </pre>
--	---

If both the SUSPEND and RESUME return 'OK', the example follows the rules for the normal synchronization; processing finishes at Get results. If neither SUSPEND nor RESUME returns 'OK', the example follows the rules for the task-purge protocol, and processing finishes at Clean up.

For further information about SUSPEND and RESUME processing, see the *CICS/ESA Problem Determination Guide*.

Alternative approach to task purge

The sequence described above is one method for dealing with the problem of task purge. Using this method, task B does not know, when it is processing the request, whether or not task A has been purged; this means that B must take great care in its use of resources owned by A (in case A has been purged). In some situations, this restriction may cause difficulties.

A different approach is as follows; if task A is to be purged:

1. A communicates to B that it is no longer available, thus informing B not to use any resources owned by A.
2. A performs its own clean-up processing (including issuing the RESUME call for the purged SUSPEND, as required by the dispatcher protocols), and abends.
3. B performs its own clean-up processing.

The ADD_SUSPEND call

ADD_SUSPEND acquires a suspend token that can later be used to identify a SUSPEND/RESUME pair.

ADD_SUSPEND

```

DFHDSSRX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(ADD_SUSPEND),
  [RESOURCE_NAME(name8 | string | 'string'),]
  [RESOURCE_TYPE(name8 | string | 'string'),]]
  [OUT,
  SUSPEND_TOKEN(name4 | (Rn)),
  RESPONSE(name1 | *),
  REASON(name1 | *)]
        
```

RESOURCE_NAME(name8 | string | 'string')

refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

- name8** The name of the location where an 8-byte value is stored.
- string** A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.
- 'string'** A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_NAME on ADD_SUSPEND supplies a default value which is used if RESOURCE_NAME is not specified on a SUSPEND call.

RESOURCE_TYPE(name8 | string | 'string')

refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

- name8** The name of the location where an 8-byte value is stored.
- string** A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.
- 'string'** A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: RESOURCE_TYPE on ADD_SUSPEND supplies a default value which is used if RESOURCE_TYPE is not specified on a SUSPEND call.

SUSPEND_TOKEN(name4 | (Rn))

is a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

- name4** The name of a 4-byte field where the token is stored
- (Rn)** A register into which the token value is loaded.

RESPONSE and REASON values for ADD_SUSPEND:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

The SUSPEND call

SUSPEND suspends execution of a running task. The task can be resumed in one of two ways. You can issue the XPI RESUME call, or the task is resumed automatically if the INTERVAL value that you specify on the DFHDSSRX macro expires. Suspended tasks can also be purged by the operator, or by an application, or by the deadlock time-out facility.

```

SUSPEND
DFHDSSRX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(SUSPEND),
  PURGEABLE(YES|NO),
  SUSPEND_TOKEN(name4 | (Rn)),
  [INTERVAL(name4 | (Rn)),]
  [RESOURCE_NAME(name8 | string | 'string'),]
  [RESOURCE_TYPE(name8 | string | 'string'),]]
  [OUT,
  [WLM_WAIT_TYPE,]
  [COMPLETION_CODE(name1 | (Rn)),]
  RESPONSE(name1 | *),
  REASON(name1 | *)]
  
```

PURGEABLE(YES|NO)

indicates whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 100. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task.
- To suppress the deadlock time-out (DTIMOUT) facility (if applicable to this task) for the duration of this request.

Table 100. SUSPEND call – RESPONSE(PURGED)

REASON	TASK_CANCELLED		TIMED_OUT	
CONDITION	PURGE	FORCEPURGE	DTIMOUT	INTERVAL
PURGEABLE (NO)	Canceled	Proceeds normally	Canceled	Proceeds normally
PURGEABLE (YES)	Proceeds normally	Proceeds normally	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

SUSPEND_TOKEN(name4 | (Rn))

is a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

- name4** The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call
- (Rn)** A register containing the token value.

INTERVAL(name4 | (Rn))

specifies in seconds the time after which the task is automatically resumed and given a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'. The INTERVAL value overrides any time-out (DTIMOUT) value specified for the transaction.

name4 The name of a 4-byte area, which is interpreted as a binary fullword.

(Rn) A register containing the interval value, a binary fullword.

RESOURCE_NAME(name8 | string | 'string')

refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8 The name of the location where an 8-byte value is stored.

string A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

'string' A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Notes:

1. CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in the *CICS/ESA Problem Determination Guide*.
2. If RESOURCE_NAME is not specified, the default value, if any, from ADD_SUSPEND is used.

RESOURCE_TYPE(name8 | string | 'string')

refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name8 The name of the location where an 8-byte value is stored.

string A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

'string' A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Notes:

1. CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in the *CICS/ESA Problem Determination Guide*.
2. If RESOURCE_TYPE is not specified, the default value, if any, from ADD_SUSPEND is used.

WLM_WAIT_TYPE(name1)

specifies, in a 1-byte location, the reason for suspending the task. This indicates the nature of the task's wait state to the MVS workload manager.

The equated values for the type of wait are as follows:

CONV

Waiting on a conversation.

CMDRESP

Waiting on a command response.

DISTRIB

Waiting on a distributed request.

IDLE

A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK

Waiting on a lock.

MISC

Waiting on an unidentified resource.

Note: This is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT

Waiting on another product to complete its function; for example, when the workload has been passed to DB2.

SESS_LOCALMVS

Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK

Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX

Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER

Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), you are recommended to specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

COMPLETION_CODE (name1 | (Rn))

is a field to be used in communicating a user-defined “reason for RESUME” code during suspend and resume processing.

name1 The name of a 1-byte area to receive the code. The value in this field is user-defined, and is ignored by CICS.

dispatcher functions

(Rn) A register in which the low-order byte contains the completion code and the other bytes are zero.

RESPONSE and REASON values for SUSPEND:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED
	TIMED_OUT

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘TASK_CANCELLED’ means that the task has been canceled by operator action or by an application command.
3. After a ‘PURGED’ response, the suspend token must not be reused in another SUSPEND until it has been reset by a RESUME corresponding to the purged SUSPEND.
4. ‘TIMED_OUT’ means that the task has been automatically resumed because the specified INTERVAL (or the time-out value specified at task attach) has expired. The token, however, remains suspended and must be the object of a RESUME before it can be the object of a DELETE_SUSPEND.

The RESUME call

RESUME restarts execution of a task that is suspended or timed out. There must be only one RESUME request for each SUSPEND. However, because this is an asynchronous interface, a SUSPEND can be received either before or after its corresponding RESUME. You must ensure that you keep account of the SUSPEND and RESUME requests issued from your exit program.

RESUME

```
DFHDSSRX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(RESUME),
  SUSPEND_TOKEN(name4 | (Rn)),
  [COMPLETION_CODE(name1 | (Rn)),]]
  [OUT,
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```


SUSPEND_TOKEN(name4 | (Rn))

is a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

- name4** The name of a location where you have a 4-byte token previously obtained as output from an ADD_SUSPEND call
- (Rn)** A register containing the token value.

COMPLETION_CODE(name1 | (Rn))

is a field to be used in communicating a user-defined “reason for action” code during suspend and resume processing.

- name1** The name of a 1-byte area to receive the code
- (Rn)** A register, in which the low-order byte contains the completion code and the other bytes are zero.

RESPONSE and REASON values for RESUME:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	TASK_CANCELLED TIMED_OUT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘TASK_CANCELLED’ means that the task was canceled by operator action while it was suspended, and that the suspend token is available for use.

The DELETE_SUSPEND call

DELETE_SUSPEND releases a suspend token associated with this task.

```

DELETE_SUSPEND
DFHDSSRX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(DELETE_SUSPEND),
          SUSPEND_TOKEN(name4 | (Rn)),]
          [OUT,
          RESPONSE(name1 | *),
          REASON(name1 | *)]
    
```

SUSPEND_TOKEN(name4 | (Rn))

is a token assigned by the system to identify the SUSPEND/RESUME pair of operations used on the task.

- name4** The name of a 4-byte field, where the token obtained by an ADD_SUSPEND call has been stored
- (Rn)** A register containing the token value previously obtained.

RESPONSE and REASON values for DELETE_SUSPEND:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

The WAIT_MVS call

WAIT_MVS requests a wait on an MVS event control block (ECB) or on a list of MVS ECBs. For example, you could issue the WAIT_MVS to wait for completion of an MVS task for which you have issued ATTACH and provided a task-completion ECB.

The dispatcher does not clear the ECBs when a WAIT_MVS request is received. If any ECB is already posted, control is returned immediately to the exit program with a response of 'OK'.

A single ECB must not be the subject of more than one wait at a time. If any ECB is already being waited on when a WAIT_MVS request is received, the request is rejected. The RESPONSE code is 'DSSR_INVALID', and the REASON code 'DSSR_ALREADY_WAITING'.

Note: ECBs used in WAIT_MVS requests must never be “hand posted”. They must be posted using the MVS POST macro.

```

WAIT_MVS
DFHDSSRX [CALL,]
    [CLEAR,]
    [IN,
    FUNCTION(WAIT_MVS),
    {ECB_ADDRESS(name4 | (Ra)) | ECB_LIST_ADDRESS(name4 | (Ra)),}
    PURGEABLE(YES|NO),
    [INTERVAL(name4 | (Rn)),]
    [RESOURCE_NAME(name8 | string | 'string'),]
    [RESOURCE_TYPE(name8 | string | 'string'),]
    [OUT,
    [WLM_WAIT_TYPE,]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
```

ECB_ADDRESS(name4 | (Ra))

is the address of the ECB to be waited on.

- name4** The name of a location that contains an ECB address.
- (Ra)** A register that contains the address of an ECB.

ECB_LIST_ADDRESS(name4 | (Ra))

is the address of a list of ECB addresses to be waited on.

- name4** The name of a location that contains an ECB address, possibly followed by more ECB addresses. The last address word in the list has the high-order bit set to 1.
- (Ra)** A register pointing to an address list as described above.

PURGEABLE(YES|NO)

indicates whether your code can cope with the request being abnormally terminated as a result of a purge. There are four types of purge, as shown in Table 101. Specifying PURGEABLE(NO) tells the dispatcher:

- To reject any attempt to PURGE the task
- To suppress the deadlock time-out (DTIMOUT) facility (if applicable to this task) for the duration of this request.

<i>Table 101. WAIT_MVS call – RESPONSE(PURGED)</i>				
REASON	TASK_CANCELLED		TIMED_OUT	
CONDITION	PURGE	FORCEPURGE	DTIMOUT	INTERVAL
PURGEABLE (NO)	Canceled	Proceeds normally	Canceled	Proceeds normally
PURGEABLE (YES)	Proceeds normally	Proceeds normally	Proceeds normally	Proceeds normally

Note: A FORCEPURGE always assumes that the user wants the task to be purged, and so overrides the PURGEABLE(NO) option. If the user has set an INTERVAL, then this, too, overrides the PURGEABLE(NO) option.

INTERVAL(name4 | (Rn))

specifies in seconds the time after which the task is automatically resumed and given a RESPONSE value of 'PURGED' and a REASON value of 'TIMED_OUT'.

The INTERVAL value overrides any time-out (DTIMOUT) value specified for the transaction.

- name4** The name of a 4-byte area, which is interpreted as a binary fullword
- (Rn)** A register containing the interval value, a binary fullword.

RESOURCE_NAME(name8 | string | 'string')

refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

- name8** The name of the location where an 8-byte value is stored.
- string** A string of characters without intervening blanks; if it is not 8 bytes long, it is extended with blanks or truncated as required.

'string' A string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_NAME information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in the *CICS/ESA Problem Determination Guide*.

RESOURCE_TYPE(name8 | string | 'string')

The option refers to an 8-character string that can be used to document and trace the resource involved in suspend and resume processing. You cannot use register notation to specify the address of the string.

name the name of the location where an 8-byte value is stored.

string a string of characters without intervening blanks; if it is not 8 bytes long, it will be extended with blanks or truncated as required.

'string' a string of characters enclosed in quotation marks. Blanks are permitted in the enclosed string. If you want to document a name (label) in your program, use this form.

Note: CICS does not use the RESOURCE_TYPE information but includes it in trace entries, and displays it on appropriate CEMT screens to help you to see what your task is doing. CICS internal requests specify values, and you should use different values to avoid ambiguity. CICS internal request values are documented in the *CICS/ESA Problem Determination Guide*.

WLM_WAIT_TYPE(name1)

specifies, in a 1-byte location, the reason for suspending the task. This indicates the nature of the task's wait state to the MVS workload manager.

The equated values for the type of wait are as follows:

CONV

Waiting on a conversation.

CMDRESP

Waiting on a command response.

DISTRIB

Waiting on a distributed request.

IDLE

A CICS task, acting as a work manager, that has no work request that is allowed to service within the monitoring environment. For example, journaling code that suspends itself when there are no journaling I/O operations to perform.

IO Waiting on an I/O operation or indeterminate I/O-related operation (locks, buffer, string, and so on).

LOCK

Waiting on a lock.

MISC

Waiting on an unidentified resource.

Note: This is the default reason given to the wait if you suspend a task and do not specify the WLM_WAIT_TYPE parameter.

OTHER_PRODUCT

Waiting on another product to complete its function; for example, when the workload has been passed to DB2.

SESS_LOCALMVS

Waiting on the establishment of a session in the MVS image on which this CICS region is running.

SESS_NETWORK

Waiting on the establishment of a session elsewhere in the network (that is, not on this MVS image).

SESS_SYSPLEX

Waiting on establishment of a session somewhere in the sysplex (that is, not on this MVS image).

TIMER

Waiting on the timeout of a timer (for example, a task that puts itself to sleep).

If you are running CICS in an MVS goal-mode workload management environment (that is, you are using goal-oriented performance management), you are recommended to specify the reason for suspending the task on the WLM_WAIT_TYPE parameter.

RESPONSE and REASON values for WAIT_MVS:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	TASK_CANCELLED
	TIMED_OUT

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in "General form of an XPI call" on page 234.
2. 'TIMED_OUT' is returned if the INTERVAL expires, or if a deadlock time-out interval expires.
3. 'TASK_CANCELLED' means that the task has been canceled by operator action or by an application command.

The CHANGE_PRIORITY call

CHANGE_PRIORITY allows the issuing task to change its own priority. It cannot be used to change the priority of another task. This command causes the issuing task to release control, and so provide other tasks with the opportunity to run.

```

CHANGE_PRIORITY
DFHDSATX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(CHANGE_PRIORITY),
          PRIORITY(name1 | (Rn) | decimalint | literalconst),]
          [OUT,
          [OLD_PRIORITY(name1 | (Rn)),]
          RESPONSE(name1 | *),
          REASON(name1 | *)]
    
```

PRIORITY(name1 | (Rn) | decimalint | literalconst)

is the new priority to be assigned to the issuing task.

- name1** The name of a 1-byte field, with a value in the range 0 through 255.
- (Rn)** A register with the low-order byte containing the new priority value.
- decimalint** A decimal integer not exceeding 255 in value. Neither an expression nor hexadecimal notation is allowed.
- literalconst** A number in the form of a literal, for example B'00000000', X'FF', X'FCF4', '0' or an equate symbol with a similar value.

OLD_PRIORITY(name1 | (Rn))

is the previous priority of the issuing task.

- name1** The name of a 1-byte field where the task's previous priority is stored
- (Rn)** A register in which the low-order byte receives the previous priority value and the other bytes are set to zero.

RESPONSE and REASON values for CHANGE_PRIORITY:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	None
INVALID	None
KERNERROR	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "General form of an XPI call" on page 234.

Dump control functions

There are two XPI dump control functions. These are the DFHDUDUX macro calls SYSTEM_DUMP and TRANSACTION_DUMP.

DFHDUDUX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Journal control program
- Transient data program.

The SYSTEM_DUMP call

SYSTEM_DUMP causes a system dump to be taken. If the system dump code that you supply on input is in the system dump code table, the dump may be suppressed. For information about the dump table and how it works, refer to the *CICS/ESA Problem Determination Guide* and the *CICS/ESA System Programming Reference* manual.

SYSTEM_DUMP

```
DFHDUDUX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(SYSTEM_DUMP),
          SYSTEM_DUMPCODE(name8 | string | 'string'),
          [CALLER(block-descriptor),]
          [TITLE(block-descriptor),]]
          [OUT,
          DUMPID(name9 | *),
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

SYSTEM_DUMPCODE(name8 | string | 'string')

is the code corresponding to the error that caused this system dump call. System dump codes are held in the dump table; for information about the dump table and how it works, refer to the *CICS/ESA Problem Determination Guide* and the *CICS/ESA System Programming Reference* manual.

name8 The name of a location containing an 8-byte string.

string A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.

'string' A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the “string” above.

CALLER(block-descriptor)

can be used to identify the source of a system dump request. The information that you supply here appears in the dump header, so you could use it to identify the exit program that initiated the system dump request. For a description of valid block-descriptors, see page 247.

dump control functions

TITLE(block-descriptor)

specifies an area containing the text you want to appear in the dump header when the system dump is printed.

DUMPID(name9 | *)

is the dump identifier.

name9 The name of a 9-byte field to receive the assigned ID.

RESPONSE and REASON values for SYSTEM_DUMP:

RESPONSE	REASON
OK	None
EXCEPTION	FESTAE_FAILED INSUFFICIENT_STORAGE IWMWQWRK_FAILED NO_DATASET PARTIAL_SYSTEM_DUMP SDUMP_BUSY SDUMP_FAILED SDUMP_NOT_AUTHORIZED SUPPRESSED_BY_DUMPOPTION SUPPRESSED_BY_DUMPTABLE SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE INVALID_PROBDESC INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

The TRANSACTION_DUMP call

TRANSACTION_DUMP causes a transaction dump to be taken. If the transaction dump code that you supply on input is in the transaction dump code table, the dump may be suppressed and, optionally, a system dump may be taken. For information about the dump table and how it works, refer to the *CICS/ESA Problem Determination Guide* and the *CICS/ESA System Programming Reference* manual.

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 4, “Writing initialization and shutdown programs” on page 335.

TRANSACTION_DUMP

```
DFHDUDUX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(TRANSACTION_DUMP),
          TRANSACTION_DUMPCODE(name4 | string | 'string')
          [CSA(YES|NO),]
          [PROGRAM(YES|NO),]
          [TCA(YES|NO),]
          [TERMINAL(YES|NO),]
          [TRANSACTION(YES|NO),]
          [TRT(YES|NO),]
          [SEGMENT(block-descriptor),]
          [SEGMENT_LIST(block-descriptor),]]
          [OUT,
          DUMPID(name9 | *),
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

TRANSACTION_DUMPCODE(name4 | string | 'string')

is the code corresponding to the error that caused this transaction dump call. Transaction dump codes are held in the dump table; for information about the dump table and how it works, refer to the *CICS/ESA Problem Determination Guide* and the *CICS/ESA System Programming Reference* manual.

name4 The name of a location containing a 4-byte string.

string A string of characters without intervening blanks. The macro generates a literal constant of length 4 bytes from the string, extending with blanks or truncating as required.

'string' A string, enclosed in quotation marks and possibly containing blanks. This value is processed in the same way as the “string” above.

CSA(YES|NO)

specifies whether the common system area (CSA) is to be included in the transaction dump. The default is NO.

PROGRAM(YES|NO)

specifies whether all program storage areas associated with this task are to be included in the transaction dump. The default is NO.

TCA(YES|NO)

specifies whether the task control area (TCA) is to be included in the transaction dump. The default is NO.

TERMINAL(YES|NO)

specifies whether all terminal storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRANSACTION(YES|NO)

specifies whether all transaction storage areas associated with the task are to be included in the transaction dump. The default is NO.

TRT(YES|NO)

specifies whether the trace table (TRT) is to be included in the transaction dump. The default is NO.

dump control functions

SEGMENT(block-descriptor)

defines the address and the length of a single block of storage that is to be dumped. See page 247 for a description of valid block-descriptors. SEGMENT and SEGMENT_LIST are mutually exclusive.

SEGMENT_LIST(block-descriptor)

specifies the address and length of a set of contiguous word pairs. The first word in each pair specifies the **length** in bytes of a storage segment to be dumped; the second word contains the **address** of the storage segment. The end of the list must be marked by a word containing X'FFFFFFFF'. SEGMENT and SEGMENT_LIST are mutually exclusive.

DUMPID(name9 | *)

is the dump identifier.

name9 The name of a 9-byte field to receive the assigned ID.

RESPONSE and REASON values for TRANSACTION_DUMP:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	INSUFFICIENT_STORAGE FESTAE_FAILED IWMWQWRK_FAILED NOT_OPEN OPEN_ERROR PARTIAL_SYSTEM_DUMP PARTIAL_TRANSACTION_DUMP SDUMP_BUSY SDUMP_FAILED SDUMP_NOT_AUTHORIZED SUPPRESSED_BY_DUMPOPTION SUPPRESSED_BY_DUMPTABLE SUPPRESSED_BY_USEREXIT
DISASTER	None
INVALID	INVALID_DUMPCODE INVALID_PROBDESC INVALID_SVC_CALL
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in "General form of an XPI call" on page 234.
2. 'NOT_OPEN' means that the CICS dump data set is closed.
3. 'OPEN_ERROR' means that an error occurred while a CICS dump data set was being opened.
4. 'PARTIAL' means that the transaction dump resulting from this request is incomplete.

Kernel domain functions

The **START_PURGE_PROTECTION** function

The **START_PURGE_PROTECTION** function is provided on the **DFHKEDSX** macro call. It inhibits purge, but not force-purge, for the current task. This function can be used by all global user exit programs if they want to inhibit purge during a global user exit call.

In general, each **START_PURGE_PROTECTION** call should have a corresponding **STOP_PURGE_PROTECTION** function call to end the purge protection period on completion of any program logic that needs such protection.

START_PURGE_PROTECTION

```
DFHKEDSX [CALL,]
         [CLEAR,]
         [IN,
         FUNCTION(START_PURGE_PROTECTION),]
         [OUT,
         RESPONSE (name1 | *)]
```

There are no input or output parameters on this call, only a **RESPONSE**.

RESPONSE values for START_PURGE_PROTECTION:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	None
INVALID	None

The **STOP_PURGE_PROTECTION** function

The **STOP_PURGE_PROTECTION** function is provided on the **DFHKEDSX** macro call. It re-enables purge for the current task after purge has been suspended by a preceding **START_PURGE_PROTECTION** function call.

STOP_PURGE_PROTECTION

```
DFHKEDSX [CALL,]
         [CLEAR,]
         [IN,
         FUNCTION(STOP_PURGE_PROTECTION),]
         [OUT,
         RESPONSE (name1 | *)]
```

There are no input or output parameters on this call, only a **RESPONSE**.

RESPONSE values for STOP_PURGE_PROTECTION:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	None
INVALID	None

Nesting purge protection calls

Note that the START_ and STOP_PURGE_PROTECTION functions can be nested. You should ensure that, if multiple START_PURGE_PROTECTION calls are issued for a task, that the correct number of STOP_PURGE_PROTECTION calls are issued to cancel the purge protection. If you issue two starts and only one stop, purge protection is left on for the current task.

For example, for any current task, more than one global user exit program may be driven. You must design your exit programs to ensure that purge protection is correctly cancelled. An example of nesting is shown as follows:

```
XEIIIN:  
  EXIT_PROG1: Calls START_PURGE_PROTECTION  
  
  XFCREQ:  
  EXIT_PROG2: Calls START_PURGE_PROTECTION  
  
  XFCREQC:  
  EXIT_PROG3: Calls STOP_PURGE_PROTECTION  
  
XEIOUT:  
  EXIT_PROG4: Calls STOP_PURGE_PROTECTION
```

Loader functions

There are four XPI loader functions. These are the DFHLDLDX calls ACQUIRE_PROGRAM, RELEASE_PROGRAM, DEFINE_PROGRAM, and DELETE_PROGRAM.

DFHLDLDX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Journal control program
- Transient data program.

The DEFINE_PROGRAM call

DEFINE_PROGRAM allows you to define new programs to the loader domain, or to change the details of programs that have already been defined. The details that you provide are recorded on the local catalog, and become immediately available. They are used on all subsequent ACQUIRE requests for the named program. However, note that program definitions made in this way are **not** retained over an XRF takeover.

DEFINE_PROGRAM

```
DFHLDLX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(DEFINE_PROGRAM),
        PROGRAM_NAME(name8 | string | 'string' ),
        [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
        [EXECUTION_KEY(CICS|USER),]
        [PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD),]
        [REQUIRED_AMODE(24|31|AMODE_ANY),]
        [REQUIRED_RMODE(24|RMODE_ANY),]
        [OUT,
        [NEW_PROGRAM_TOKEN(name4),]
        RESPONSE(name1 | *),
        REASON(name1 | *)]
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program to be defined.

- name8** The name of a location where there is an 8-byte program name.
- string** A string of characters, without intervening blanks, naming the program.
- 'string'** A string of characters within quotation marks. The string length is set to 8 by padding with blanks or by truncation.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)

specifies where to load the program from.

PRIVATE The program is in the relocatable program library (RPL). A PRIVATE program need not be reentrant, and is given only limited protection from unauthorized overwriting. The degree of protection depends on the type of dynamic storage area (DSA) into which the program is loaded (see the EXECUTION_KEY option):

DSA	Protection from unauthorized overwriting
ERDSA	Complete—cannot be overwritten by USER tasks or CICS tasks
CDSA	Cannot be overwritten by USER tasks
ECDSA	Cannot be overwritten by USER tasks
UDSA	None
EUDSA	None.

SHARED The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY Either the RPL or the LPA copy of the program may be used, though preference is given to the LPA copy.

EXECUTION_KEY(CICS|USER)

defines, in conjunction with other program attributes, the type of dynamic storage area (DSA) into which the loader is to load the program.

CICS For non-reentrant programs, means that the program is to be loaded into a CICS DSA, above or below the 16MB line; that is, the CDSA or ECDSA. The choice of CICS DSA is dependent on the residence

mode (RMODE) attribute of the program as defined to the linkage-editor.

For reentrant RMODE(24) programs, means that the program is to be loaded into the CDSA.

USER For non-reentrant programs, means that the program is to be loaded into a user DSA, above or below the 16MB line; that is, the UDSA or ERDSA. The choice of user DSA is dependent on the residence mode (RMODE) attribute of the program as defined to the linkage-editor.

For reentrant RMODE(24) programs, means that the program is to be loaded into the UDSA.

Reentrant programs eligible to reside above the 16MB line: If a program is link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into the extended read-only DSA (the ERDSA). For details of the type of storage allocated for the ERDSA, see the RENTPGM system initialization parameter.

See Table 102 for a summary of the effect of the EXECUTION_KEY option in conjunction with other factors.

Table 102. Summary of attributes defining DSA eligibility

EXECUTION_KEY option	Reentrant	Above or below 16MB line	Dynamic storage area (DSA)
CICS	No	Below	CDSA
CICS	Yes	Below	CDSA
CICS	No	Above	ECDSA
CICS	Yes	Above	ERDSA
USER	No	Below	UDSA
USER	Yes	Below	UDSA
USER	No	Above	EUDSA
USER	Yes	Above	ERDSA

PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD)

describes the residency status of the program.

RESIDENT There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quasireentrant.

REUSABLE The program is at least quasireentrant; a single copy in storage can be used by several tasks in the system. A REUSABLE program becomes eligible for removal from storage as part of the normal dynamic program compression scheme when its use count reaches zero.

TRANSIENT Similar to REUSABLE, except that the program is removed from storage immediately its use count reaches zero. This should be specified only for less-frequently used programs, or for programs in systems that are critically short on storage.

RELOAD Every ACQUIRE_PROGRAM request for this program is satisfied by loading a new copy into storage. When a RELEASE request is issued for a copy, it is removed from storage.

Note: Do not use this attribute when defining an exit program.

REQUIRED_AMODE(24|31|AMODE_ANY)

is the addressing mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an 'EXCEPTION' response and the REASON value 'PROGRAM_NOT_FOUND'.

Notes:

1. AMODE_ANY and AMODE 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|RMODE_ANY)

is the residency mode of the program. If, during subsequent ACQUIRE_PROGRAM processing, no copy of the program that meets the defined addressing requirement can be found, the ACQUIRE_PROGRAM call receives an 'EXCEPTION' response and the REASON value 'PROGRAM_NOT_FOUND'.

Note: You cannot use this option to override the link-edited residence mode of the program.

NEW_PROGRAM_TOKEN(name4)

is the token supplied for the newly-defined program.

name4 The name of a location to contain the 4-byte token obtained.

RESPONSE and REASON values for DEFINE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	CATALOG_ERROR
	CATALOG_NOT_OPERATIONAL
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in "General form of an XPI call" on page 234.

The ACQUIRE_PROGRAM call

ACQUIRE_PROGRAM returns the entry and load point addresses, the length, and a new program token for a usable copy of the named program, which can be identified by either its name or a program token.

ACQUIRE_PROGRAM

```
DFHLDLDX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(ACQUIRE_PROGRAM),
          {PROGRAM_NAME(name8 | string | 'string')|
          PROGRAM_TOKEN(name4)},
          [SUSPEND(YES|NO),]
          [OUT,
          [PROGRAM_ATTRIBUTE(name1 | (Rn)),]
          ENTRY_POINT(name4 | (Ra)),
          [LOAD_POINT(name4 | (Ra)),]
          [PROGRAM_LENGTH(name4 | (Rn)),]
          [NEW_PROGRAM_TOKEN(name4),]
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program to be acquired.

- name8** The name of a location containing an 8-byte program name.
- string** A string of characters naming the program.
- 'string'** A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4),

is a token identifying the program whose details are to be acquired.

- name4** The name of a location containing a 4-byte token obtained by a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

SUSPEND(YES|NO)

specifies whether execution is to be suspended until the request can be granted.

PROGRAM_ATTRIBUTE(name1 | (Rn))

is the program attribute returned.

- name1** The name of a 1-byte location to receive the program attribute.
- (Rn)** A register in which the low-order byte receives the program attribute and the other bytes are set to zero. It can have the values RESIDENT, REUSABLE, TRANSIENT, or RELOAD.
 - RESIDENT** There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quaireentrant. Any program of PROGRAM_TYPE SHARED has the RESIDENT attribute by default. The DELETE_PROGRAM call has no effect on this type of RESIDENT program.
 - REUSABLE** Similar to RESIDENT, except that a REUSABLE program that is not in use can be removed from storage by CICS, for storage optimization reasons.

TRANSIENT Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its use count drops to zero.

RELOAD The program is not reusable, and therefore several copies of the program may be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

ENTRY_POINT(name4 | (Ra))

is the program's entry point address.

name4 The name of a 4-byte location to receive the 31-bit entry address

(Ra) A register to receive the entry address.

LOAD_POINT(name4 | (Ra))

is the program's load point address.

name4 The name of a 4-byte location to receive the loaded address

(Ra) A register that is to contain the load address.

PROGRAM_LENGTH(name4 | (Rn))

is the length of the named program.

name4 The name of a 4-byte location that is to receive the length in bytes, expressed in binary

(Rn) A register to contain the length in bytes, expressed in binary.

NEW_PROGRAM_TOKEN(name4)

is the new program token for a usable copy of the named program.

name4 The name of a location to receive a 4-byte token that identifies this program and instance.

RESPONSE and REASON values for ACQUIRE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	NO_STORAGE PROGRAM_NOT_DEFINED PROGRAM_NOT_FOUND
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in "General form of an XPI call" on page 234.
2. A REASON of 'NO_STORAGE' with a RESPONSE of 'EXCEPTION' means that there was insufficient storage to satisfy this request, and SUSPEND(NO) was specified.
3. A REASON of 'PROGRAM_NOT_FOUND' is returned if the program has not been included in the library concatenation, or if the link-edit failed. In such a

case, the program is marked as “not executable”; it must be re-linked before it can be successfully acquired.

The RELEASE_PROGRAM call

RELEASE_PROGRAM decrements the use count of a currently loaded program by one.

If the program has been defined with the RELOAD attribute, the storage occupied by this copy of the program is released.

You should issue the ACQUIRE_PROGRAM and RELEASE_PROGRAM requests for a single program during the same execution of the exit program. If you do not want to do this, you should acquire the program once during CICS initialization, and leave it resident until CICS termination.

RELEASE_PROGRAM

```
DFHLDLX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(RELEASE_PROGRAM),
        ENTRY_POINT(pointer),
        {PROGRAM_NAME(name8 | string | 'string')|
        PROGRAM_TOKEN(name4)},,]
        [OUT,
        RESPONSE(name1 | *),
        REASON(name1 | *)]
```

ENTRY_POINT(pointer)

is the address of the entry point of this copy of the named program.

PROGRAM_NAME(name8 | string | 'string')

is the name of the program to be released.

name8 The name of a location containing an 8-byte program name.
string A string of characters naming the program.
'string' A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4),

is a token identifying the program to be released.

name4 The name of a location containing an 4-byte token obtained by a previous DEFINE_PROGRAM or ACQUIRE_PROGRAM call.

RESPONSE and REASON values for RELEASE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED PROGRAM_NOT_IN_USE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘PROGRAM_NOT_DEFINED’ is returned if the program that you name is not known to the system.
3. ‘PROGRAM_NOT_IN_USE’ is returned when the use count for the named program is already zero.

The DELETE_PROGRAM call

DELETE_PROGRAM removes the definition of a named program from the catalog and from the list of current programs. When this request executes successfully, subsequent ACQUIRE_PROGRAM requests fail with a REASON value of ‘PROGRAM_NOT_DEFINED’.

DELETE_PROGRAM

```
DFHLDLX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(DELETE_PROGRAM),
        PROGRAM_NAME(name8 | string | 'string' ),]
        [OUT,
        RESPONSE(name1 | *),
        REASON(name1 | *)]
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program to be deleted.

name8	The name of a location containing an 8-byte program name.
string	A string of characters naming the program.
'string'	A string in quotation marks. The string length is set to 8 by padding with blanks or truncating.

RESPONSE and REASON values for DELETE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

Monitoring functions

There are two XPI monitoring functions. These are the DFHMNMNX calls MONITOR and INQUIRE_MONITORING_DATA.

DFHMNMNX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program
- Journal control program (DFHJCP).

INQUIRE_MONITORING_DATA calls cannot be used in any exit program invoked from any global user exit point in DFHTCP or DFHZCP (that is, at any of the exit points named “XTCx...” or “XZCx...”).

The MONITOR call

The MONITOR XPI call is similar to the EXEC CICS MONITOR command. It enables you to invoke user event-monitoring points (EMPs) in your exit programs. The user event-monitoring points must be defined in the usual way in the monitoring control table (MCT). For more information about CICS monitoring, read Chapter 19, “CICS monitoring” on page 539.

At a user EMP, you can add your own data (up to 256 counters, up to 256 clocks, and a single character string of up to 256 bytes) to fields reserved unconditionally for you in performance class monitoring data records.

MONITOR

```
DFHMNMNX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(MONITOR),
          [DATA1(name4 | (Ra) | expression | *),]
          [DATA2(name4 | (Rn) | expression | *),]
          [ENTRYNAME(name8 | string | 'string'),]
          POINT(name2 | (Rn) | expression),]
          [OUT,
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITORING_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 4, “Writing initialization and shutdown programs” on page 335.

DATA1(name4 | (Ra) | expression | *)

specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA1 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA1 variable is an area with the address of a series of adjacent fullwords containing the values to be added to the user count fields defined in the user EMP definition.
- If the MCT user EMP definition contains a MOVE option, the DATA1 variable is an area with the address of the character string to be moved.

For details of the user EMP options, see the *CICS/ESA Resource Definition Guide*.

name4	The name of a 4-byte field containing the fullword binary variable for this EMP.
(Ra)	A register containing the fullword binary variable for this EMP.
expression	A valid assembler-language expression giving the fullword binary variable for this EMP.
*	The value of this option is already present in the parameter list, or the option is not specified for this EMP.

DATA2(name4 | (Rn) | expression | *)

specifies a fullword binary variable whose contents depend on the type of user EMP being used:

- If the MCT user EMP definition contains an ADDCNT, SUBCNT, NACNT, EXCNT, or ORCNT option, the DATA2 variable is an area used as defined by the user EMP definition.
- If the MCT user EMP definition contains an MLTCNT option, the DATA2 variable is an area with the number of user count fields to be updated.

The number specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MLTCNT operation from being successful; but, if it is, an exception response of 'DATA2_NOT_SPECIFIED' is returned. See note 5 on page 279.

- If the MCT user EMP definition contains a MOVE option, the DATA2 variable is an area with the length of the character string to be moved.

The length specified in DATA2 overrides the default value defined in the MCT for the operation. A value of 0 instructs monitoring to use the default. Not specifying a value for DATA2 does not prevent the MOVE operation from being successful; but, if it is, an exception response of 'DATA2_NOT_SPECIFIED' is returned. See note 5 on page 279.

For details of the user EMP options, see the *CICS/ESA Resource Definition Guide*.

name4	The name of a 4-byte field containing the fullword binary variable for this EMP.
(Rn)	A register containing the fullword binary variable for this EMP.

monitoring functions

expression A valid assembler-language expression giving the fullword binary variable for this EMP.
* The value of this option is already present in the parameter list, or the option is not specified for this EMP.

ENTRYNAME(name8 | string | 'string')

is the monitoring point entry name, which qualifies the POINT value and which is defined in the monitoring control table (MCT).

name8 The name of a location containing an 8-byte string.
string A string of characters without intervening blanks. The macro generates, from the string, a literal constant of length 8 bytes, extending with blanks or truncating as required.
'string' A string, enclosed in quotation marks, and possibly containing blanks. This value is processed in the same way as the “string” above.

Note: If, when defining the EMP in the MCT, you do not specify an entry name, the entry name defaults to ‘USER’. ENTRYNAME likewise defaults to ‘USER’ if not specified.

POINT(name2 | (Rn) | expression)

specifies the monitoring point identifier as defined in the MCT, and is in the range 0 through 255. Note, however, that point identifiers in the range 200 through 255 are reserved for use by IBM program products.

name2 The name of a 2-byte source of point data
(Rn) A register containing the point data in the low-order 2 bytes
expression A valid assembler-language expression that can be expressed in 2 bytes.

RESPONSE and REASON values for MONITOR:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	POINT_NOT_DEFINED INVALID_DATA1_VALUE INVALID_DATA2_VALUE DATA1_NOT_SPECIFIED DATA2_NOT_SPECIFIED
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘POINT_NOT_DEFINED’ means that the EMP you have specified was not defined in the MCT.
3. ‘INVALID_DATA1_VALUE’ and ‘INVALID_DATA2_VALUE’ are most likely to have been caused by provision of bad addresses; this causes a program check.

4. DATA1_NOT_SPECIFIED and DATA2_NOT_SPECIFIED mean that you have not specified DATA1 or DATA2 respectively when the operation required them. See the description of DATA2.
5. Any error response terminates processing of the EMP. Operations defined to execute before the point of failure will have done so; later operations are canceled.

The INQUIRE_MONITORING_DATA call

The INQUIRE_MONITORING_DATA function returns to the exit program the performance class monitoring data that has been accumulated for the issuing task.

The DFHMNTDS DSECT that maps the data is of fixed format. Note that:

- All the CICS system-defined fields in the performance records (including fields that you have specified for exclusion using the EXCLUDE option of the DFHMCT TYPE=RECORD macro) are listed.
- No user-defined data fields are listed.

INQUIRE_MONITORING_DATA

```
DFHMNMN [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(INQUIRE_MONITORING_DATA),
        DATA_BUFFER(buffer-descriptor),]
        [OUT,
        RESPONSE(name1 | *),
        REASON(name1 | *)]
```

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 4, "Writing initialization and shutdown programs" on page 335.

DATA_BUFFER(buffer-descriptor)

defines the address and the length of a buffer to contain the returned monitoring data; see page 248 for a full definition of a buffer-descriptor. The DSECT DFHMNTDS maps the monitoring data.

RESPONSE and REASON values for INQUIRE_MONITORING_DATA:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	LENGTH_ERROR MONITOR_DATA_UNAVAILABLE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘LENGTH_ERROR’ means that the length specified in the buffer-descriptor was too short for the monitoring data returned from the XPI call.

Program management functions

There are eight XPI program management functions. These are the DFHPGISX calls:

```
INQUIRE_PROGRAM
INQUIRE_CURRENT_PROGRAM
SET_PROGRAM
START_BROWSE_PROGRAM
GET_NEXT_PROGRAM
END_BROWSE_PROGRAM
```

and the DFHPGAQX calls:

```
INQUIRE_AUTOINSTALL
SET_AUTOINSTALL.
```

Used with the Loader functions DEFINE_PROGRAM, ACQUIRE_PROGRAM, RELEASE_PROGRAM, and DELETE_PROGRAM, these calls give you a comprehensive set of tools for manipulating programs. (Note, however, that the tokens returned in the NEW_PROGRAM_TOKEN fields of DFHPGISX calls are different from those returned by DFHLDLDX Loader calls. You should not use a token obtained from a DFHPGISX call in a DFHLDLDX call, or vice versa.)

The INQUIRE_PROGRAM call

INQUIRE_PROGRAM returns information about the attributes of a specified program.

INQUIRE_PROGRAM

```
DFHPGISX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(INQUIRE_PROGRAM),
  {PROGRAM_NAME(name8 | string | 'string')|
  PROGRAM_TOKEN(name4)},]
  [OUT,
  [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
  [HOLD_STATUS(TASK_LIFE|CICS_LIFE|NOT_APPLIC),]
  [LOAD_STATUS(LOADABLE|NOT_LOADABLE|NOT_LOADED|NOT_APPLIC),]
  [INSTALL_TYPE(RDO|CATALOG|GROUPLIST|AUTO|SYSAUTO|MANUAL),]
  [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
  LE370|PLI|NOT_DEFINED|NOT_APPLIC),]
  [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
  COBOL2|LE370|PLI|NOT_DEDUCED|NOT_APPLIC),]
  [AVAIL_STATUS(ENABLED|DISABLED),]
  [MODULE_TYPE(PROGRAM|MAPSET|PARTITIONSET),]
  [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
  [EXECUTION_SET(FULLAPI|DPLSUBSET|NOT_APPLIC),]
  [REMOTE_DEFINITION(REMOTE|LOCAL),]
  [REMOTE_PROGID(name8),]
  [REMOTE_SYSID(name4),]
  [REMOTE_TRANID(name4),]
  [EXECUTION_KEY(CICS|USER|NOT_APPLIC),]
  [NEW_PROGRAM_TOKEN(name4),]
  [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY|NOT_APPLIC),]
  [PROGRAM_USAGE(NUCLEUS|APPLICATION),]
  [PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD),]
  [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED),]
  [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
  [PROGRAM_LENGTH(name4),]
  [PROGRAM_USE_COUNT(name4),]
  [PROGRAM_USER_COUNT(name4),]
  [LOAD_POINT(name4),]
  [ENTRY_POINT(name4),]
  [LOCATION(CDSA|ECDSA|SDSA|ESDSA|RDSA|ERDSA|LPA|ELPA|NONE),]
  [ACCESS(USER|CICS|READ_ONLY|NONE),]
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program to be queried.

- name8** The name of a location containing an 8-byte program name.
- string** A string of characters naming the program.
- 'string'** A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)

is a token identifying the program to be queried.

- name4** The name of a location containing a 4-byte token obtained from a previous INQUIRE_PROGRAM call.

CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC)

is the EDF status of the program.

- CEDF** When the program is running under the control of the CICS execution diagnostic facility (EDF), EDF diagnostic screens are displayed.
- NOCEDF** EDF diagnostic screens are not displayed.
- NOT_APPLIC** Not applicable. This is a mapset, partitionset, or a remote program.

HOLD_STATUS(TASK_LIFE|CICS_LIFE|NOT_APPLIC)

indicates how long the program is to remain loaded.

- TASK_LIFE** The program remains loaded for the lifetime of the task.
- CICS_LIFE** The program remains loaded until CICS is shut down.
- NOT_APPLIC** Not applicable. The program is not loaded, or is remote.

LOAD_STATUS(LOADABLE|NOT_LOADABLE|NOT_LOADED|NOT_APPLIC)

indicates whether or not the program can be loaded.

- LOADABLE** The program is loadable.
- NOT_LOADABLE** CICS has tried to load the program and failed; the program is not in the library.
- NOT_LOADED** CICS has not yet tried to load the program.
- NOT_APPLIC** Not applicable. The program is remote.

INSTALL_TYPE(RDO|CATALOG|GROUPLIST|AUTO|SYSAUTO|MANUAL)

is the method that was used to install the PROGRAM resource definition.

- RDO** RDO commands.
- CATALOG** The CICS global catalog, after a restart.
- GROUPLIST** The CICS startup grouplist.
- AUTO** Autoinstall.
- SYSAUTO** System autoinstall (that is, autoinstalled by CICS without calling the autoinstall user program). The program may be a CICS internal module or, for example, a first phase PLTPI program.
- MANUAL** The program is a CICS internal module explicitly defined to the Program Manager by another CICS component.

LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|LE370|PLI|NOT_DEFINED|NOT_APPLIC)

is the programming language specified on the resource definition.

LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|COBOL2|LE370|PLI|NOT_DEDUCED|NOT_APPLIC)

is the language deduced by CICS for the program.

AVAIL_STATUS(ENABLED|DISABLED)

indicates whether the program can be used—that is, whether or not it has been enabled.

MODULE_TYPE(PROGRAM|MAPSET|PARTITIONSET)

is the kind of program resource.

DATA_LOCATION(ANY|BELOW|NOT_APPLIC)

indicates whether or not the program can access data located above the 16MB line.

- ANY** The program can handle 31-bit addresses, and can therefore be passed data located above or below the 16MB line.
- BELOW** The program can handle only 24-bit addresses, and must therefore only be passed data located below the 16MB line.
- NOT_APPLIC** Not applicable. This is a mapset, partitionset, or a remote program.

EXECUTION_SET(FULLAPI|DPLSUBSET|NOT_APPLIC)

indicates whether CICS links to and runs the program as if it were running in a remote CICS region.

- FULLAPI** CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.
- DPLSUBSET** CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.
- NOT_APPLIC** Not applicable. This is a mapset, partitionset, or a remote program. (The EXECUTIONSET option of DEFINE PROGRAM applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.)

REMOTE_DEFINITION(REMOTE|LOCAL)

indicates whether this program is a local or a remote resource. If it is a remote resource, CICS treats requests to link to the program as distributed program link (DPL) requests, and ships them to the remote region.

REMOTE_PROGID(name8)

is the name by which the program is known in the remote CICS region, if the program is a remote resource. If REMOTESYSTEM was specified on the PROGRAM definition, and REMOTENAME omitted, the remote name will be the same as the local name (that is, REMOTE_PROGID will default to the value of PROGRAM_NAME).

REMOTE_SYSID(name4)

is the name of the remote CICS region that owns the program, if the program is a remote resource.

REMOTE_TRANID(name4)

is the name of the transaction that the remote CICS attaches, and under which it runs the program, if the program is a remote resource.

EXECUTION_KEY(CICS|USER|NOT_APPLIC)

is the key in which CICS gives control to the program, which determines whether the program can modify CICS-key storage.

- CICS** CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA), above or below the 16MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute as defined to the linkage-editor.
- USER** CICS gives control to the program in user key. The program is loaded into a user DSA, above or below the 16MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute as defined to the linkage-editor.

NOT_APPLIC Not applicable. This is a mapset, partitionset, or a remote program.

NEW_PROGRAM_TOKEN(name4)

is a token that can be used to identify the named program.

name4 The name of a location to receive a 4-byte token that identifies this program.

If PROGRAM_NAME is specified on the request, NEW_PROGRAM_TOKEN is set to a program token that can be used on subsequent requests for the same program. If PROGRAM_TOKEN is specified on the request, NEW_PROGRAM_TOKEN is set to the same value.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY|NOT_APPLIC)

is where the next new copy of the program is to be loaded from.

PRIVATE The program is to be loaded from the relocatable program library (RPL). A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED The program is to be loaded from the link pack area (LPA). SHARED programs must be reentrant, and are protected.

The next time a NEWCOPY or PHASEIN is received, an LPA copy of the program is used if it is available. If no LPA version is available, the program is loaded from DFHRPL.

TYPE_ANY Either the RPL or the LPA copy of the program can be used, though preference is given to the LPA copy.

NOT_APPLIC Not applicable. The program is remote.

PROGRAM_USAGE(NUCLEUS|APPLICATION)

indicates whether the program is used as a CICS nucleus program or as a user application program.

PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD)

is the residency status of the program—that is, when its storage is released.

RESIDENT There is a single copy of the program that is not removed from storage unless deleted. RESIDENT programs must be at least quaireentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE Similar to RESIDENT, except that a REUSABLE program that is not in use can be removed from storage by CICS, for storage optimization reasons.

TRANSIENT Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

RELOAD The program is not reusable, and therefore several copies may be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED)

is the addressing mode specified on a DEFINE_PROGRAM call.

SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED)

is the residency mode (that is, whether the program should be loaded above or below the 16MB line) specified on a DEFINE_PROGRAM call.

PROGRAM_LENGTH(name4)

is the length of the program, in bytes, expressed in binary.

PROGRAM_USE_COUNT(name4)

is the number of different users that have invoked the program.

PROGRAM_USER_COUNT(name4)

is the current number of users of the program.

LOAD_POINT(name4)

is the program's load point address, as it would be returned by a Loader domain ACQUIRE_PROGRAM call.

ENTRY_POINT(name4)

is the program's entry point address, as it would be returned by a Loader domain ACQUIRE_PROGRAM call.

LOCATION(CDSA|ECDSA|SDSA|ESDSA|RDSA|ERDSA|LPA|ELPA|NONE)

is where the most recently loaded copy of the program resides.

CDSA	The CICS dynamic storage area
ECDSA	The extended CICS dynamic storage area
SDSA	The shared dynamic storage area
ESDSA	The extended shared dynamic storage area
RDSA	The readonly dynamic storage area
ERDSA	The extended readonly dynamic storage area
LPA	The link pack area
ELPA	The extended link pack area
NONE	The program has not been loaded.

ACCESS(USER|CICS|READ_ONLY|NONE)

indicates the type of storage into which the program has been loaded.

USER	User-key
CICS	CICS-key
READ_ONLY	Readonly
NONE	The program has not been loaded.

RESPONSE and REASON values for INQUIRE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED_TO_LD PROGRAM_NOT_DEFINED_TO_PG
DISASTER	LOCK_ERROR ABEND
INVALID	INVALID_PROGRAM_TOKEN
KERNERROR	None
PURGED	None

The INQUIRE_CURRENT_PROGRAM call

INQUIRE_CURRENT_PROGRAM returns information about the attributes of the program that is currently running. If this call is issued from within a global or task-related user exit, it returns the attributes of the global or task-related user exit program itself.

INQUIRE_CURRENT_PROGRAM

```
DFHPGISX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(INQUIRE_CURRENT_PROGRAM),]
  [OUT,
  [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
  [HOLD_STATUS(TASK_LIFE|CICS_LIFE|NOT_APPLIC),]
  [LOAD_STATUS(LOADABLE|NOT_LOADABLE|NOT_LOADED|NOT_APPLIC),]
  [INSTALL_TYPE(RDO|CATALOG|GROUPLIST|AUTO|SYSAUTO|MANUAL),]
  [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
    LE370|PLI|NOT_DEFINED|NOT_APPLIC),]
  [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
    COBOL2|LE370|PLI|NOT_DEDUCED|NOT_APPLIC),]
  [AVAIL_STATUS(ENABLED|DISABLED),]
  [MODULE_TYPE(PROGRAM|MAPSET|PARTITIONSET),]
  [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
  [EXECUTION_SET(FULLAPI|DPLSUBSET|NOT_APPLIC),]
  [REMOTE_DEFINITION(REMOTE|LOCAL),]
  [REMOTE_PROGID(name8),]
  [REMOTE_SYSID(name4),]
  [REMOTE_TRANID(name4),]
  [EXECUTION_KEY(CICS|USER|NOT_APPLIC),]
  [NEW_PROGRAM_TOKEN(name4),]
  [CURRENT_PROGRAM_NAME(name8),]
  [INVOKING_PROGRAM_NAME(name8),]
  [RETURN_PROGRAM_NAME(name8),]
  [CURRENT_PROGRAM_LENGTH(name4),]
  [CURRENT_CEDF_STATUS(CEDF|NOCEDF),]
  [CURRENT_EXECUTION_SET(FULLAPI|DPLSUBSET),]
  [CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
  [CURRENT_AMODE(24|31),]
  [CURRENT_LOAD_POINT(name4),]
  [CURRENT_ENTRY_POINT(name4),]
  [INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM),]
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call.

CURRENT_PROGRAM_NAME(name8)

is the name of the program that is currently running.

INVOKING_PROGRAM_NAME(name8)

APAR PN67276

Documentation for PN67276 added on 1 March 1995.

+ is the name of the most recent program to invoke the current program, which
 + was not itself a global user exit or task-related user exit program.

+ **RETURN_PROGRAM_NAME(name8)**
 + is the name of the program to which control will be returned, after any
 + intermediate global user exit or task-related user exit programs have completed.

| **CURRENT_PROGRAM_LENGTH(name4)**
 | is the length of the current program, in bytes, expressed in binary.

| **CURRENT_CEDF_STATUS(CEDF|NOCEDF)**
 | is the EDF status of the current instance of the program. The value returned is
 | the same as for CEDF_STATUS, which is the EDF status specified on the
 | program definition. See the CEDF_STATUS option of INQUIRE_PROGRAM.

| **CURRENT_EXECUTION_SET(FULLAPI|DPLSUBSET)**
 | is the API execution set used by the current instance of the program. The
 | value returned is the same as for EXECUTION_SET (which is the API
 | execution set specified on the program definition) *unless* this is the first
 | program in a transaction, when the value may be different. This is because the
 | DPLSUBSET attribute applies only to linked-to programs. It is ignored for the
 | first program in a transaction, because this cannot be the target of a DPL call.
 | Therefore, for the first program in a transaction, if EXECUTION_SET returns
 | DPLSUBSET CURRENT_EXECUTION_SET nevertheless returns FULLAPI.
 | See the EXECUTION_SET option of INQUIRE_PROGRAM.

| **CURRENT_ENVIRONMENT(EXEC|GLUE|PLT|SYSTEM|TRUE|URM)**
 | is the environment in which the current program is running—that is, the type of
 | program it is.

EXEC	User application program
GLUE	Global user exit program
PLT	Program list table program
SYSTEM	CICS system code
TRUE	Task-related user exit program
URM	User-replaceable program

| **CURRENT_AMODE(24|31)**
 | is the addressing mode which the running program is currently using.

| **CURRENT_LOAD_POINT(name4)**
 | is the load point address of the current program.

| **CURRENT_ENTRY_POINT(name4)**
 | is entry point address of the current program.

| **INVOKING_ENVIRONMENT (EXEC|GLUE|PLT|SYSTEM|TRUE|URM)**
 | is the environment from which the current program was invoked. The values
 | are as described for CURRENT_ENVIRONMENT.

RESPONSE and REASON values for INQUIRE_CURRENT_PROGRAM:

RESPONSE	REASON
OK	None
EXCEPTION	NO_CURRENT_PROGRAM
DISASTER	LOCK_ERROR
	ABEND
INVALID	None
KERNERROR	None
PURGED	None

The SET_PROGRAM call

SET_PROGRAM allows you to set selected attributes in the definition of a specified program.

```

SET_PROGRAM
DFHPGISX [CALL,]
    [CLEAR,]
    [IN,]
    FUNCTION(SET_PROGRAM),
    {PROGRAM_NAME(name8 | string | 'string')|
    PROGRAM_TOKEN(name4)},]
    [CEDF_STATUS(CEDF|NOCEDF),]
    [AVAIL_STATUS(ENABLED|DISABLED),]
    [EXECUTION_SET(FULLAPI|DPLSUBSET),]
    [EXECUTION_KEY(CICS|USER),]
    [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY),]
    [PROGRAM_USAGE(NUCLEUS|APPLICATION),]
    [PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD),]
    [REQUIRED_AMODE(24|31|AMODE_ANY),]
    [REQUIRED_RMODE(24|RMODE_ANY),]
    [OUT,]
    RESPONSE(name1 | *),
    REASON(name1 | *)]
    
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program whose attributes are to be changed.

- name8** The name of a location containing an 8-byte program name.
- string** A string of characters naming the program.
- 'string'** A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

PROGRAM_TOKEN(name4)

is a token identifying the program.

- name4** The name of a location containing a 4-byte token obtained from a previous INQUIRE_PROGRAM, INQUIRE_CURRENT_PROGRAM, START_BROWSE_PROGRAM, or GET_NEXT_PROGRAM call.

CEDF_STATUS(CEDF|NOCEDF)

specifies whether, when the program is running under the control of the CICS execution diagnostic facility (EDF), EDF diagnostic screens are displayed.

AVAIL_STATUS(ENABLED|DISABLED)

specifies whether the program can be used—that is, whether or not it is enabled.

EXECUTION_SET(FULLAPI|DPLSUBSET)

specifies whether CICS is to link to and run the program as if it were running in a remote CICS region.

Note: EXECUTION_SET applies only to local program definitions. Its purpose is to test programs in a local CICS environment as if they were running as DPL programs.

FULLAPI CICS links to and runs the program without the API restrictions of a remote DPL program. The program can use the full CICS API.

DPLSUBSET CICS links to and runs the program with the API restrictions of a remote DPL program. The program can use only a subset of the CICS API.

EXECUTION_KEY(CICS|USER)

specifies the key in which CICS is to give control to the program, which determines whether the program can modify CICS-key storage.

CICS CICS gives control to the program in CICS key. The program is loaded into a CICS dynamic storage area (DSA), above or below the 16MB line; that is, the CDSA or ECDSA, depending on its residency mode (RMODE) attribute as defined to the linkage-editor.

USER CICS gives control to the program in user key. The program is loaded into a user DSA, above or below the 16MB line; that is, the UDSA or EUDSA, depending on its residency mode (RMODE) attribute as defined to the linkage-editor.

Note: If the program has been link-edited as reentrant with AMODE(31),RMODE(ANY), the EXECUTION_KEY option is ignored, and it is loaded into the extended readonly DSA (ERDSA). For details of the type of storage allocated for the ERDSA, see the RENTPGM system initialization parameter.

PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY)

specifies where the program is to be loaded from.

PRIVATE The program is in the relocatable program library (RPL). A PRIVATE program need not be reentrant, and is given only limited protection against unauthorized overwriting. The degree of protection depends on the type of dynamic storage area into which the program is loaded (see the description of the PROGRAM_TYPE option of the DEFINE_PROGRAM call).

SHARED The program is located in the link pack area (LPA), is reentrant, and is protected.

TYPE_ANY Either the RPL or the LPA copy of the program can be used, though preference is given to the LPA copy.

PROGRAM_USAGE(NUCLEUS|APPLICATION)

specifies whether the program is used as a CICS nucleus program or as a user application program.

PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD)

specifies the residency status of the program—that is, when its storage is to be released.

RESIDENT At any one time there will be no more than a single copy of the program in storage, and this will not be removed unless deleted. RESIDENT programs must be at least quaireentrant. Any program of PROGRAM_TYPE SHARED is RESIDENT by default.

REUSABLE Similar to RESIDENT, except that a REUSABLE program that is not in use can be removed from storage by CICS, for storage optimization reasons.

TRANSIENT Similar to RESIDENT, except that a TRANSIENT program is removed from storage as soon as its user count drops to zero.

RELOAD The program is not reusable, and therefore several copies may be loaded. A copy is removed from storage when a RELEASE_PROGRAM call (for that copy) is issued.

REQUIRED_AMODE(24|31|AMODE_ANY)

specifies the addressing mode of the program. If, during subsequent processing, no copy of the program that meets the defined addressing requirement can be found, an exception occurs.

Notes:

1. AMODE_ANY and 31 have identical meanings for this function.
2. You cannot use this option to override the link-edited addressing mode of the program.

REQUIRED_RMODE(24|AMODE_ANY)

specifies the residency mode of the program (that is, whether it is to be loaded above or below the 16MB line). If, during subsequent processing, no copy of the program that meets the defined residency requirement can be found, an exception occurs.

Note: You cannot use this option to override the link-edited residency mode of the program.

RESPONSE and REASON values for SET_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED_TO_PG PROGRAM_NOT_DEFINED_TO_LD CEDF_STATUS_NOT_FOR_REMOTE CEDF_STATUS_NOT_FOR_MAPSET CEDF_STATUS_NOT_FOR_PTNSET EXEC_SET_NOT_FOR_REMOTE EXEC_SET_NOT_FOR_MAPSET EXEC_SET_NOT_FOR_PTNSET EXEC_KEY_NOT_FOR_REMOTE

	EXEC_KEY_NOT_FOR_MAPSET
	EXEC_KEY_NOT_FOR_PTNSSET
DISASTER	CATALOG_NOT_OPERATIONAL
	CATALOG_ERROR
	LOCK_ERROR
	ABEND
INVALID	INVALID_PROGRAM_TOKEN
	INVALID_PROGRAM_NAME
	INVALID_MODE_COMBINATION
	INVALID_TYPE_ATTRIB_COMBIN
KERNERROR	None
PURGED	None

The START_BROWSE_PROGRAM call

START_BROWSE_PROGRAM returns a token that enables you to begin browsing through program definitions, optionally starting at the definition of a specified program.

START_BROWSE_PROGRAM

```
DFHPGISX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(START_BROWSE_PROGRAM),
          [PROGRAM_NAME(name8 | string | 'string'),]]
          [OUT,
          BROWSE_TOKEN(name4)
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

PROGRAM_NAME(name8 | string | 'string')

is the name of the program whose definition you want to look at first. The browsing sequence is alphabetical. If there is no program with the specified name, CICS returns a token for the next definition in the alphabetic sequence. If you do not specify a program, CICS returns a token for the first definition.

name8 The name of a location containing an 8-byte program name.
string A string of characters naming the program.
'string' A string of characters in quotation marks. The string length is set to 8 by padding with blanks or truncating.

BROWSE_TOKEN(name4)

is a token to be used on a GET_NEXT_PROGRAM call, to initiate a sequential browse of program definitions.

name4 The name of a location to receive a 4-byte token.

program management functions

RESPONSE and REASON values for START_BROWSE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	INVALID_DIRECTORY LOCK_ERROR ABEND
INVALID	None
KERNERROR	None
PURGED	None

The GET_NEXT_PROGRAM call

GET_NEXT_PROGRAM allows you to inquire on the next program definition during a browse sequence initiated by START_BROWSE_PROGRAM. The browsing sequence is alphabetical. The end of the alphabetic list of definitions is indicated by an 'END_LIST' exception response.

GET_NEXT_PROGRAM

```
DFHPGISX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(GET_NEXT_PROGRAM),
  BROWSE_TOKEN(name4),]
  [OUT,
  PROGRAM_NAME(name8),
  [CEDF_STATUS(CEDF|NOCEDF|NOT_APPLIC),]
  [HOLD_STATUS(TASK_LIFE|CICS_LIFE|NOT_APPLIC),]
  [LOAD_STATUS(LOADABLE|NOT_LOADABLE|NOT_LOADED|NOT_APPLIC),]
  [INSTALL_TYPE(RDO|CATALOG|GROUPLIST|AUTO|SYSAUTO|MANUAL),]
  [LANGUAGE_DEFINED(ASSEMBLER|C370|COBOL|
                    LE370|PLI|NOT_DEFINED|NOT_APPLIC),]
  [LANGUAGE_DEDUCED(ASSEMBLER|C370|COBOL|
                    COBOL2|LE370|PLI|NOT_DEDUCED|NOT_APPLIC),]
  [AVAIL_STATUS(ENABLED|DISABLED),]
  [MODULE_TYPE(PROGRAM|MAPSET|PARTITIONSET),]
  [DATA_LOCATION(ANY|BELOW|NOT_APPLIC),]
  [EXECUTION_SET(FULLAPI|DPLSUBSET|NOT_APPLIC),]
  [REMOTE_DEFINITION(REMOTE|LOCAL),]
  [REMOTE_PROGID(name8),]
  [REMOTE_SYSID(name4),]
  [REMOTE_TRANID(name4),]
  [EXECUTION_KEY(CICS|USER|NOT_APPLIC),]
  [NEW_PROGRAM_TOKEN(name4),]
  [PROGRAM_TYPE(PRIVATE|SHARED|TYPE_ANY|NOT_APPLIC),]
  [PROGRAM_USAGE(NUCLEUS|APPLICATION),]
  [PROGRAM_ATTRIBUTE(RESIDENT|REUSABLE|TRANSIENT|RELOAD),]
  [SPECIFIED_AMODE(24|31|AMODE_ANY|AMODE_NOT_SPECIFIED),]
  [SPECIFIED_RMODE(24|RMODE_ANY|RMODE_NOT_SPECIFIED),]
  [PROGRAM_LENGTH(name4),]
  [PROGRAM_USE_COUNT(name4),]
  [PROGRAM_USER_COUNT(name4),]
  [LOAD_POINT(name4),]
  [ENTRY_POINT(name4),]
  [LOCATION(CDSA|ECDSA|SDSA|ESDSA|RDSA|ERDSA|LPA|ELPA|NONE),]
  [ACCESS(USER|CICS|READ_ONLY|NONE),]
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```

Note: The options not described in the following list are identical to the equivalent options of the INQUIRE_PROGRAM call.

BROWSE_TOKEN(name4)

is a token identifying the definition to be browsed. This can be either the token returned in the NEW_PROGRAM_TOKEN field of the last GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_PROGRAM call).

name4 The name of a location containing a 4-byte token.

program management functions

NEW_PROGRAM_TOKEN(name4)

is a token that identifies the next definition in the browse sequence. You can use it in the BROWSE_TOKEN field of your next GET_NEXT_PROGRAM call (or END_BROWSE_PROGRAM call, if you want to end the sequence). You can also use it in the PROGRAM_TOKEN field of INQUIRE_ and SET_PROGRAM calls.

name4 The name of a location to receive a 4-byte token that identifies the next program definition.

RESPONSE and REASON values for GET_NEXT_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	PROGRAM_NOT_DEFINED_TO_LD INVALID_BROWSE_TOKEN END_LIST
DISASTER	LOCK_ERROR ABEND
INVALID	None
KERNERROR	None
PURGED	None

The END_BROWSE_PROGRAM call

END_BROWSE_PROGRAM allows you to end a browse of program definitions initiated by START_BROWSE_PROGRAM.

END_BROWSE_PROGRAM

```
DFHPGISX [CALL,]  
  [CLEAR,]  
  [IN,  
  FUNCTION(END_BROWSE_PROGRAM),  
  BROWSE_TOKEN(name4),]  
  [OUT,  
  RESPONSE(name1 | *),  
  REASON(name1 | *)]
```

BROWSE_TOKEN(name4)

can be either the token returned in the NEW_PROGRAM_TOKEN field of the last GET_NEXT_PROGRAM call, or that in the BROWSE_TOKEN field of the START_BROWSE_PROGRAM call (this token is updated after every GET_NEXT_PROGRAM call).

RESPONSE and REASON values for END_BROWSE_PROGRAM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	INVALID_BROWSE_TOKEN
DISASTER	LOCK_ERROR
	ABEND
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_AUTOINSTALL call

INQUIRE_AUTOINSTALL returns information about the current settings of the autoinstall function for programs, mapsets, and partitionsets.

INQUIRE_AUTOINSTALL

```
DFHPGAQX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(INQUIRE_AUTOINSTALL),]
          [OUT,
          [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
          [AUTOINSTALL_CATALOG (MODIFY|NONE|ALL),]
          [AUTOINSTALL_EXIT_NAME(name8),]
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

AUTOINSTALL_STATE(ACTIVE|INACTIVE)

is the status of the program autoinstall function.

- ACTIVE** Autoinstall is enabled for programs, mapsets, and partitionsets.
- INACTIVE** Autoinstall is not enabled for programs, mapsets, and partitionsets.

AUTOINSTALL_CATALOG(MODIFY|ALL|NONE)

is the catalog status for autoinstalled program definitions.

- MODIFY** Autoinstalled program, map, and partitionset definitions are recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.
- ALL** All autoinstalled program, map, and partitionset definitions are cataloged.
- NONE** No autoinstalled program, map, or partitionset definitions are cataloged.

AUTOINSTALL_EXIT_NAME(name8)

is the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

RESPONSE and REASON values for INQUIRE_AUTOINSTALL:

RESPONSE	REASON
OK	None
EXCEPTION	None
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The SET_AUTOINSTALL call

SET_AUTOINSTALL enables you to change the settings of the autoinstall function for programs, mapsets, and partitionsets.

```
SET_AUTOINSTALL
DFHPGAQX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(SET_AUTOINSTALL),
          [AUTOINSTALL_STATE (ACTIVE|INACTIVE),]
          [AUTOINSTALL_CATALOG (MODIFY|NONE|ALL),]
          [AUTOINSTALL_EXIT_NAME(name8),]
          [LANGUAGES_AVAILABLE(YES|NO),]
          [OUT,
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

AUTOINSTALL_STATE(ACTIVE|INACTIVE)

specifies the status of the program autoinstall function.

- ACTIVE** Enable autoinstall for programs, mapsets, and partitionsets.
- INACTIVE** Disable autoinstall for programs, mapsets, and partitionsets.

AUTOINSTALL_CATALOG(MODIFY|ALL|NONE)

specifies the catalog status for autoinstalled program definitions.

- MODIFY** Autoinstalled program, map, and partitionset definitions are to be recorded on the CICS global catalog only if they are modified by a SET PROGRAM command after being autoinstalled.
- ALL** All autoinstalled program, map, and partitionset definitions are to be cataloged.
- NONE** No autoinstalled program, map, or partitionset definitions are to be cataloged.

AUTOINSTALL_EXIT_NAME(name8)

specifies the name of the user-replaceable autoinstall control program for programs, mapsets, and partitionsets.

LANGUAGES_AVAILABLE(YES|NO)

specifies whether the autoinstall control program can be called. It can only be called after language establishment.

YES The control program can be called.
NO The control program cannot be called.

RESPONSE and REASON values for SET_AUTOINSTALL:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

State data access functions

The state data access functions allow you to inquire on and set certain system data in the AP domain.

The INQ_APPLICATION_DATA call

The INQ_APPLICATION_DATA call enables you to inquire on application system data in the AP domain.

INQ_APPLICATION_DATA

```
DFHAPIQX [CALL,]
[CLEAR,]
[IN,
FUNCTION(INQ_APPLICATION_DATA),]
[OUT,
[DSA(name4 | (Rn) | * ),]
[EIB(name4 | (Rn) | * ),]
[RSA(name4 | (Rn) | * ),]
[SYSEIB(name4 | (Rn) | * ),]
[TCTUA(name4 | (Rn) | * ),]
[TCTUASIZE(name4 | * ),]
[TWA(name4 | (Rn) | * ),]
[TWASIZE(name4 | (Rn) | * ),]
RESPONSE (name1 | * ),
REASON (name1 | * )]
```

DSA(name4 | (Rn | *)

is the head of the chain of dynamic storage used by application programs to make them reentrant (for example, for assembler programs, the DFHEISTG storage).

name4 The name of a 4-byte area that is to receive the address of the head of the dynamic storage chain.

(Rn) A register that is to receive the DSA address.

* The parameter list itself, in name APIQ_DSA, is used to hold the address.

EIB(name4 | (Rn) | *)

is the address of the EXEC interface block (EIB) for the current task.

name4 The name of a fullword area that is to receive the address of the EIB.

(Rn) A register that is to receive the address of the EIB.

* The parameter list itself, in name APIQ_EIB, is used to hold the address.

RSA(name4 | (Rn | *)

is the address of the register save area for the current task.

name4 The name of a fullword area that is to receive the address of the register save area.

(Rn) A register that is to receive the address of the register save area.

* The parameter list itself, name APIQ_RSA, is used to hold the address.

SYSEIB(name4 | (Rn) | *)

is the address of the system EXEC interface block of the current task.

name4 The name of a fullword area that is to receive the address of the system EXEC interface block.

(Rn) A register that is to receive the address of the system EXEC interface block.

* The parameter list itself, name APIQ_SYSEIB, is used to hold the address.

TCTUA(name4 | (Rn) | *)

is the address of the terminal control table user area (TCTUA) for the current task.

name4 The name of a fullword area that is to receive the address of the TCTUA.

(Rn) A register that is to receive the address of the TCTUA.

* The parameter list itself, name APIQ_TCTUA, is used to hold the address.

TCTUASIZE(name4 | (Rn) | *)

is the length in bytes of the TCTUA for the current task.

name4 The name of a 4-byte area that is to receive the length in bytes of the TCTUA.

(Rn) A register that is to receive the length of the TCTUA.

* The parameter list itself, name APIQ_TCTUASIZE, is used to hold the length of the TCTUA.

TWA(name4 | (Rn) | *)

is the address of the transaction work area.

name4 The name of a fullword area that is to receive the address of the TWA.

(Rn) A register that is to receive the address of the TWA.

* The parameter list itself, name APIQ_TWA, is used to hold the address of the TWA.

TWASIZE(name4 | (Rn) | *)

is the length, in bytes, of the transaction work area (TWA).

name4 The name of a 4-byte area that is to receive the length, in bytes, of the TWA.

(Rn) A register that is to receive the length of the TWA.

* The parameter list itself, name APIQ_TWASIZE, is used to hold the length of the TWA.

RESPONSE and REASON values for INQ_APPLICATION_DATA:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	DPL_PROGRAM NO_TRANSACTION_ENVIRONMENT TRANSACTION_DOMAIN_ERROR
DISASTER	ABEND LOOP INQ_FAILED
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_SYSTEM call

The INQUIRE_SYSTEM call gives you access to CICS system data in the AP domain.

INQUIRE_SYSTEM

```
DFHSAIQX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(INQUIRE_SYSTEM),
  [GMMTEXT(name4),]]
  [OUT,
  [AKP(name4 | *),]
  [CICSREL(name4 | *),]
  [CICSSTATUS(INITIALIZING | ACTIVE | FIRSTQUIESCE |
              FINALQUIESCE),]
  [CICSSYS(name1 | *),]
  [CWA(name4 | (Rn) | *),]
  [CWALENGTH(name2 | *),]
  [DATE(name4|*),]
  [DTRPRGRM(name8 | *),]
  [GMMLENGTH(name2 | *),]
  [GMMTRANID(name4 | *),]
  [INITSTATUS(FIRSTINIT | SECONDINIT | THIRDINIT |
              INITCOMPLETE),]
  [JOBNAME(name8 | *),]
  [OPREL(name4 | *),]
  [OPSYS(name1 | *),]
  [PLTPI(name2 | *),]
  [SECURITYMGR(EXTSECURITY | NOSECURITY),]
  [SHUTSTATUS(SHUTDOWN | CONTROLSHUT | NOSHUTDOWN),]
  [STARTUP(COLDSTART | EMERGENCY | WARMSTART),]
  [TERMURM(name8 | *),]
  [XRFSTATUS(PRIMARY | TAKEOVER | NOXRF),]
  [STARTUPDATE(name4 | *),]
  [TIMEOFDAY(name4 | *),]
  RESPONSE (name1 | * ),
  REASON (name1 | * )]
```

GMMTEXT(name4)

specifies the address of an area of storage, at least 244 bytes in length and owned by the caller, into which CICS is to return the good morning message.

name4 The address of an area of storage that is to receive the good morning message.

Note: The GMMTEXT parameter must follow the IN statement as an input parameter.

AKP(name4 | *)

is the activity keypointing frequency of the CICS region.

name4 The name of a 4-byte location that is to receive the frequency value

CICSREL(name4 | *)

is the release under which the CICS region is running.

name4 The name of a 4-byte location that is to receive the release characters as hexadecimal values.

CICSSTATUS(INITIALIZING|ACTIVE|FIRSTQUIESCE|FINALQUIESCE)

is the status of the CICS region.

INITIALIZING The CICS region is initializing

ACTIVE The CICS region is active and ready to receive work.

FIRSTQUIESCE The CICS region is shutting down, and is in the first stage of quiescing.

FINALQUIESCE The CICS region is shutting down, and is in the final stage of quiescing.

CICSSYS(name1 | *)

is the operating system for which the running CICS has been built.

name1 The name of a 1-byte area that is to receive the hexadecimal character of the operating system. A value of "X" represents MVS/ESA.

CWA(name4 | (Rn) | *)

is the address of the common work area.

name4 The name of a 4-byte field that is to receive the address of the CWA.

(Rn) A register to receive the address of the CWA.

CWALENGTH(name2 | *)

is the length in bytes of the CWA.

name2 The name of a 2-byte field that is to receive the length of the CWA.

DATE(name4 | *)

is today's date in packed-decimal form—4-bytes **0iyydddc**, where:

i is a century indicator. (0=1900, 1=2000, 2=2100, and so on.)

yy=years.

ddd=days.

c is the sign.

name4 The name of a 4-byte location that is to receive the date.

DTRPRGRM(name8 | *)

is the name of the dynamic transaction routing program.

name8 The name of an 8-byte area that is to receive the name of the dynamic transaction routing program.

GMMLENGTH(name2 | *)

is the length in bytes of the "good morning" message.

name2 The name of a 2-byte area that is to receive the length of the good morning message.

GMMTRANID(name4 | *)

is the transaction identifier of the CICS good morning transaction.

name4 The name of a 4-byte area that is to receive the CICS good morning transaction id.

INITSTATUS(FIRSTINIT|SECONDINIT|THIRDINIT|INITCOMPLETE)

indicates the stage reached during CICS initialization.

FIRSTINIT The first stage of CICS initialization.

SECONDINIT The second stage of CICS initialization. This stage corresponds to the period when first phase PLTPI programs are run; that is those programs in a PLT that are defined **before** the DFHDELIM statement.

THIRDINIT The third stage of CICS initialization. This stage corresponds to the period when second phase PLTPI programs are run; that is those programs in a PLT that are defined **after** the DFHDELIM statement.

INITCOMPLETE CICS initialization is complete.

JOBNAME(name8 | *)

is the 8-character MVS job name under which the CICS region is running.

name8 The name of a 8-byte area that is to receive the MVS job name.

OPREL(name2 | *)

is the release number of the currently running operating system

name2 The name of a 2-byte area that is to receive, as a half-word binary value, the release number of the operating system.

OPSYS(name1 | *)

is the type of operating system on which the CICS regions is running.

name1 The name of a 1-byte area that is to receive the hexadecimal character of the operating system on which CICS is running. A value of "X" represents MVS/ESA.

PLTPI(name2 | *)

is the suffix that identifies the program list table (PLT) containing the list of programs to be run during CICS initialization—the program list table post initialization (PLTPI) list.

name2 The name of a 2-byte area that is to receive the suffix.

SECURITYMGR(EXTSECURITY|NOSECURITY)

indicates whether security is active.

EXTSECURITY CICS is using an external security manager (for example, RACF).

NOSECURITY Security is not in use in the CICS region—SEC=NO is specified as a system initialization parameter.

SHUTSTATUS(SHUTDOWN|CONTROLSHUT)

is the shutdown status of the CICS region.

SHUTDOWN CICS is performing an immediate shutdown.

CONTROLSHUT CICS is performing a controlled shutdown; that is, a normal shutdown with a warm keypoint.

NOTSHUTDOWN CICS is not in shutdown mode.

STARTUP(COLDSTART|EMERGENCY|WARMSTART)

is the type of startup the CICS region performed.

- COLDSTART** CICS performed a cold start, either because this was explicitly specified on the system initialization parameter, or because CICS forced a cold start because of the state of the global catalog.
- EMERGENCY** CICS performed an emergency restart because the previous run did not shut down normally with a warm keypoint.
- WARMSTART** CICS performed a warm restart following the normal shutdown of the previous run.

STARTUPDATE(name4 | *)

is the start-up-date of this CICS region, in packed decimal form (4-bytes **00yydddc** where **yy**=years, **ddd**=days, **c** is the sign).

- name4** The name of a 4-byte location that is to receive the startup date of this CICS system.

TERMURM(name8 | *)

is the name of the autoinstall user program for terminals.

- name8** The name of an 8-byte area that is to receive the name of the autoinstall user program for terminals.

TIMEOFDAY(name4 | *)

is the current time-of-day in packed decimal form (4-bytes **hhmmssct** where **hh**=hours, **mm**=minutes, **ss**=seconds, **t**=tenths of a second, and **c** is the sign).

- name4** The name of a 4-byte location that is to receive the time.

XRFSTATUS(PRIMARY|TAKEOVER|NOXRF)

is the XRF status of the CICS region.

- PRIMARY** The CICS region was started as an active CICS in an XRF environment.
- TAKEOVER** The CICS region was started as an alternate CICS, with the **START=STANDBY** system initialization parameter.
- NOXRF** CICS was started with the system initialization parameter **XRF=NO** specified. XRF is not active.

RESPONSE and REASON values for INQUIRE_SYSTEM

<i>RESPONSE</i>	<i>REASON</i>
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	UNKNOWN_DATA
	LENGTH_ERROR
DISASTER	INQ_FAILED
PURGED	None

The SET_SYSTEM call

The SET_SYSTEM call allows you to set CICS system data values in the AP domain.

SET_SYSTEM

```
DFHSAIQX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(SET_SYSTEM),
  [AKP(name4 | (Rn) ),]
  [DTRPRGRM(name8 | string | 'string'),]
  [GMMLNGTH(name2 | (Rn) | expression),]
  [GMMTEXT(name8 | (Rn)),]]
  [OUT,
  RESPONSE (name1 | * ),
  REASON (name1 | * )]
```

AKP(name4 | (Rn))

specifies the activity keypointing frequency of the CICS region.

name4 The name of a 4-byte location that contains the new frequency value.

(Rn) A register that contains the new frequency value.

DTRPRGRM(name8 | string | 'string')

specifies the name of the dynamic transaction routing program.

name8 The name of an 8-byte area that contains the name of the dynamic transaction routing program.

string A string of character, without intervening blanks, that defines the name of the dynamic transaction routing program being set.

'string' A string of character without intervening blanks. If you want to document a name (label) in your program, use this form.

GMMLNGTH(name2 | (Rn))

specifies the length of the new "good morning" message supplied by the GMMTEXT parameter.

name2 The name of a 2-byte area that contains, as a half-word binary value, the length of the new good morning message.

(Rn) A register that contains the length of the new good morning message.

GMMTEXT(name4 | (Rn))

specifies the new good morning message.

name4 The name of a 4-byte location that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

(Rn) A register that contains the address of a storage area (up to a maximum of 246 bytes long) that contains the good morning message.

RESPONSE and REASON values for SET_SYSTEM:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
INVALID	INVALID_FUNCTION
EXCEPTION	AKP_SIZE_ERROR
	NO_KEYPOINT
DISASTER	SET_FAILED
PURGED	None

Storage control functions

There are seven XPI storage control functions. These are the DFHSMCX macro calls GETMAIN, FREEMAIN, INQUIRE_ELEMENT_LENGTH, and INQUIRE_TASK_STORAGE, and the DFHSMRX calls INQUIRE_ACCESS, INQUIRE_SHORT_ON_STORAGE, and SWITCH_SUBSPACE.

DFHSMCX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Journal control program
- Transient data program.

The GETMAIN call

GETMAIN acquires an element of storage for use by your exit program. You can ask for a particular CLASS of storage, and you can request that it be initialized to a single-byte value.

Storage in the following classes, acquired by a GETMAIN call, is released by CICS when the TCA being used at the time of the acquisition terminates:

CICS24
CICS
USER24
USER.

In contrast, storage in the following classes is **not** released automatically at task-end: you should use the FREEMAIN call to release it:

SHARED_CICS24
SHARED_CICS
SHARED_USER24
SHARED_USER
TERMINAL.

In addition, some user exits may be invoked from system tasks, and in these circumstances storage is not released until the next CICS shutdown. Therefore you should use FREEMAIN to release all storage areas acquired by GETMAIN as soon as you have finished using them.

GETMAIN

```
DFHSMCX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(GETMAIN),
  GET_LENGTH(name4 | (Rn) | expression),
  STORAGE_CLASS(TERMINAL|CICS24|CICS|USER24|USER|SHARED_CICS24|
                SHARED_CICS|SHARED_USER24|SHARED_USER),
  SUSPEND(YES|NO),
  [INITIAL_IMAGE(name1 | literalconst),]
  [TCTTE_ADDRESS(name4 | (Ra)),]]
  [OUT,
  ADDRESS(name4 | (Rn) | *),
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```

GET_LENGTH(name4 | (Rn) | expression)

specifies the number of bytes of storage you want, expressed in any of the following ways:

- name4** The name of a fullword specifying, in binary, the number of bytes
- (Rn)** A register containing, in binary, the number of bytes
- expression** A valid assembler-language expression; for instance, a number, a symbolic expression, or a combination of the two.

If you request **TERMINAL** storage, the length you specify should not include the length of the storage accounting area (SAA), and the maximum length you can specify is 65 515 bytes. **CICS** storage management adds an 8-byte SAA, and the address returned by the XPI call is that of the start of the SAA.

If you request **CICS24**, **CICS**, **USER24**, **USER**, **SHARED_CICS24**, **SHARED_CICS**, **SHARED_USER24**, or **SHARED_USER** storage, you need only specify the length needed by your program. The address returned is that of the start of your data storage. The maximum size of storage for these storage classes is the same as the size of the DSA from which they are allocated.

STORAGE_CLASS(TERMINAL|CICS24|CICS|USER24|USER|SHARED_CICS24|SHARED_CICS|SHARED_USER24|SHARED_USER)

is the class of the storage that is the subject of the call. The values you can assign to this option, and the type of storage each represents, are listed in Table 103 on page 307.

<i>Table 103. CICS storage classes</i>	
STORAGE_CLASS	Type of storage
TERMINAL	This class of storage has an 8-byte SAA.
CICS24	Task-lifetime CICS-key storage below 16MB
CICS	Task-lifetime CICS-key storage above 16MB
USER24	Task-lifetime user-key storage below 16MB
USER	Task-lifetime user-key storage above 16MB
SHARED_CICS24	Shared CICS-key storage below 16MB
SHARED_CICS	Shared CICS-key storage above 16MB
SHARED_USER24	Shared user-key storage below 16MB
SHARED_USER	Shared user-key storage above 16MB

You must specify a storage class on a GETMAIN request. On a FREEMAIN request it is an optional parameter, and any value that you specify is not checked by CICS.

SUSPEND(YES|NO)

specifies whether to suspend your request if there is less storage available than you have asked for on the GET_LENGTH option.

INITIAL_IMAGE(name1 | literalconst)

specifies the initializing pattern. For example, you might want to set the acquired storage to binary zeros.

name1 The name of a location where the one-byte initializing pattern is stored

literalconst A number in the form of a literal, for example B'00000000', X'FF', X'FC', '0', or an equate symbol with a similar value.

TCTTE_ADDRESS(name4 | (Ra))

is the address of the terminal control table terminal entry (TCTTE). On GETMAIN requests, you must code this option if, on the STORAGE_CLASS option, you specify a class of TERMINAL. On FREEMAIN requests, you must code it if you are freeing TERMINAL-class storage.

Note: Before obtaining TERMINAL class storage, check TCAFCI bit 7 to ensure that the TCA is running under a terminal.

name4 The name of a fullword containing the address
(Ra) A register that points to the TCTTE.

ADDRESS(name4 | (Rn) | *)

is the address of the storage obtained by the call.

name4 The name of a fullword where the obtained storage address is saved

(Rn) A register that is set to point to the obtained storage

***** The parameter list itself, name SMMC_ADDRESS, is used to keep the address.

RESPONSE and REASON values for GETMAIN:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	INSUFFICIENT_STORAGE
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Notes:

1. For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.
2. ‘INSUFFICIENT_STORAGE’ is returned if the GETMAIN request was specified with SUSPEND(NO), and there was not enough storage available to satisfy the request.
3. ‘PURGED’ is returned if the GETMAIN request was specified with SUSPEND (YES), there was not enough storage to satisfy the request, and the task was purged.

The FREEMAIN call

FREEMAIN releases an area of storage that is currently allocated to your exit program.

```
— FREEMAIN —  
DFHSMCX [CALL,]  
  [CLEAR,]  
  [IN,  
  FUNCTION(FREEMAIN),  
  ADDRESS(name4 | (Rn) | *),  
  [STORAGE_CLASS(TERMINAL|CICS24|CICS|USER24|USER|SHARED_CICS24|  
  SHARED_CICS|SHARED_USER24|SHARED_USER),]  
  [TCTTE_ADDRESS(pointer),]  
  [OUT,  
  RESPONSE(name1 | *),  
  REASON(name1 | *)]
```

The explanation of the options is the same as that given above for the GETMAIN function.

RESPONSE and REASON values for FREEMAIN:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	None
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

The INQUIRE_ACCESS call

INQUIRE_ACCESS returns the access-key of an element of storage specified by start address and length. If the element is not wholly contained within one of the CICS dynamic storage areas (DSAs), CICS returns an exception response.

INQUIRE_ACCESS

```
DFHMSRX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(INQUIRE_ACCESS),
  ELEMENT_ADDRESS(name4 | (Rn) | *),
  ELEMENT_LENGTH(name4 | (Rn) | *),]
  [OUT,
  ACCESS(CICS | USER | READ_ONLY),
  RESPONSE(name1 | *),
  REASON(name1 | *)]
```

ELEMENT_ADDRESS(name4 | (Rn) | *)

specifies the address of the storage element.

ELEMENT_LENGTH(name4 | (Rn) | *)

specifies the length of the storage element, in bytes. A length of zero is treated as a length of one.

ACCESS(CICS | USER | READ_ONLY)

is the access-key of the storage element.

CICS	CICS-key
USER	User-key
READ_ONLY	Readonly storage.

RESPONSE and REASON values for INQUIRE_ACCESS:

RESPONSE	REASON
OK	None
EXCEPTION	INVALID_ELEMENT
DISASTER	None
INVALID	None
KERNERROR	None

The INQUIRE_ELEMENT_LENGTH call

The INQUIRE_ELEMENT_LENGTH call's purpose is to enable the caller to pass the address of any part of an element of task-lifetime storage, and to obtain from CICS the start address and the length of the storage element that contains the passed address.

INQUIRE_ELEMENT_LENGTH

```
DFHSMCX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION (INQUIRE_ELEMENT_LENGTH),
        ADDRESS (name4 | (Rn) | *),]
        [OUT,
        ELEMENT_ADDRESS(name4 | (Rn) | *),
        ELEMENT_LENGTH(name4 | (Rn) | *),
        RESPONSE (name1 | *),
        REASON (name1 | *)]
```

ADDRESS(name4 | (Rn) | *)

specifies an address that lies within an element of task-lifetime storage of the current task.

CICS accepts addresses that reference the leading or trailing check zones as being valid addresses for the element of storage you are inquiring upon.

ELEMENT_ADDRESS(name4 | (Rn) | *)

returns the start address of the element of task-lifetime storage referenced by the ADDRESS parameter. The start address returned does **not** include the leading check zone.

ELEMENT_LENGTH(name4 | (Rn) | *)

returns the length of the element of task-lifetime storage referenced by the ADDRESS parameter. The length returned does **not** include the leading or trailing check zones.

RESPONSE and REASON values for INQUIRE_ELEMENT_LENGTH:

RESPONSE	REASON
OK	None
EXCEPTION	INVALID_ADDRESS
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The INQUIRE_SHORT_ON_STORAGE call

The INQUIRE_SHORT_ON_STORAGE call's purpose is to enable the caller to determine whether CICS is short on storage either above or below the 16MB line.

INQUIRE_SHORT_ON_STORAGE

```
DFHMSRX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(INQUIRE_SHORT_ON_STORAGE),]
        [OUT,
        SOS_BELOW_THE_LINE(YES|NO),
        SOS_ABOVE_THE_LINE(YES|NO),
        RESPONSE (name1 | *),
        REASON (name1 | *)]
```

SOS_BELOW_THE_LINE(YES|NO),

returns YES if CICS is currently short-on-storage in any of the DSAs below the 16MB line, and NO if not.

SOS_ABOVE_THE_LINE(YES|NO),

returns YES if CICS is currently short-on-storage in any of the DSAs above the 16MB line, and NO if not.

RESPONSE and REASON values for INQUIRE_SHORT_ON_STORAGE:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	None
KERNERROR	None

The INQUIRE_TASK_STORAGE call

The INQUIRE_TASK_STORAGE call's purpose is to enable the caller to request details of all elements of task-lifetime storage belonging to a task. You can specify the transaction number of the task explicitly on the call, or let it default to the current task.

INQUIRE_TASK_STORAGE

```
DFHSMCX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION (INQUIRE_TASK_STORAGE),
        [TRANSACTION_NUMBER(name4 | (Rn) | *),]
        ELEMENT_BUFFER(buffer-descriptor),
        LENGTH_BUFFER(buffer-descriptor),]
        [OUT,
        NUMBER_OF_ELEMENTS(name4 | (Rn) | *),
        RESPONSE (name1 | *),
        REASON (name1 | *)]
```

ELEMENT_BUFFER(buffer-descriptor)

defines the address and length of a buffer into which CICS returns a list of start addresses of all the elements of task-lifetime storage belonging to either the specified task or, by default, the current task.

The start addresses returned do **not** include the leading check zone.

storage control functions

LENGTH_BUFFER(buffer-descriptor)
defines the address and length of a buffer into which CICS returns a list of the
lengths of the elements of task-lifetime storage belonging to either the specified
task or, by default, the current task. The lengths returned do **not** include the
leading or trailing check zones.

| **NUMBER_OF_ELEMENTS(name4 | (Rn) | *)**
| returns the number of entries in each of the two buffers, ELEMENT_BUFFER
| and LENGTH_BUFFER, as a full-word binary value.

| **TRANSACTION_NUMBER(name4 | (Rn) | *)**
| specifies, as a 4 byte packed decimal value, the transaction number of the task
| to whom the storage belongs.

| If you omit the transaction (task) number, CICS assumes the current task.

RESPONSE and REASON values for INQUIRE_TASK_STORAGE:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	INSUFFICIENT_STORAGE NO_TRANSACTION_ENVIRONMENT
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

The SWITCH_SUBSPACE call

The SWITCH_SUBSPACE call's purpose is to cause CICS to switch from a subspace to base space, if the task is not already executing in the base space. If the task is already in the base space, storage manager ignores the call.

This function can be used by global user exit programs that receive control in subspace and for some reason need to switch into basespace.

SWITCH_SUBSPACE

```
DFHMSRX [CALL,]  
        [CLEAR,]  
        [IN,  
        FUNCTION(SWITCH_SUBSPACE),  
        SPACE(BASESPACE),]  
        [OUT,  
        RESPONSE (name1 | *),  
        REASON (name1 | *)]
```

SPACE(BASESPACE)

Issue a switch to the basespace if the task issuing the call is currently executing within a subspace. This enables the task to read and write to another task's user-key task-lifetime storage.

RESPONSE and REASON values for SWITCH_SUBSPACE:

RESPONSE	REASON
OK	None
DISASTER	None
KERNERROR	None

Trace control function

There is one XPI trace control function. This is the DFHTRPTX call TRACE_PUT.

DFHTRPTX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Journal control program
- Transient data program.

The TRACE_PUT call

TRACE_PUT writes a trace entry to the active trace destinations. You should only make a TRACE_PUT call when UEPTRON indicates that tracing is active for the function containing the exit program (see UEPTRON in DFHUEPAR). You may prefer to make “exception” trace entries, in case of serious errors, without testing UEPTRON.

If you use TRACE_PUT to write exception trace entries, you should identify these so they are highlighted as exception trace entries by the trace formatting utility program. To identify an exception trace entry, enter the literal string ‘USEREXC’ in the DATA1 block descriptor field on the DFHTRPTX call. See the *C/CS/ESA Problem Determination Guide* for details of how an exception trace entry is interpreted.

TRACE_PUT

```
DFHTRPTX [CALL,]
  [CLEAR,]
  [IN,
  FUNCTION(TRACE_PUT),
  POINT_ID(name2 | (Rn) | literalconst),
  [DATA1(block-descriptor),]
  [DATA2(block-descriptor),]
  [DATA3(block-descriptor),]
  [DATA4(block-descriptor),]
  [DATA5(block-descriptor),]
  [DATA6(block-descriptor),]
  [DATA7(block-descriptor),]
  [RETURN_ADDR(name4 | (Ra) | expression),]
  [OUT,
  RESPONSE(name1 | *)]
```

POINT_ID(name2 | (Rn) | literalconst)

identifies the trace entries made as a result of this request. Every TRACE_PUT call within a calling domain should have a unique POINT_ID to locate the origin of a trace call when examining a formatted trace. The POINT_IDs must be in the range decimal 256 through 511 (X'100' through X'1FF'). This range is not used in CICS modules, but is reserved for user exits.

name2 The name of a 2-byte field containing the ID
(Rn) A register, the two low-order bytes of which contain the ID
literalconst A number in the form of a literal, containing the ID.

DATA n (block-descriptor)

define up to seven areas to be included in the data section of the trace entry. For a description of valid block-descriptors, see page 247. If you specify any given DATA n , then DATA1 through DATA($n-1$) **must** be coded before DATA n . The specified DATA items are printed in the trace output in the order specified, that is, in order of DATA1 through DATA n . A 2-byte length field is printed before the data field itself. The maximum total length of the data that can be traced in one call is $4040 - (2 * n)$ bytes, where n is the number of data fields that you specify.

RETURN_ADDR(name4 | (Ra) | expression)

is the value that appears in the return address field of the trace entry.

name4 The name of a fullword containing the address
(Ra) A register containing the address
expression A valid assembler-language expression that results in the address.

RESPONSE values for TRACE_PUT: The RESPONSE field is never set for the TRACE_PUT function. This is for performance reasons. It is not considered that any useful purpose could be served by testing for this value. Note, however, that the syntax requires that RESPONSE is always specified as a parameter on the call. It is recommended that RESPONSE(*) is always used.

Transaction management functions

This section describes the transaction management XPI calls.

The INQUIRE_DTRTRAN call

The INQUIRE_DTRTRAN call returns the name of the dynamic transaction routing (DTR) transaction definition.

The DTR transaction definition provides common attributes for transactions that are to be dynamically routed and which do not have a specific transaction definition. It is specified on the DTRTRAN system initialization parameter; the CICS-supplied default definition is CRTX.

INQUIRE_DTRTRAN

```
DFHXMSRX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(INQUIRE_DTRTRAN),]
          [OUT,
          DTRTRAN(name4),
          RESPONSE (name1 | *),
          REASON (name1 | *)]
```

DTRTRAN(name4)

is the name of the DTR transaction definition to used for routing transactions that are not defined by an explicit transaction resource definition.

name4 The name of a 4-byte location that is to receive the name of the DTR transaction definition.

APAR PN70976

Documentation for PN70976 added on 29 June 1995.

If 'NO' was specified on the DTRTRAN system initialization parameter, 'NO' will be placed in this field.

RESPONSE and REASON values for INQUIRE_DTRTRAN:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	LOGIC_ERROR
	ABEND
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_MXT call

The INQUIRE_MXT function is provided on the DFHXMSRX macro call. Its purpose is to provide current value of the MXT parameter.

INQUIRE_MXT

```
DFHXMSRX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(INQUIRE_MXT),]
          [OUT,
          MXT_LIMIT(name4 | (Rn)),
          CURRENT_ACTIVE(name4 | (Rn) ),
          MXT_QUEUED(name4 | (Rn) ),
          TCLASS_QUEUED(name4 | (Rn) ),
          RESPONSE (name1 | *),
          REASON (name1 | *)]
```

transaction management functions

MXT_LIMIT(name4 | (Rn))

is the current number of the MXT parameter.

name4 The name of a 4-byte location that is to receive the maximum number of all user tasks currently allowed, expressed as a binary value.

(Rn) A register to receive the maximum number of all tasks currently allowed, expressed as a binary value.

CURRENT_ACTIVE(name4 | (Rn))

is the current number of all active user tasks.

name4 The name of a 4-byte location that is to receive the current number of active user tasks, expressed as a binary value.

(Rn) A register to receive the current number of active user tasks, expressed as a binary value.

MXT_QUEUED(name4 | (Rn))

is the current number of user transactions that are queued as a result of the maximum tasks (MXT) being reached.

name4 The name of a 4-byte location that is to receive the current number of queued user tasks, expressed as a binary value.

(Rn) A register to receive the current number of queued user tasks, expressed as a binary value.

TCLASS_QUEUED(name4 | (Rn))

is the current number of all transactions that are queued for transaction class membership.

name4 The name of a 4-byte location that is to receive the current number of queued transaction class members, expressed as a binary value.

(Rn) A register to receive the current number of queued transaction class members, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_MXT:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	LOGIC_ERROR
	ABEND
	LOOP
INVALID	INVALID_FUNCTION
KERNERROR	None
PURGED	None

The INQUIRE_TCLASS call

The INQUIRE_TCLASS function is provided on the DFHXMCLX macro call. Its purpose is to provide current information about the specified transaction class (TCLASS).

INQUIRE_TCLASS

```
DFHXMCLX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(INQUIRE_TCLASS),
          INQ_TCLASS_NAME(name8 | string
          | 'string'),]
          [OUT,
          [MAX_ACTIVE(name4 | (Rn)),]
          [PURGE_THRESHOLD(name4 | (Rn)),]
          [CURRENT_ACTIVE(name4 | (Rn)),]
          [CURRENT_QUEUED(name4 | (Rn)),]
          RESPONSE (name1 | *),
          REASON (name1 | *)]
```

INQ_TCLASS_NAME(name8 | string | 'string')

specifies the name of the transaction class for this inquiry.

- name8** The name of an 8-byte location that contains the name of the transaction class being inquired on.
- string** A string of characters, without intervening blanks, naming the transaction class.
- 'string'** A string of characters, within quotation marks, naming the transaction class. The string length is set to 8 by padding with blanks within the quotation marks.

MAX_ACTIVE(name4 | (Rn))

is the current maximum number of active tasks allowed for the transaction class.

- name4** The name of a 4-byte location that is to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.
- (Rn)** A register to receive the current maximum number of active tasks currently allowed for this transaction class, expressed as a binary value.

PURGE_THRESHOLD(name4 | (Rn))

is the purge threshold limit for this transaction class.

- name4** The name of a 4-byte location that is to receive the current purge threshold limit for this transaction class, expressed as a binary value.
- (Rn)** A register to receive the current purge threshold limit for this transaction class, expressed as a binary value.

transaction management functions

CURRENT_ACTIVE(name4 | (Rn))

is the current number of active user tasks in this transaction class.

name4 The name of a 4-byte location that is to receive the current number of active user tasks for this transaction class, expressed as a binary value.

(Rn) A register to receive the current number of active user tasks for this transaction class, expressed as a binary value.

CURRENT_QUEUED(name4 | (Rn))

is the current number of queued user tasks.

name4 The name of a 4-byte location that is to receive the current number of queued user tasks in this transaction class, expressed as a binary value.

(Rn) A register to receive the current number of queued user tasks, expressed as a binary value.

RESPONSE and REASON values for INQUIRE_TCLASS:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	LOGIC_ERROR
INVALID	None
EXCEPTION	UNKNOWN_CLASS

The INQUIRE_TRANDEF call

The INQUIRE_TRANDEF function is provided on the DFHXMIDX macro call. Its purpose is to allow you to obtain information about the specified transaction definition. In general, this function call is equivalent to the EXEC CICS INQUIRE TRANSACTION command, with some differences.

INQUIRE_TRANDEF

```

DFHMXDX [CALL,]
        [CLEAR,]
        [IN,]
        FUNCTION(INQUIRE_TRANDEF),
        INQ_TRANSACTION_ID(name4 | string
        | 'string'),]
        [OUT,]
        [TRANSACTION_ID(name4),]
        [INITIAL_PROGRAM(name8),]
        [PROFILE_NAME(name8),]
        [TRAN_PRIORITY(name4 | (Rn)),]
        [TASIZE(name4 | (Rn)),]
        [STATUS(name1),]
        [PARTITIONSET(name1),]
        [PARTITIONSET_NAME(name8),]
        [TASKDATAKEY(name1),]
        [TASKDATALOC(name1),]
        [STORAGE_CLEAR(name1),]
        [SYSTEM_RUNAWAY(name1),]
        [RUNAWAY_LIMIT(name4 | (Rn)),]
        [DYNAMIC(name1),]
        [REMOTE_SYSTEM(name4),]
        [REMOTE_NAME(name8),]
        [TRAN_ROUTING_PROFILE(name8),]
        [LOCAL_QUEUEING(name1),]
        [TCLASS(name1), [TCLASS_NAME(name8),]]
        [DTIMEOUT(name4 | (Rn)),]
        [INDOUBT(name1),]
        [RESTART(name1),]
        [SPURGE(name1),]
        [TPURGE(name1),]
        [DUMP(name1),]
        [TRACE(name1),]
        [SHUTDOWN(name1),]
        [RESSEC(name1),]
        [CMDSEC(name1),]
        [STORAGE_FREEZE(name1),]
        [REMOTE(name1),]
        [ISOLATE(name1),]
        [SYSTEM_ATTACH(name1),]
        RESPONSE (name1 | *),
        REASON (name1 | *)]

```

The following parameter descriptions explain briefly the possible values that can be returned on an INQUIRE_TRANDEF call. For a more detailed explanation of some of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in the *CICS/ESA Resource Definition Guide*.

transaction management functions

INQ_TRANSACTION_ID(name4 | string | 'string')

specifies the transaction identifier for this transaction definition inquiry.

name4 The name of a 4-byte location that contains the name of the transaction identifier.

string A string of characters, without intervening blanks, naming the transaction identifier.

'string' A string of characters, within quotation marks, naming the transaction identifier. The string length is set to 4 by padding with blanks within the quotation marks.

TRANSACTION_ID(name4)

is the primary transaction identifier for this transaction definition inquiry.

name4 The name of a 4-byte location that contains the name of the transaction identifier.

INITIAL_PROGRAM(name8)

is the name of the initial program to be given control for the transaction.

name8 The name of an 8-byte location to receive the initial program name.

PROFILE_NAME(name8)

is the name of the profile definition that is associated with the transaction definition.

name8 The name of an 8-byte location to receive the name of the profile definition associated with the transaction definition.

TRAN_PRIORITY(name4 | (Rn))

is the transaction priority specified on the transaction definition.

name4 The name of a 4-byte location to receive the transaction priority, expressed as a binary value.

(Rn) A register to receive the transaction priority, expressed as a binary value.

TWASIZE(name4 | (Rn))

is the size of the transaction work area specified on the transaction definition.

name4 The name of a 4-byte location to receive the size of the transaction work area, expressed as a binary value.

(Rn) A register to receive the size of the transaction work area, expressed as a binary value.

STATUS(name1)

indicates, in a 1-byte location (*name1*), the status of the transaction definition.

ENABLED The transaction definition is enabled.

DISABLED The transaction definition is disabled.

PARTITIONSET(name1)

indicates, in a 1-byte location (*name1*), the partitionset specified on the transaction definition.

- NONE** There is no partitionset specified for the transaction definition.
- KEEP** The reserved name KEEP is specified for the partitionset, which means tasks running under this transaction definition use the application partitionset for the terminal associated with the transaction.
- OWN** The reserved name OWN is specified for the partitionset, which means tasks running under this transaction definition perform their own partitionset management.
- NAMED** The partitionset is named specifically on the transaction definition. The name is returned on the PARTITIONSET_NAME parameter.

PARTITIONSET_NAME(name8)

is the name of the partitionset defined on the transaction definition.

- name8** The name of an 8-byte location that is to receive the name of the partitionset.

TASKDATAKEY(name1)

indicates, in a 1-byte location (*name1*), the storage key of task-lifetime storage for tasks associated with this transaction definition.

- CICS** CICS key is specified for task-lifetime storage.
- USER** USER key is specified for task-lifetime storage.

TASKDATALOC(name1)

indicates, in a 1-byte location (*name1*), the data location of task-lifetime storage for tasks associated with this transaction definition.

- ANY** Task-lifetime storage can be located above 16MB in virtual storage.
- BELOW** Task-lifetime storage must be located below 16MB in virtual storage.

STORAGE_CLEAR(name1)

indicates, in a 1-byte location (*name1*), whether task-lifetime storage, of tasks associated with this transaction definition, is to be cleared before it is freed by a FREEMAIN command.

- YES** Task-lifetime storage must be cleared before it's freed.
- NO** Task-lifetime storage need not be cleared before it's freed.

SYSTEM_RUNAWAY(name1)

indicates, in a 1-byte location (*name1*), whether the transaction definition specifies the system default runaway-task time limit, which is specified on the ICVR system initialization parameter.

- YES** The transaction definition specifies the system default runaway limit.
- NO** The transaction is not governed by the system runaway limit.

RUNAWAY_LIMIT(name4 | (Rn))

is the runaway-task time limit specified on the transaction definition. If SYSTEM_RUNAWAY is YES, the value returned is the value defined by the ICVR system initialization parameter.

name4 The name of a 4-byte location that is to receive the task runaway limit, expressed as a binary value.

(Rn) A register to receive the task runaway limit, expressed as a binary value.

DYNAMIC(name1)

indicates, in a 1-byte location (*name1*), whether the transaction is defined for dynamic transaction routing.

YES The transaction is to be dynamically routed to a remote CICS.

NO The transaction is not to be dynamically routed.

REMOTE_SYSTEM(name4)

is the name of the remote system as specified on the transaction definition.

If the DYNAMIC parameter returns YES, REMOTE_SYSTEM returns the default name, which can be changed by the dynamic routing program.

If the DYNAMIC parameter returns NO, this is the actual remote system to which the transaction is to be routed.

name4 The name of a 4-byte location to receive the defined name of the remote system.

REMOTE_NAME(name8)

is the name by which the transaction is known in a remote system.

name8 The name of an 8-byte location to receive the transaction's remote name.

TRAN_ROUTING_PROFILE(name8)

is the name of the profile that CICS is to use to route the transaction to a remote system.

name8 The name of an 8-byte location to receive the transaction-routing profile.

LOCAL_QUEUEING(name1)

indicates, in a 1-byte location (*name1*), whether a start request for this transaction is eligible to queue locally if the transaction is to be started on another system, and the remote system is not available.

YES The request can be queued locally.

NO The request is not to be queued locally.

TCLASS(name1)

indicates, in a 1-byte location (*name1*), whether the transaction belongs to a transaction class.

YES The transaction is a member of the transaction class named in the TCLASS_NAME parameter.

NO The transaction is not a member of a transaction class.

TCLASS_NAME(name8)

is the name of the transaction class to which the transaction belongs.

name8 The name of an 8-byte location to receive transaction class name to which the transaction belongs.

DTIMEOUT(name4)

is the deadlock time-out value for the transaction.

name4 The name of a 4-byte location that is to receive the deadlock time-out value, expressed as a binary value.

(Rn) A register to receive the deadlock time-out value, expressed as a binary value.

Note that a value of zero means that the transaction resource definition specifies DTIMOUT(NO).

INDOUBT(name1)

indicates, in a 1-byte location (*name1*), the action CICS is to take if the transaction abends while using intercommunication, and syncpointing status is in doubt.

BACKOUT Any changes made by the transaction to recoverable resources are to be backed out.

COMMIT Any changes made by the transaction to recoverable resources are to be committed.

WAIT Any changes made to recoverable temporary storage are locked until the session is recovered.

RESTART(name1)

indicates, in a 1-byte location (*name1*), whether the transaction restart is to be considered for transaction restart.

YES The transaction can be restarted.

NO The transaction cannot be restarted.

SPURGE(name1)

indicates, in a 1-byte location (*name1*), whether the transaction is defined as system-purgeable.

YES The transaction is system-purgeable.

NO The transaction is not system-purgeable.

TPURGE(name1)

indicates, in a 1-byte location (*name1*), whether the transaction is defined as purgeable in the event of a VTAM terminal error.

YES The transaction can be purged if a terminal error occurs.

NO The transaction can not be purged if a terminal error occurs.

DUMP(name1)

indicates, in a 1-byte location (*name1*), whether CICS is to take a transaction dump if the transaction abends.

YES A transaction dump is required.

NO A transaction dump is not required.

transaction management functions

TRACE(name1)

indicates, in a 1-byte location (*name1*), the level of tracing defined for the transaction

STANDARD

CICS standard-level trace This equates to TRACE(YES) in the TRANSACTION resource definition.

SPECIAL

CICS special-level trace This is the result of special trace being set by means of an EXEC CICS SET TRANSACTION command.

SUPPRESSED

Tracing is suppressed for the transaction This equates to TRACE(NO) in the TRANSACTION resource definition.

SHUTDOWN(name1)

indicates, in a 1-byte location (*name1*), whether the transaction can be run during CICS shutdown.

ENABLED

The transaction is enabled to run during CICS shutdown.

DISABLED

The transaction is disabled from running during CICS shutdown.

RESSEC(name1)

indicates, in a 1-byte location (*name1*), whether resource security checking is required for the transaction.

YES

Resource security checking is required.

NO

Resource security checking is not required.

CMDSEC(name1)

indicates, in a 1-byte location (*name1*), whether command security checking is required for the transaction.

YES

Command security checking is required.

NO

Command security checking is not required.

STORAGE_FREEZE(name1 | (Rn))

indicates, in a 1-byte location (*name1*), whether storage freeze is defined for the transaction by means of the STGFRZ option on the CICS-supplied field engineering transaction, CSFE.

YES

Storage that is normally freed during the running of a transaction is frozen.

NO

Storage is freed normally during the running of the transaction.

REMOTE(name1)

indicates, in a 1-byte location (*name1*), whether the transaction is defined as remote.

YES

The transaction is a remote transaction.

NO

The transaction is not a remote transaction.

ISOLATE(name1)

indicates, in a 1-byte location (*name1*), whether transaction isolation is required for the transaction's task-lifetime user-key storage.

YES

Transaction isolation is required for task-lifetime user-key storage.

NO

Transaction isolation is not required for task-lifetime user-key storage.

SYSTEM_ATTACH(name1)

indicates, in a 1-byte location (*name1*), whether the tasks attached with this tranid are always to be attached as system tasks.

YES A system task is being attached for this transaction.
NO A user task is being attached for this transaction.

RESPONSE and REASON values for INQUIRE_TRANDEF:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	UNKNOWN_TRANSACTION_ID
INVALID	None
DISASTER	LOGIC_ERROR
PURGED	None

The INQUIRE_TRANSACTION call

The INQUIRE_TRANSACTION function is provided on the DFHXMIQX macro call. Its purpose is to allow you to obtain information about a transaction that is attached (task). In general, this call is equivalent to the EXEC CICS INQUIRE TASK command, with some minor differences.

INQUIRE_TRANSACTION

```

DFHXMIQX [CALL,]
    [CLEAR,]
    [IN,]
    FUNCTION(INQUIRE_TRANSACTION),
    [TRANSACTION_TOKEN(name8),]
    [OUT,]
    [CICS_UOW_ID(name8),]
    [CMDSEC(name1),]
    [DTIMEOUT(name4 | (Rn)),]
    [DUMP(name1),]
    [DYNAMIC(name1),]
    [FACILITY_NAME(name4),]
    [FACILITY_TYPE(name1),]
    [INDOUBT(name1),]
    [INITIAL_PROGRAM(name8),]
    [ISOLATE(name1),]
    [LOCAL_QUEUEING(name1),]
    [NETNAME(name8),]
    [ORIGINAL_TRANSACTION_ID(name4),]
    [OUT_TRANSACTION_TOKEN(name8),]
    [PROFILE_NAME(name8),]
    [REMOTE(name1),]
    [REMOTE_NAME(name8),]
    [REMOTE_SYSTEM(name4),]
    [RESOURCE_NAME(name8),]
    [RESOURCE_TYPE(name8),]
    [RESSEC(name1),]
    [RESTART(name1),]
    [RESTART_COUNT(name2 | (Rn)),]
    [RUNAWAY_LIMIT(name4 | (Rn)),]
    [SPURGE(name1),]
    [START_CODE(name1),]
    [STATUS(name1),]
    [STORAGE_CLEAR(name1),]
    [SUSPEND_TIME(name4 | (Rn)),]
    [SYSTEM_TRANSACTION(name1),]
    [TASK_PRIORITY(name1),]
    [TASKDATAKEY(name1),]
    [TASKDATALOC(name1),]
    [TCLASS(name1), [TCLASS_NAME(name8),]]
    [TPURGE(name1),]
    [TRACE(name1),]
    [TRAN_PRIORITY(name1),]
    [TRAN_ROUTING_PROFILE(name8),]
    [TRANNUM(name4 | string | 'string'),]
    [TRANSACTION_ID(name4),]
    [Twasize(name4),]
    [USERID(name8),]
    RESPONSE (name1 | *),
    REASON (name1 | *)]

```

The descriptions of the following parameters are the same as the corresponding parameters on the INQUIRE_TRANDEF function call.

CMDSEC	RESSEC
DTIMEOUT	RESTART
DUMP	RUNAWAY_LIMIT
DYNAMIC	SPURGE
INDOUBT	STATUS
INITIAL_PROGRAM	STORAGE_CLEAR
ISOLATE	TASKDATAKEY
LOCAL_QUEUEING	TASKDATALOC
PROFILE_NAME	TCLASS
REMOTE	TRAN_ROUTING_PROFILE
REMOTE_NAME	TRANSACTION_ID
REMOTE_SYSTEM	TWASIZE

The parameter descriptions that follow explain briefly the possible values that can be returned on an INQUIRE_TRANSACTION call. For a more detailed explanation of these parameters, see the corresponding parameter descriptions for the TRANSACTION resource definition in the *CICS/ESA Resource Definition Guide*.

TRANSACTION_TOKEN(name8)

specifies the transaction token for the task being inquired upon. This parameter is optional, and if omitted, the current task is assumed.

If you issue this call within an XXMATT global user exit program, the current task may be a CICS system task. To inquire on the user task for which XXMATT is invoked, you must specify the transaction token passed on the XXMATT exit-specific parameter list.

name8 The name of an 8-byte location that contains the transaction token.

CICS_UOW_ID(name8)

is the CICS unit of work identifier for the task.

name8 The name of an 8-byte location to receive the unit of work id.

FACILITY_NAME(name4)

is the name of the principal facility associated with the task.

name4 The name of a 4-byte location to receive the name of the principal facility.

FACILITY_TYPE(name1)

indicates, in a 1-byte location (*name1*), the type of principal facility associated with the task.

NONE There is no principal facility.

TERMINAL The principal facility is a terminal.

TD The principal facility is a transient data queue.

START The principal facility is an interval control element.

NETNAME(name8)

is the network name of the principal facility associated with this task.

name8 The name of an 8-byte location to receive the network name.

ORIGINAL_TRANSACTION_ID(name4)

is the transaction id that was used to attach the transaction. For example, if an alias was used at a terminal, this field returns the alias.

name4 The name of a 4-byte location to receive the name of the original transaction identifier.

OUT_TRANSACTION_TOKEN(name8)

is the token that represents the task.

name8 The name of an 8-byte location to receive the transaction token for the task.

RESOURCE_NAME(name8)

is the name of a resource that the (suspended) transaction waiting for.

name8 The name of an 8-byte location to receive the name of the resource on which the transaction is waiting.

RESOURCE_TYPE(name8)

is the type of resource that the (suspended) transaction waiting for.

name8 The name of an 8-byte location to receive the type of resource on which the transaction is waiting.

RESTART_COUNT(name2 | (Rn))

is the number of times this instance of the transaction has been restarted.

name2 The name of a 2-byte location to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

(Rn) A register to receive the number of times the transaction has been restarted, expressed as a half-word binary value.

START_CODE(name1)

indicates, in a 1-byte location (*name1*), how the task was started:

C	A CICS internal attach.
T	A terminal input attach.
TT	A permanent transaction terminal attach.
QD	A transient data trigger level attach.
S	A START command without any data.
SD	A START command with data.
SZ	A front end programming interface (FEPI) attach.
DF	The start code isn't yet known—to be set later.

SUSPEND_TIME(name4 | (Rn))

is the length of time that the task has been in its current suspended state.

name4 The name of a 4-byte location to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

(Rn) A register to receive the number of seconds, rounded down, the task has been suspended, expressed as a binary value.

SYSTEM_TRANSACTION(name1)

indicates, in a 1-byte location (*name1*), whether the task is CICS system task.

YES The task is a CICS system task.
NO The task is not a CICS system task.

TASK_PRIORITY(name1)

is the combined task priority, which is the sum of the priorities defined for the terminal, transaction, and operator.

name1 The name of a 1-byte location to receive the task priority, expressed as a binary number.

TRANNUM(name4)

is the task number of the transaction.

name4 The name of a 4-byte location to receive the task number.

USERID(name8)

is the userid associated with this task.

name8 The name of an 8-byte location to receive the userid.

RESPONSE and REASON values for INQUIRE_TRANSACTION:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
DISASTER	ABEND LOOP
INVALID EXCEPTION	None NO_TRANSACTION_ ENVIRONMENT BUFFER_TOO_SMALL INVALID_TRANSACTION_TOKEN
KERNERROR	None

The SET_TRANSACTION call

The SET_TRANSACTION function is provided on the DFHXMIX macro call. Its purpose is to allow you to change the task priority and transaction class of the current task.

Note that you can use this call to change the TCLASS_NAME only when it is invoked from an XXMATT global user exit program.

SET_TRANSACTION

```
DFHXMIX [CALL,]
        [CLEAR,]
        [IN,
        FUNCTION(SET_TRANSACTION),
        [TRANSACTION_TOKEN(name8),]
        [TASK_PRIORITY(name4),]
        [TCLASS_NAME(name8),]]
        [OUT,
        RESPONSE (name1 | *),
        REASON (name1 | *)]
```

user journaling function

TRANSACTION_TOKEN(name8)

specifies the transaction token that represents the task being modified. If you omit this parameter, the call defaults to the current task.

name8 The name of an 8-byte location that contains the transaction token.

TASK_PRIORITY(name4)

specifies the new task priority being set for the task identified by TRANSACTION_TOKEN.

name4 The name of a 4-byte location that contains the new task priority number, expressed as a binary value.

TCLASS_NAME(name8)

specifies the new transaction class name that you want to associate this task with. To specify that the task is not to be in any transaction class, specify the special default system name DFHTCL00.

name8 The name of an 8-byte location that contains the name of the new transaction class. Set this field to DFHTCL00 for no transaction class.

RESPONSE and REASON values for SET_TRANSACTION:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	NO_TRANSACTION_ ENVIRONMENT UNKNOWN_TCLASS INVALID_TRANSACTION_TOKEN
DISASTER	ABEND LOOP
INVALID	None
KERNERROR	None

User journaling function

There is one XPI user journaling function, which is the DFHJCJCX call WRITE_JOURNAL_DATA.

DFHJCJCX calls cannot be used in any exit program invoked from any global user exit point in the:

- Statistics domain
- Monitor domain
- Dump domain
- Dispatcher domain
- Transient data program
- Journal control program (DFHJCP).

The WRITE_JOURNAL_DATA call

WRITE_JOURNAL_DATA writes a single journal record to a named CICS journal data set.

WRITE_JOURNAL DATA

```
DFHJCJCX [CALL,]
          [CLEAR,]
          [IN,
          FUNCTION(WRITE_JOURNAL_DATA),
          FROM(block-descriptor),
          JOURNAL_ID(name2 | (Rn) | expression),
          JOURNAL_RECORD_ID(name2 | string | 'string'),
          WAIT(YES|NO),
          [RECORD_PREFIX(block-descriptor),]]
          [OUT,
          RESPONSE(name1 | *),
          REASON(name1 | *)]
```

Important

There is a restriction in using the XPI early during initialization. Do not start exit programs that use the XPI functions TRANSACTION_DUMP, WRITE_JOURNAL_DATA, MONITOR, and INQUIRE_MONITOR_DATA until the second phase of the PLTPI. For further information about the PLTPI, refer to Chapter 4, "Writing initialization and shutdown programs" on page 335.

FROM(block-descriptor)

is the address and the length of the journal record.

#

The block-descriptor comprises 8 bytes of data. The first 4 bytes hold the address of the data to be written. The second 4 bytes hold the length of the data. The block-descriptor is moved by the DFHJCJCX macro call to the location JCJC_FROM, which is mapped by the DFHJCJCY DSECT.

JOURNAL_ID(name2 | (Rn) | expression)

is the number of the journal data set to which the journal record is to be written.

name2 The name of a 2-byte location where the ID is held
(Rn) A register with the low-order 2 bytes holding the ID
expression Any arithmetic expression, valid in assembler language, that yields a number that can be expressed in 2 bytes.

JOURNAL_RECORD_ID(name2 | string | 'string')

is a 2-character value to be written to the journal record to identify its origin.

name2 The name of a 2-byte location
string A character string that is limited to a length of 2 in the generated code
'string' A character string enclosed in quotation marks.

WAIT(YES|NO)

specifies whether CICS is to wait until the record is written to auxiliary storage before returning control to the exit program.

user journaling function

RECORD_PREFIX(block-descriptor)

is the optional user prefix.

RESPONSE and REASON values for WRITE_JOURNAL_DATA:

<i>RESPONSE</i>	<i>REASON</i>
OK	None
EXCEPTION	IO_ERROR
	LENGTH_ERROR
	JOURNAL_NOT_FOUND
	JOURNAL_NOT_OPEN
	STATUS_ERROR
DISASTER	None
INVALID	None
KERNERROR	None
PURGED	None

Note: For more detail, refer to the explanation of RESPONSE and REASON in “General form of an XPI call” on page 234.

Part 2. Customizing with initialization and shutdown programs

Chapter 4. Writing initialization and shutdown programs	335	Shutdown programs	337	
Initialization programs	335	General considerations	338

initialization and shutdown programs

Chapter 4. Writing initialization and shutdown programs

You can write programs to run during the initialization and shutdown phases of CICS processing. Any program that is to run at these times must be defined to CICS in a program list table (PLT). Information about how to code the PLT is provided in the *CICS/ESA Resource Definition Guide*.

Initialization programs

There are three stages in CICS initialization processing and two phases of program list table (PLT) execution. You can write programs to execute during the second and the third stages of initialization, but not during the first. Any program that is to execute during CICS initialization must be specified in a program list table (PLT), and the suffix of that PLT must be named on the program list table post initialization (PLTPI) system initialization parameter.

PLT programs for the second stage of initialization

During the *second* stage of CICS initialization processing, the only PLT programs that can execute are those containing the enabling commands for global and task-related user exit programs. These programs are specified in the *first* part of the PLTPI list. This allows you to enable those exit programs that are needed during recovery.

The following points apply to all first phase PLTPI programs:

- The programs must be written in assembler language.
- They must run AMODE 31.
- The only EXEC CICS commands they should contain are:
 - ENABLE
 - EXTRACT EXIT

#

Because this stage occurs before recovery when initialization is incomplete, no other CICS services can be invoked.

- If a first phase PLTPI program enables an exit program that issues any of the XPI calls INQUIRE_MONITORING_DATA, MONITOR, TRANSACTION_DUMP, or WRITE_JOURNAL_DATA, it must not specify the START option on the EXEC CICS ENABLE COMMAND.
- First phase PLTPI programs must not enable any task-related user exit program with the TASKSTART option.
- You do not have to define first phase PLTI programs to CICS. If you do not, default definitions are installed automatically by CICS. Note that this happens whether or not program autoinstall is specified as active on the PGAIPGM system initialization parameter. The autoinstall user program is **not** invoked to allow the definitions to be modified. The programs are defined with the following attributes:

|
|
|
|
|
|
|
|
|
|
|

```
LANGUAGE(Assembler)
RELOAD(No)
STATUS(Enabled)
CEDF(No)
```

initialization programs

DATALOCATION(Below)
EXECKEY(CICS)
EXECUTIONSET(Fullapi)

If any of the default attributes are unsuitable, you must define the programs statically (by defining entries in the CSD and installing the definitions).

PLT programs for the third stage of initialization

During the *third* stage of CICS initialization, most CICS services are available to PLT programs. These programs are specified in the *second* part of the PLTPI list. The limitations on the services that are available to second phase PLTPI programs are described below.

- Because interregion communication (IRC) and intersystem communication (ISC) have pseudo-terminal entries associated with their function, you cannot run any IRC or ISC functions during PLTPI processing. This includes performing inquiries on those ISC/IRC functions.
- PLTPI programs may request services that could suspend the issuing task. (But note that this affects the time at which control is given to CICS.) The SUSPEND must not require the decision to resume to be taken by another task.
- Although PLTPI programs can issue interval control START commands, the requested transactions are not attached before the initialization stages have completed. Because this cannot happen until after the PLTPI programs themselves have been completed, the latter must not be dependent on anything that the requested transactions might do.
- PLTPI programs must not issue dump requests.
- PLTPI programs must not use the EXEC CICS PERFORM SHUTDOWN command, or a severe error will occur in DFHDMDM. The EXEC CICS PERFORM SHUTDOWN IMMEDIATE command is allowed.
- You must define second phase PLTPI programs to CICS. You can either define the programs statically, or use program autoinstall (program autoinstall is described in the *CICS/ESA Resource Definition Guide*).

Rebuilding protected resources from second phase programs

When a protected resource is accessed, CICS normally enqueues on the resource to ensure that the issuing task has exclusive control. The automatic dequeuing of a protected resource is deferred until the task terminates or until the task issues an EXEC CICS SYNCPOINT command.

When a second phase PLTPI program is involved in rebuilding a protected resource during recovery, an enqueue occurs on each logical record. Because the dequeues are deferred, you should declare syncpoints throughout the recovery process to allow dequeues to occur. If you do not do this, dynamic storage fills up with the CICS control blocks that control the enqueue and dequeue processes. However, note that you must not issue a SYNCPOINT ROLLBACK command in a PLTPI program. Also, you should ensure that no other task is attached that could access the resource until the PLTPI program has completed its recovery work, because the second task would be able to gain control of the resource whenever the PLTPI program declared a syncpoint.

Shutdown programs

A controlled CICS shutdown has three quiesce stages. You can write programs to execute during the first two quiesce stages, but not during the third. Any program that is to execute during CICS shutdown must be defined in a program list table (PLT), and the PLT must be named on the program list table shutdown (PLTSD) system initialization parameter. You can override the PLTSD value by providing a PLT name on the CEMT PERFORM SHUTDOWN command, or on the EXEC CICS PERFORM SHUTDOWN command. If a PLTSD program abends, syncpoint rollback occurs.

PLT programs for first quiesce stage

Programs that are to execute during the first quiesce stage of CICS shutdown are specified in the first half of the PLT.

You must define first stage PLTSD programs to CICS. You can either define the programs statically, or use program autoinstall.

Although terminals are still available during the first quiesce stage, tasks that are started by terminal input are rejected unless they are named in a shutdown transaction list table (XLT), or are CICS-supplied transactions, such as CEMT, CSAC, CSTE, and CSNE, that are defined as SHUTDOWN(ENABLED) in the supplied definitions.

The first quiesce stage is complete when all of the first-stage PLT programs have executed, and when there are no user tasks in the system.

PLT programs for the second quiesce stage

Programs that are to execute during the second quiesce stage of CICS shutdown are specified in the second half of the PLT.

You do not have to define second stage PLTSD programs to CICS. If you do not, default definitions are installed automatically by CICS, as described for first phase PLTPI programs. If any of the default attributes are unsuitable, you must define the programs statically.

During the second quiesce stage, no new tasks can start, and no terminals are available. Because of this, second phase PLT programs must not cause other tasks to be started, and they cannot communicate with terminals. Further, second phase PLT programs must not cause any resource security checking to be performed.

If a transaction abend occurs while the PLTSD program is running, CICS is left in a permanent wait state. To avoid this happening, ensure that your PLTSD program handles **all** abend conditions.

The second quiesce stage is complete when all of the second phase PLT programs have been executed.

The sample shutdown program, DFH\$SDAP

CICS provides a sample shutdown program, DFH\$SDAP, for use during the first quiesce stage of shutdown.

DFH\$SDAP's function is to prevent CICS from becoming suspended for long periods when doing a normal shutdown, because of long running tasks which have not completed. The program is in four phases. Each phase issues one of the following commands, with a delay between each one:

- FORCE
- FORCEPURGE
- VTAM FORCECLOSE
- SHUTDOWN IMMEDIATE.

Tasks which are purged during the first three of these instructions should be backed out. If the system is successfully terminated before the SHUTDOWN IMMEDIATE is issued, a warm keypoint is taken.

You may want to change the delay intervals to suit the level of activity in your system. For example, the VTAM FORCECLOSE command should be given sufficient time to work. If it is not, the warm keypoint is not taken.

The source code of the sample program is provided in assembler only. You must translate, assemble, and link-edit it. You must then use RDO to define and install DFH\$SDAP and its associated transaction SDAP. You must define SDAP with the SHUTDOWN attribute specified as ENABLED. Finally, to cause DFH\$SDAP to be invoked, put an entry in the first part of the PLT that is named on the PLTSD system initialization parameter.

General considerations

The comments in the remainder of the chapter apply to both initialization and shutdown programs.

- It is recommended that you terminate all PLT programs with an EXEC CICS RETURN command.
- PLT programs receive control in primary-space translation mode. (For information about translation modes, see the *IBM ESA/370 Principles of Operation* manual.) They must return control to CICS in the same mode, and must restore any general purpose registers or access registers that they use.

+ Storage keys for PLT programs

+ You need to consider the following (whether or not you are running CICS with the storage protection facility):

- The execution key in which your PLT programs are invoked
- The storage key of data storage obtained for your PLT programs.

Execution key for PLT programs

CICS always gives control to PLT programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it passes control to any PLT programs invoked during initialization or shutdown. However, if a PLT-defined *shutdown* program itself passes control to another program (via a link or transfer-control command), the program thus invoked executes according to the execution key (EXECKEY) defined in its program resource definition.

Important

You are strongly recommended to specify EXECKEY(CICS) when defining both PLT programs and programs to which a PLT program passes control.

Data storage key for PLT programs

The storage key of storage used by PLT programs depends on how the storage is obtained:

- Any working storage requested by the PLT program is in the key set by the TASKDATAKEY of the transaction under which the PLT program is invoked. In the case of those PLT programs that run during initialization (PLTPI programs), the transaction is always an internal CICS transaction, in which case the TASKDATAKEY is always CICS. In the case of those programs that run during shutdown (PLTSD programs), it depends on the transaction you use to issue the shutdown command. If you issue a CEMT PERFORM SHUTDOWN command, the TASKDATAKEY is always CICS. If you run a user-defined transaction, to invoke a program that issues an EXEC CICS PERFORM SHUTDOWN command, the TASKDATAKEY can be either USER or CICS.
- PLT programs can use EXEC CICS commands to obtain storage by issuing:
 - Explicit EXEC CICS GETMAIN commands
 - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the PLT program is invoked, exactly as described for working storage.

As an example, consider a transaction defined with TASKDATAKEY(USER) that causes a PLT shutdown program to be invoked. In this case, any implicit or explicit storage acquired by the PLT program by means of an EXEC CICS command is, by default, in user-key storage. However, on an EXEC CICS GETMAIN command, the PLT program can override the TASKDATAKEY option by specifying either CICS DATAKEY or USER DATAKEY.

Part 3. Customizing with user-replaceable programs

Chapter 5. General notes about user-replaceable programs	343	The autoinstall control program at INSTALL	453
		The autoinstall control program at DELETE	456
		The sample autoinstall control program for APPC connections	456
Chapter 6. Writing a program error program	351		
Chapter 7. Writing a transaction restart program	357		
Chapter 8. Writing a terminal error program	361		
Background to error handling for TCAM and sequential devices	361		
The sample terminal error program	363		
User-written terminal error programs	384		
Chapter 9. Writing a node error program	395		
Background to CICS-VTAM error handling	396		
When an abnormal condition occurs	403		
The sample node error program	411		
User-written node error programs	421		
Using the node error program with XRF or persistent sessions	427		
Using the node error program with VTAM generic resources	429		
Chapter 10. Writing a program to control autoinstall of terminals	431		
Preliminary considerations	431		
The autoinstall control program at INSTALL	433		
The autoinstall control program at DELETE	441		
Naming, testing, and debugging your autoinstall control program	442		
The sample programs and copy books	444		
Chapter 11. Writing a program to control autoinstall of APPC connections	451		
Preliminary considerations	451		
		The autoinstall control program at INSTALL	453
		The autoinstall control program at DELETE	456
		The sample autoinstall control program for APPC connections	456
		+ Chapter 12. Writing a program to control + autoinstall of shipped terminals	459
		+ Installing shipped terminals and connections	459
		+ The autoinstall control program at INSTALL	461
		+ The autoinstall control program at DELETE	464
		+ Default actions of the sample programs	465
		Chapter 13. Writing a program to control autoinstall of programs	467
		Preliminary considerations	467
		Benefits of autoinstall	469
		Requirements for autoinstall	469
		The autoinstall control program at INSTALL	470
		The sample autoinstall control program for programs, DFHPGADX	472
		Chapter 14. Writing a dynamic transaction routing program	477
		The dynamic transaction routing program, DFHDYP	477
		Parameters passed to the dynamic transaction routing program	484
		Naming your dynamic routing program	491
		Testing your dynamic transaction routing program	491
		Dynamic transaction routing sample programs	492
		Chapter 15. The user-replaceable journaling programs	493
		Chapter 16. Writing a CICS-DBCTL interface status program	497

user-replaceable programs

Chapter 5. General notes about user-replaceable programs

The comments in this chapter apply to all the user-replaceable programs described in Part 3 of this book.

A user-replaceable program is a CICS-supplied program that is always invoked at a particular point in CICS processing, as if it were part of the CICS code. You can modify the supplied program by including your own logic, or replace it with a version that you write yourself.

Rewriting user-replaceable programs

There are some general considerations that you must bear in mind when creating your own versions of user-replaceable programs:

- User-replaceable programs are all command-level programs (not user exits).
- With the exception of the journaling programs, DFHXJCO and DFHXJCC, which must be written in assembler language, you can code user-replaceable programs in any of the languages supported by CICS (that is, in assembler language, COBOL, PL/I, or C). An assembler-language version of each program is provided, in source form, in the CICS410.SDFHSAMP library. In addition, COBOL, PL/I, or C versions are provided for some programs. The relevant chapter lists the sample programs, copy books, and macros supplied in each case.
- You can trap an abend in a user-replaceable program by making the program issue an EXEC CICS HANDLE ABEND command. However, if no HANDLE ABEND is issued, CICS does not abend the task but returns control to the CICS module that called the program. The action taken by the CICS module depends on the user-replaceable program concerned.
- Upon return from any user-replaceable program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

- User-replaceable programs, and any programs invoked by user-replaceable programs, can be RMODE ANY but **must** be AMODE 31.
- You must ensure that user-replaceable programs are defined as local. User-replaceable programs cannot be run in a remote region. This applies to all user-replaceable programs, including the autoinstall control program and the dynamic transaction routing program.
- User-replaceable programs produce only system dumps when a program check occurs; they do not produce transaction dumps.
- You can use the CICS Execution Diagnostic Facility (EDF) to test user-replaceable programs. However, EDF does not work if the initial transaction is a CICS-supplied transaction.

Assembling and link-editing user-replaceable programs

The source for the CICS-supplied user-replaceable programs is installed in the CICS410.SDFHSAMP library. If you intend changing any of these programs, take a copy of the CICS410.SDFHSAMP library and update the copy only. If the original SDFHSAMP is serviced, and a user-replaceable program is modified, you may like to reflect the changes in your own version of the code.

To replace one of these CICS-supplied programs, assemble and link-edit your version of the program. All programs are supplied as command-level programs, and must be translated before assembly and link-edit. Note that the translator options NOPROLOG and NOEPILOG should be coded with your versions of DFHZNEP and DFHTEP.

If you have user-written versions of DFHZNEP, DFHPEP, DFHTEP, DFHXJCO, or DFHXJCC from an earlier release of CICS, and they use macros, recode the programs to use EXEC CICS commands.

To translate, assemble, and link-edit user-replaceable programs, you can use the CICS-supplied procedure DFHEITAL. For information about using DFHEITAL, see the *CICS/ESA System Definition Guide*. If you use SMP/E, you can give the object-deck output after translation and assembly to SMP/E for link-editing.

When link-editing a user-replaceable program, you must link-edit it with the EXEC interface module (stub). This stub enables the program to communicate with the EXEC interface program (DFHEIP). If you use the DFHEITAL procedure, it link-edits programs with the EXEC interface stub by default. For more information about the EXEC interface stub, see the *CICS/ESA System Definition Guide*.

The job stream in Figure 12 on page 345 is an example of the assembly and link-edit of a user-replaceable program. The figure is followed by some explanatory notes.


```

//ASSMBLE EXEC DFHASMVS,MOD='program_name',           1 2
//      INDEX='CICS410',ASMBLR=IEV90,
//      LIST='LIST,XREF(SHORT),RENT,ALIGN'
//ASSEM.SYSPUNCH DD DSN=yourtext_dataset(program_name),DISP=OLD 3
//ASSEM.SYSIN DD *
  TITLE 'CICS/ESA : V3.2.1 : ASSEMBLE AND LINK-EDIT OF program_name'
    COPY DFHGDEFS GLOBAL SYMBOL DEFINITIONS           4
&MVS      SETB 1          SET WHEN MVS
&MVS811   SETB 1          SET WHEN MVS/ESA
&VSDSECT  SETA 1          PRINT NO DSECTS
          DFHCOVER OS
//          DD DSN=your_dataset(program_name),DISP=SHR 5
/*
//LNKEDIT EXEC DFHLNKVS,
//      PARM='LIST,XREF,LET,RENT,REFR',
//      NAME=SDFHLOAD,
// INDEX='CICS410',
// INDEX2='CICS410'
//SDFHLOAD DD DSN=CICS410.SDFHLOAD,DISP=SHR
//USERTEXT DD DSN=yourtext_dataset,DISP=SHR
//SYSLIN DD *
/* link-edit statements, see Figure 13 on page 346      */ 6
/*

```

Figure 12. Job stream to assemble and link-edit a user-replaceable program

Notes:

1 This job stream uses the CICS-supplied procedure DFHASMVS to assemble and link-edit user-replaceable programs. The DFHASMVS procedure refers to the MVS library SYS1.MODGEN. If you have not yet restructured MVS/ESA (moving members from SYS1.AMODGEN to SYS1.MODGEN), change the SYS1.MODGEN reference to SYS1.AMODGEN in the DFHASMVS procedure, until you have restructured MVS/ESA. When you have restructured MVS/ESA, you must return the SYS1.AMODGEN reference to SYS1.MODGEN.

2 program_name is the name of the program (on your_dataset) being modified.

3 yourtext_dataset is the name of the data set containing the text after assembly.

4 The assembler statements perform the following functions:

- Define the global symbols for the assembly (by a copy statement for the DFHGDEFS module)
- Set the following global symbols to '1':

Statement	Description
&MVS SETB 1	CICS is to run under MVS.
&MVS811 SETB 1	CICS is to run under MVS/ESA.
&VSDSECT SETA 1	Stop printing of CICS dummy sections (DSECTS).

5 your_dataset is the name of the data set containing your version of the code.

6 The input to the linkage editor must include several statements specific to the user-replaceable module. The appropriate statements are given in Figure 13.

notes about user-replaceable programs

Link-edit statements for DFHPEP.

```
ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHPEP      this CSECT is in USERTXT(DFHPEP)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHPEP)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHPEP
NAME DFHPEP(R)
```

Link-edit statements for DFHREST.

```
ORDER DFHEAI
ORDER DFHREST
ORDER DFHEAI0
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE SDFHLOAD(DFHREST)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
```

Link-edit statements for DFHTEP.

```
ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHTEP      this CSECT is in USERTXT(DFHXTEP)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHXTEP)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHTEPNA
NAME DFHTEP(R)
```

Link-edit statements for DFHZNEP.

```
ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHZNEP0    this CSECT is in USERTXT(DFHZNEP0)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHZNEP0)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHZNENA
NAME DFHZNEP(R)
```

Figure 13 (Part 1 of 3). Link-edit statements for user-replaceable programs

Link-edit statements for DFHZATDX.

```

ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHZATDX    this CSECT is in USERTXT(DFHZATDX)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHZATDX)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
NAME DFHZATDX(R)

```

Link-edit statements for DFHDYP.

```

ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHDYP      this CSECT is in USERTXT(DFHDYP)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHDYP)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHDYP
NAME DFHDYP(R)

```

Link-edit statements for DFHXJCO.

```

ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHXJCO     this CSECT is in USERTXT(DFHXJCO)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHXJCO)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHXJCO
NAME DFHXJCO(R)

```

Link-edit statements for DFHXJCC.

```

ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHXJCC     this CSECT is in USERTXT(DFHXJCC)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHXJCC)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHXJCC
NAME DFHXJCC(R)

```

Figure 13 (Part 2 of 3). Link-edit statements for user-replaceable programs

notes about user-replaceable programs

Link-edit statements for DFHDBUEX.

```
ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHDBUEX    this CSECT is in USERTXT(DFHDBUEX)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHDBUEX)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHDBUEX
NAME DFHDBUEX(R)
```

Link-edit statements for DFHXCURM.

```
ORDER DFHEAI      this CSECT is in SDFHLOAD(DFHEAI)
ORDER DFHXCURM    this CSECT is in USERTXT(DFHXCURM)
ORDER DFHEAI0     this CSECT is in SDFHLOAD(DFHEAI0)
INCLUDE SDFHLOAD(DFHEAI)
INCLUDE USERTXT(DFHXCURM)
INCLUDE SDFHLOAD(DFHEAI0)
MODE AMODE(31),RMODE(ANY)
ENTRY DFHXCURM
NAME DFHXCURM(R)
```

Figure 13 (Part 3 of 3). Link-edit statements for user-replaceable programs

User-replaceable programs and the CICS storage protection facility

When you are running CICS with the storage protection facility, there are two points you need to consider:

- The execution key in which your user-replaceable programs run
- The storage key of data storage obtained for your user-replaceable programs.

Execution key for user-replaceable programs

When you are running with storage protection active, CICS always gives control to user-replaceable programs in CICS key. Even if you specify EXECKEY(USER) on the program resource definition, CICS forces CICS key when it invokes the program. However, if a user-replaceable program itself passes control to another program, the program thus invoked executes according to the execution key (EXECKEY) defined in its program resource definition.

Important

You are strongly recommended to specify EXECKEY(CICS) when defining both user-replaceable programs and programs to which a user-replaceable program passes control.

Data storage key for user-replaceable programs

The storage key of storage used by user-replaceable programs depends on how the storage is obtained:

- The communication area passed to the user-replaceable program by its caller is always in CICS key.
- Any working storage obtained for the user-replaceable program is in the key set by the TASKDATAKEY of the transaction under which the program is invoked.
- User-replaceable programs can use EXEC CICS commands to obtain storage, by issuing:
 - Explicit EXEC CICS GETMAIN commands
 - Implicit storage requests as a result of EXEC CICS commands that use the SET option.

The default storage key for storage obtained by EXEC CICS commands is set by the TASKDATAKEY of the transaction under which the user program is invoked.

As an example, consider a transaction defined with TASKDATAKEY(USER) that causes a user-replaceable program to be invoked. In this case, any implicit or explicit storage acquired by the user program by means of an EXEC CICS command is, by default, in user-key storage. However, on an EXEC CICS GETMAIN command, the user program can override the TASKDATAKEY option by specifying either CICSDATAKEY or USERDATAKEY.

notes about user-replaceable programs

Chapter 6. Writing a program error program

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

The CICS-supplied default program error program (DFHPEP) contains code to:

- Obtain program addressability
- Access the communication area
- Return control to DFHACP through an EXEC CICS RETURN command.

The source of DFHPEP is provided in assembler-language and C versions; you can modify one of these to include your own logic, or you can write your own program error program in any of the languages supported by CICS. There is a discussion of the reasons for using your own program error program in the *CICS/ESA Recovery and Restart Guide*. Note, however, that when writing a program error program you are subject to specific restrictions:

- Your program must be named DFHPEP.
- It must not issue any EXEC CICS commands that make use of MRO or ISC facilities (such as distributed transaction processing or function shipping).
- It must not issue any commands that access recoverable resources.
- It cannot influence the taking of a transaction dump.

The default DFHPEP module is a dummy module. If you want to customize it, you have to code the source yourself. To help you, a listing of DFHPEP is provided in Figure 14 on page 353. When you have written your program error program, translate and assemble it, and use it to replace the supplied dummy program. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to “Assembling and link-editing user-replaceable programs” on page 344.

Information available to DFHPEP in the communication area includes:

- The current abend code, at PEP_COM_CURRENT_ABEND_CODE.
- The original abend code, at PEP_COM_ORIGINAL_ABEND_CODE. The “original” and “current” abend codes are different if the transaction has suffered more than one abend—for example, if the failing program abended while handling a previous abend. In this case, the “original” abend is the first abend that the transaction suffered.
- The EIB at the time of the last EXEC CICS command, at PEP_COM_USERS_EIB.
- The name of the program that suffered the (current) abend, at PEP_COM_ABPROGRAM. PEP_COM_ABPROGRAM identifies the program as follows:
 - If the abend occurred in a distributed program link (DPL) server program running in a remote system, it identifies the server program.

the program error program

- If the abend is a local 'ASRA', 'ASRB', or 'ASRD', it identifies the program in which the program check or operating system abend occurred.
- In all other cases, it identifies the current PPT entry.
- The program status word (PSW) at the time of the (current) abend, at PEP_COM_PSW. The full contents of PEP_COM_PSW are significant for abend codes 'ASRA', 'ASRB', and 'ASRD' only; the last four characters (the PSW address) apply also to code 'AICA'.
- The GP registers (0–15) at the time of the (current) abend, at PEP_COM_REGISTERS.
- The execution key of the program at the time it suffered the (current) abend, at PEP_COM_KEY. The value of PEP_COM_KEY is significant for abend codes 'ASRA' and 'ASRB' only.
- Whether the (current) abend occurred as the result of a storage protection exception, at PEP_COM_STORAGE_HIT. The value of PEP_COM_STORAGE_HIT is significant for abend code 'ASRA' only, and indicates which of the protected dynamic storage areas (the CDSA, ECDSA, or ERDSA), if any, the failing program attempted to overwrite.
- The address of the task abend control block (TACB). This provides details of the subspace and access registers current at the time of the abend. The subspace value is in PEP_COM_SPACE.

•

APAR PN71385

Documentation for PN71385 added on 22 June 1995.

Program status word interrupt information, at PEP_COM_INT.

Note that information about the PSW, registers, execution key, and type of storage "hit" is meaningful only if the abend occurred in the local system; these fields are set to zeros if the abend occurred in a DPL server program running in a remote system.

In order to disable the transaction, you should assign the value 'PEP_COM_RETURN_DISABLE' to the field PEP_COM_RETURN_CODE. Otherwise, you should allow the field to default to zero, or set it to the value 'PEP_COM_RETURN_OK'. DFHACP does not allow CICS-supplied transactions to be disabled; you should not, therefore, attempt to disable transactions whose IDs begin with "C".

The assembler-language source code of the default program error program, and of the communication area, are shown in Figures 14 and 15, respectively.


```

DFHEISTG DSECT ,
*
*       Insert your own storage definitions here
*
*       DFHPCOM TYPE=DSECT
*****
* * * * *           P R O G R A M   E R R O R           * * * * *
* * * * *           P R O G R A M           * * * * *
*****
DFHPEP  CSECT                                PROGRAM ERROR PROGRAM CSECT
DFHPEP  RMODE ANY
        DFHREGS ,                            EQUATE REGISTERS
        XR    R1,R1
        ICM  R1,B'0011',EIBCALEN Get Commarea length
        BZ   RETURNX                          ...no Commarea; exit
        EXEC CICS ADDRESS COMMAREA(R2) ,
        USING DFHPEP_COMMAREA,R2
*
*       Insert your own code here
*
        LA   R1,PEP_COM_RETURN_OK
        B    RETURN
        DFHEJECT
*
RETURNER DS    0H                                Return for error cases
        LA   R1,PEP_COM_RETURN_DISABLE
RETURN   DS    0H
        ST   R1,PEP_COM_RETURN_CODE
RETURNX  DS    0H
        EXEC CICS RETURN ,
        END  DFHPEP

```

Figure 14. Source code of the default program error program (DFHPEP)

```

DFHPEP_COMMAREA DSECT
*
*       Standard header section
*
PEP_COM_STANDARD          DS    0F
PEP_COM_FUNCTION          DS    CL1      Always '1'
PEP_COM_COMPONENT        DS    CL2      Always 'PC'
PEP_COM_RESERVED         DS    C        Reserved
*

```

Figure 15 (Part 1 of 2). Source of DFHPEP communication area (assembler-language)

the program error program

```

*                               Abend codes and EIB
*
PEP_COM_CURRENT_ABEND_CODE     DS      CL4      Current abend code
PEP_COM_ORIGINAL_ABEND_CODE    DS      CL4      Original abend code
PEP_COM_USERS_EIB              DS      CL(EIBRLDBK-EIBTIME+L'EIBRLDBK)
*                               EIB at last EXEC CICS command
*
* Debugging information (program, PSW, registers and execution key at
* time of abend, hit storage indicator). If the abend occurred in a
* DPL server program running remotely, only program is meaningful.
*
PEP_COM_DEBUG                  DS      0F
PEP_COM_ABPROGRAM              DS      CL8      Program causing abend
PEP_COM_PSW                    DS      CL8      PSW at abend
*                               (codes ASRA, ASRB, AICA, ASRD)
PEP_COM_REGISTERS              DS      CL64     GP registers at abend
*                               (registers 0-15)
PEP_COM_KEY                    DS      X       Execution key at abend
*                               (ASRA and ASRB only)
PEP_COM_USER_KEY               EQU     9       User key
PEP_COM_CICS_KEY               EQU     8       CICS key
*
PEP_COM_STORAGE_HIT           DS      X       Storage type hit by 0C4
*                               (ASRA only)
PEP_COM_NO_HIT                 EQU     0       No hit, or not 0C4
PEP_COM_CDSA_HIT               EQU     1       CDSA hit
PEP_COM_ECDSA_HIT              EQU     2       ECDSA hit
PEP_COM_ERDSA_HIT              EQU     3       ERDSA hit
PEP_COM_RDSA_HIT               EQU     4       RDSA hit
PEP_COM_EUDSA_HIT              EQU     5       EUDSA hit
PEP_COM_UDSA_HIT               EQU     6       EUDSA hit
*
PEP_COM_SPACE                  DS      X       Subspace/basespace
PEP_COM_NOSPACE                EQU     0
PEP_COM_SUBSPACE               EQU     10      abending task was in
*                               subspace
PEP_COM_BASESPACE              EQU     11      abending task was in
*                               basespace
PEP_COM_PADDING                DS      CL2     Reserved
*
*                               Return code
*
PEP_COM_RETURN_CODE            DS      F
PEP_COM_RETURN_OK              EQU     0
PEP_COM_RETURN_DISABLE         EQU     4      Disable transaction
*
+ ***** APAR PN71385. Documentation added on 22 June 1995. *****
+ * Additional Program status word information
+ *
+ PEP_COM_INT                   DS      CL8     PSW interrupt codes
*
*                               length of DFHPEP_COMMAREA
PEP_COM_LEN EQU *-PEP_COM_STANDARD

```

Figure 15 (Part 2 of 2). Source of DFHPEP communication area (assembler-language)

The sample programs and copy books

Two source-level versions of the default program are provided: DFHPEP, coded in assembler language, and DFHPEPD, coded in C. Both are in the CICS410.SDFHSAMP library. There is an assembler-language macro, DFHPCOM, and a corresponding C copy book, DFHPCOMD, that you can use to define the communication area. These are found in the CICS410.SDFHMAC and CICS410.SDFHC370 libraries, respectively.

You can code your program error program in any of the languages supported by CICS, but you must always name it DFHPEP.

the program error program

Chapter 7. Writing a transaction restart program

The transaction restart user-replaceable program (DFHREST) enables you to participate in the decision as to whether a transaction should be restarted or not. The default program requests restart under certain conditions; for example, in the event of a program isolation deadlock (that is, when two tasks each wait for the other to release a particular DL/I database segment), one of the tasks is backed out and automatically restarted, and the other is allowed to complete its update.

For general information about restarting transactions, see the *CICS/ESA Recovery and Restart Guide*.

Notes:

1. CICS invokes DFHREST only when RESTART(YES) is specified in a transaction's resource definition.
2. When transaction restart occurs, a new task is attached that invokes the initial program of the transaction. This is true even if the task abended in the second or subsequent LUW, and DFHREST requested a restart.
3. Statistics on the total number of restarts against each transaction are kept.
4. Emergency restart does not restart any tasks.
5. Making a transaction restartable involves slightly more overhead than dynamic transaction backout because more items are logged; such items are logged only on the dynamic log.
6. In some cases, the benefits of transaction restart can be obtained instead by using the SYNCPOINT ROLLBACK command. Although use of the ROLLBACK command is not usually recommended, it does keep all the executable code in the application programs (except for DFHDBP exit code). For more information about the use of the ROLLBACK option when working in an ISC or MRO environment, see the *CICS/ESA Intercommunication Guide*.

When planning to replace the default DFHREST, check to see if the logic of any of your transactions is inappropriate for restart.

- Transactions that execute as a single logical unit of work are safe. Those that execute a loop, and on each pass reading one record from a recoverable destination, updating other recoverable resources, and closing with a syncpoint, are also safe.
- There are two types of transaction that need to be modified to avoid erroneously repeating work done in the logical units of work that precede an abend:
 1. A transaction in which the first and subsequent logical units of work change different resources
 2. A transaction where the contents of the input data area are used in several logical units of work.

the transaction restart program

All the following conditions must be true for CICS to invoke the transaction restart program:

- A transaction must be terminating abnormally.
- The transaction abend which caused the transaction to be terminating abnormally must have been detected before the commit point of the implicit syncpoint at the end of the transaction has been reached.
- The transaction must be defined as restartable in its transaction definition.
- The transaction must be related to a principal facility.

If these conditions are satisfied, CICS invokes the transaction restart program, which then decides whether or not to request that the transaction be restarted. CICS can subsequently override the decision (for example, if dynamic backout fails). Also, if the transaction restart program abends, the transaction is not restarted.

If the above conditions are not satisfied, CICS does not invoke the transaction restart program and the transaction is not restarted.

The DFHREST communications area

The CICS-supplied default transaction restart program is written in assembler and contains logic to:

- Address the communications area passed to it by CICS
- Decide whether or not to request transaction restart
- Send a message to CSMT if restart is requested
- Return control to CICS using the EXEC CICS RETURN command.

The communications area is mapped by the `XMRS_COMMAREA` DSECT, which is supplied in the `DFHXMRS` copybook. The equivalent structures for C/370, COBOL, VS COBOL II, and PL/1 are contained in the copybooks `DFHXMRS`, `DFHXMRSO`, and `DFHXMRSR`, respectively.

The information passed in the communications area is as follows:

XMRS_FUNCTION

Indicates, in a 1-byte field, the function code for this call to the restart program. This is always set to 1, which equates to `XMRS_TRANSACTION_RESTART`, which means that DFHREST is called to handle transaction restart.

XMRS_COMPONENT_CODE

Indicates, in a 2-byte field, the component code of the caller. This is always set to XM, which equates to `XMRS_TRANSACTION_MANAGER`. The transaction manager is the CICS component that coordinates the decision whether or not to restart a transaction.

XMRS_READ

Indicates, in a 1-byte field, whether the transaction has issued any terminal read requests, other than for initial input.

The equated values for this parameter are:

XMRS_READ_YES

Means a terminal read has been performed by the transaction.

XMRS_READ_NO

Means no terminal read has been performed.

XMRS_WRITE

Indicates, in a 1-byte field, whether the transaction has issued any terminal write requests.

The equated values for this parameter are:

XMRS_WRITE_YES

Means a terminal write has been performed by the transaction.

XMRS_WRITE_NO

Means a terminal write has not been performed by the transaction.

XMRS_SYNCPOINT

Indicates, in a 1-byte field, whether the transaction has performed any syncpoints.

The equated values for this parameter are:

XMRS_SYNCPOINT_YES

Means one or more syncpoints have been performed.

XMRS_SYNCPOINT_NO

Means no syncpoints have been performed.

XMRS_RESTART_COUNT

This indicates, as an unsigned, half-word binary value, the number of times the transaction has been restarted.

It is zero if the transaction has not been restarted. It is **not** the total number of restarts for the transaction definition. Rather it is the total number of restarts for transactions that are attempting, for example, to process a single piece of operator input.

XMRS_ORIGINAL_ABEND_CODE

Provides the first abend code recorded by the transaction.

XMRS_CURRENT_ABEND_CODE

Provides the current abend code. The values of the original abend code and the current abend code can be different if, for example, a transaction handles an abend and then abends later.

XMRS_RESTART

This is a 1-byte output field that the transaction restart program sets to indicate whether it wants CICS to restart the transaction.

The equated values for this field are:

XMRS_RESTART_YES Requests a restart.

XMRS_RESTART_NO Requests no restart.

the transaction restart program

The CICS-supplied default transaction restart program requests that the transaction be restarted if:

1. The transaction has not performed a terminal read (other than reading the initial input data), terminal write or syncpoint, **and**
 2. The restart count is less than 20 (to limit the number of restarts), **and**
 3. The current abend code is either ADCD or AFCF, indicating that the transaction was abended due to a DL/I deadlock or a file control-detected deadlock, respectively.
- +
+
+

The source of the CICS-supplied default transaction restart program, DFHREST, is supplied in assembler language only, in the CICS410.SDFHSAMP library.

The assembler copybook for mapping the communications area is in the CICS410.SDFHMAC library.

Chapter 8. Writing a terminal error program

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

This chapter contains information about the CICS terminal error program (TEP), which handles error conditions for devices that use the TCAM DCB interface or the sequential access method. **Note that node error programs, not terminal error programs, must be used for VTAM-supported devices.** The chapter is divided into three sections:

1. **“Background to error handling for TCAM and sequential devices”** is an overview.
2. **“The sample terminal error program”** on page 363 describes the CICS-supplied sample TEP. It contains:
 - “Components of the sample terminal error program” on page 364
 - “Structure of the sample terminal error program” on page 365
 - “Sample terminal error program messages” on page 369
 - “Generating the sample terminal error program” on page 371.
3. **“User-written terminal error programs”** on page 384 discusses factors you need to consider when writing your own terminal error program. It contains:
 - “Why write your own terminal error program?” on page 384
 - “Restrictions on the use of EXEC CICS commands” on page 385
 - “Addressing the contents of the communication area” on page 385
 - “Addressing the contents of the TACLE” on page 387
 - “Example of a user-written terminal error program” on page 391.

Background to error handling for TCAM and sequential devices

CICS terminal error handling is based on the assumption that most users want to modify CICS operations in response to terminal errors. Because CICS cannot anticipate all possible courses of action, the error-handling facilities have been designed to allow maximum freedom for users to create unique solutions for errors that occur within a terminal network.

The following CICS components are involved in the detection and correction of errors that occur when TCAM terminals or sequential devices are used:

- Terminal control program (DFHTCP)
- Terminal abnormal condition program (DFHTACP)
- Terminal error program (DFHTEP).

These components are discussed in the following sections. (The corresponding CICS components for logical units are discussed in Chapter 9, “Writing a node error program” on page 395.)

When an abnormal condition occurs

When an abnormal condition associated with a particular terminal or line occurs, the terminal control program puts the terminal out of service and passes control to the terminal abnormal condition program (DFHTACP) which, in turn, passes control to a version of the terminal error program (DFHTEP, either CICS-supplied or user-written), so that it can take the appropriate action.

Terminal control program

When the terminal from which the error was detected has been put out of service, the terminal control program creates a terminal abnormal condition line entry (TACLE), which is chained off the real entry, the terminal control table line entry (TCTLE) for the line on which the error occurred. The TACLE contains information about the error.

Terminal abnormal condition program

After the TACLE has been established, a task that executes DFHTACP is attached by the terminal control program and is provided with a pointer to the real line entry (TCTLE) on which the error occurred. After performing basic error analysis and establishing the default actions to be taken, DFHTACP gives control to DFHTEP, and passes a communication area (DFHTEPCA) so that DFHTEP can examine the error and provide an alternative course of action.

The communication area provides access to all the error information necessary for correct evaluation of the error; and contains special action flags that can be manipulated to alter the default actions previously set by DFHTACP.

After DFHTEP has performed the desired function, it returns control to DFHTACP by issuing an EXEC CICS RETURN command. DFHTACP then performs the actions dictated by the action flags within the communication area, and the error-handling task terminates.

Notes:

1. DFHTACP default actions, message codes, error codes, and conditions are listed in the *CICS/ESA Problem Determination Guide*.
2. If DFHTACP has more than eight errors on a line before action can be taken, the line is put out of service to avoid system degradation.

Terminal error program

The terminal error program analyzes the cause of the terminal or line error that has been detected by the terminal control program. The CICS-supplied version (the sample terminal error program, DFHXTEP) is designed to attempt basic and generalized recovery actions. A user-written version of this program can be provided to handle specific application-dependent recovery actions. The user-written terminal error program is linked-to in the same way as the CICS-supplied version, by the terminal abnormal condition program. Information relating to the error is carried in the communication area and the TACLE.

The macros that are provided for generating the sample terminal error program are described in the sections that follow. The main steps are generating the sample DFHTEP module and tables by means of the DFHTEPM and DFHTEPT macros,

respectively. You can select the appropriate options in this sample program, and you can base your own version on it.

There is a description of the CICS-supplied sample terminal error program (DFHXTEP), and advice about how to generate a user-written version, later in this chapter.

Note: If DFHTEP abends, then the default actions specified in DFHTACP are reinstated.

The communication area

The communication area is the basic interface used by the sample DFHTEP, and should be used by a user-written DFHTEP to:

- Address the TACLE
- Indicate the course of action to be taken on return to DFHTACP.

Before giving control to DFHTEP, DFHTACP establishes which default actions should be taken. This depends on the particular error condition that has been detected. The default actions are indicated by appropriate bit settings in the 1-byte communication area field TEPCAACT. For details about communication area fields, default actions, and bit settings, refer to “User-written terminal error programs” on page 384.

Terminal abnormal condition line entry (TACLE)

The TACLE contains further information about the type of error, and about the type of terminal that is in error.

The code indicating the detected error condition is passed to DFHTEP in the 1-byte field of the TACLE labeled TCTLEPFL. (These DFHTACP error codes, message codes, conditions, and default actions are also listed in the *CICS/ESA Problem Determination Guide*.)

A format description of the terminal abnormal condition line entry (TACLE) DSECT is provided under “User-written terminal error programs” on page 384.

The sample terminal error program

CICS provides a sample terminal error program that can be used as a generalized program structure for handling terminal errors. Note that, although the source code form of the sample TEP (DFHXTEP) is provided in assembler language only, you can write your own terminal error program in any of the languages supported by CICS.

After DFHXTEP has been assembled, it is link-edited as DFHTEP. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to “Assembling and link-editing user-replaceable programs” on page 344.

You can generate and use the sample terminal error program with the default options provided, or you can customize the terminal error support to the needs of the operating environment by selecting the appropriate generation options and variables. Because each error condition is processed by a separate routine, you

the sample terminal error program

can replace a CICS-provided routine with a user-written one when the sample TEP is generated.

Components of the sample terminal error program

The sample terminal error program consists of the terminal error program itself and two terminal error program tables:

- The TEP error table
- The TEP default table.

Both tables contain “thresholds” defined for the various error conditions to be controlled and accounted for by the sample DFHTEP. A “threshold” may be thought of as the number of error occurrences that are permitted for a given type of error on a given terminal before the sample DFHTEP accepts the DFHTACP default actions. Optionally, the number of occurrences can be controlled and accounted for over prescribed time intervals (for example, if more than three of a given type of error occur in an hour, the terminal is put out of service).

TEP error table

The terminal error program (TEP) error table maintains information about errors that have occurred on a terminal. The table consists of two parts (shown in Figure 16):

- The TEP error table header (TETH), which contains addresses and constants related to the location and size of the TEP error table components.
- Terminal error blocks (TEBs), which can be either:
 - Permanent (P-TEBs), each associated with a particular terminal
 - Reusable (R-TEBs), not permanently associated with any particular terminal.

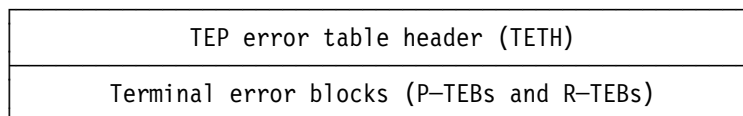


Figure 16. TEP error table

TEBs maintain error information associated with terminals. You must specify the total number of TEBs to be generated. The maximum number needed is one per terminal. In this case the TEBs are permanent.

You can reduce the total amount of storage used for TEBs by allocating a pool of reusable TEBs, that are not permanently associated with a particular terminal. Reusable TEBs are assigned dynamically on the first occurrence of an error associated with a terminal, and are released for reuse when the appropriate error processor places the terminal out of service.

Note: It is your responsibility to ensure that the pool is large enough to hold the maximum number of terminals for which errors are expected to be outstanding at any one time. If the pool limit is exceeded, handling of terminal errors may become intermittent. **No warning is given of this condition.**

You should assign permanent TEBs to terminals that are critical to the network. For the remainder of the network, you can generate a pool of reusable TEBs.

Each TEB currently in use or permanently assigned contains the symbolic terminal identifier of the terminal, and one or more error status elements (ESEs), as shown in Figure 17.

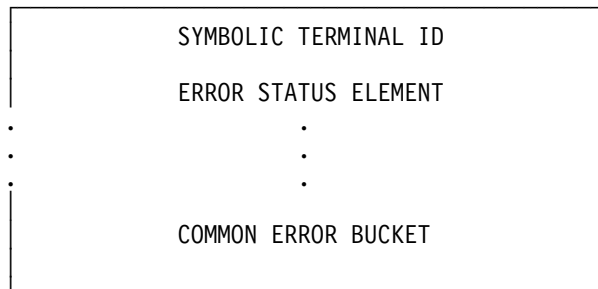


Figure 17. Terminal error block (TEB)

An ESE records the occurrence of a particular type of error associated with the terminal. The contents of an error status element are described in the TEP CD DSECT (generated by the DFHTEPM TYPE=INITIAL macro) under the comment “ERROR STATUS ELEMENT FORMAT”. The number of ESEs per TEB remains constant for all TEBs. You specify the number when the TEP tables are generated. If fewer than the maximum number of error types recognized by DFHTACP (25) are specified, one additional ESE, referred to as the “common error bucket”, is generated for each TEB.

You can permanently reserve ESE space in each TEB for specific error types. Those not permanently reserved are considered reusable, and are assigned dynamically on the first occurrence of a particular error type associated with the terminal. If an error type occurs that is not currently represented by an ESE, and if all reusable ESEs are assigned to other error types, the occurrence of this error is recorded in the common error bucket. DFHTACP can recognize far more error types than can occur in a typical terminal network. By specifying less than the maximum and allowing the sample DFHTEP to assign ESEs dynamically, you can minimize the table size, and still control and account for the types of errors relevant to the network.

TEP default table

The terminal error program (TEP) default table contains the “number and time” thresholds for each type of error to be controlled and accounted for. An index array at the beginning of the default table serves a dual function. If the value in the index is positive, then the error code has a permanently defined ESE in each TEB and the index value is the displacement to the reserved ESE. If the index value is negative, then an ESE must be assigned dynamically from a reusable ESE if one has not already been created by a prior occurrence. The complement of the negative index value is the displacement to the thresholds for the error type retained in the TEP default table.

Structure of the sample terminal error program

The structure of the sample terminal error program (DFHXTEP) can be broken down into six major areas as follows:

1. Entry and initialization
2. Terminal identification and error code lookup
3. Error processor selection

the sample terminal error program

4. Error processing execution
5. General exit
6. Common subroutines.

These areas are described in detail in the sections that follow.

Figure 18 on page 368 gives an overview of the structure of the sample terminal error program.

Entry and initialization

On entry, the sample TEP uses DFHEIENT to establish base registers and addressability to EXEC Interface components. It obtains addressability to the communication area passed by DFHTACP by means of an EXEC CICS ADDRESS COMMAREA, and addressability to the EXEC interface block with an EXEC CICS ADDRESS EIB command. It gets the address of the TACLE from the communication area, and establishes access to the TEP tables with an EXEC CICS LOAD. If time support has been generated, the error is time-stamped for subsequent processing. (Current time of day is passed in the communication area.) The first entry into the sample TEP after the system is initialized causes the TEP tables to be initialized.

Terminal ID and error code lookup

After the general entry processing, the TEP error table is scanned for a terminal error block (TEB) entry for the terminal associated with the error. If no matching entry is found, a new TEB is created. If all TEBs are currently in use (if no reusable TEBs are available), the processing is terminated and a RETURN request is issued, giving control back to DFHTACP, where default actions are taken.

After the terminal's TEB has been located or created, a similar scan is made of the error status elements (ESEs) in the TEB to determine whether the type of error currently being processed has occurred before, or if it has permanently reserved ESE space. If an associated ESE is not found, an ESE is assigned for the error type from a reusable ESE. If a reusable ESE does not exist, the error is accounted for in the terminal's common error bucket. The addresses of the appropriate control areas (TEB and ESE) are placed in registers for use by the appropriate error processor.

Error processor selection

User-specified message options are selected and the messages are written to a specified transient data destination. The type of error code is used as an index to a table to determine the address of an error processor to handle this type of error. If the error code is invalid, or the sample TEP was not generated to process this type of error, the address points to a routine that optionally generates an error message and returns control to DFHTACP, where default actions are taken. If an address of a valid error processor is obtained from the table, control is passed to that routine.

Error processing execution

The function of each error processor is to determine whether the default actions established by DFHTACP for a given error, or the actions established by the error processor, are to be performed. The common error bucket is processed by the specific error processor. However, the thresholds of the common error bucket are used in determining whether the limit has been reached. Subroutines are provided in the sample TEP to maintain count and time threshold totals for each error

associated with a particular terminal to assist the error processor to make its decision. Also available are subroutines for logging the status of the error and any recovery action taken by the error processor.

You can replace any of the error processors supplied in the sample TEP with user-written ones. Register linkage conventions, error conditions, DFHTACP default actions, and sample TEP error processor actions are described in comments given in the sample DFHXTEP source listing. However, sample DFHXTEP actions, in many cases, can be altered by changing the thresholds when generating the TEP tables.

General exit

Control is passed to a general exit routine from each error processor. This routine determines whether the terminal is to remain in service. If the terminal is to be put out of service, the terminal error block and all error status elements for that terminal are deleted from the TEP error table unless the terminal was defined as a permanent entry. When the terminal is placed back in service, a new terminal error block is assigned if a subsequent error occurs.

Common subroutines

A number of subroutines are provided in the sample DFHXTEP for use by the error processors. Each subroutine entry has a label of the form "TEPxxxx" where "xxxx" is the subroutine name. All labels within a subroutine start with TEPx where "x" is the first character of the subroutine name. All subroutines are arranged within the module in alphabetical order in the subroutine section. Register conventions and use of the subroutine are given as comments at the beginning of each subroutine in the source listing.

The following subroutines are available for writing your own error processors:

TEPACT

Used to output the names of the action bits set by DFHTACP and the sample DFHTEP in the communication area field **TEPCA** if appropriate PRINT options are selected when the program is generated.

TEPDEL

Used to delete the terminal error block and error status elements for a terminal from the TEP error table on exit from an error processor.

TEPHEXCN

Used by TEPPUTTD to convert a 4-bit hexadecimal value to its 8-bit printable equivalent.

TEPINCR

Used to update and test the count and time threshold totals maintained in the terminal's error status element.

TEPLOC

Used to locate or assign terminal error blocks and error status elements for a terminal ID.

TEPPUTTD

Used to output character or hexadecimal data to a user-defined transient data destination.

the sample terminal error program

TEPTMCHK

Used by TEPINCR to determine whether the time threshold has been passed.

TEPWGHT

Used to update the weight/time threshold values maintained in the terminal's error status elements.

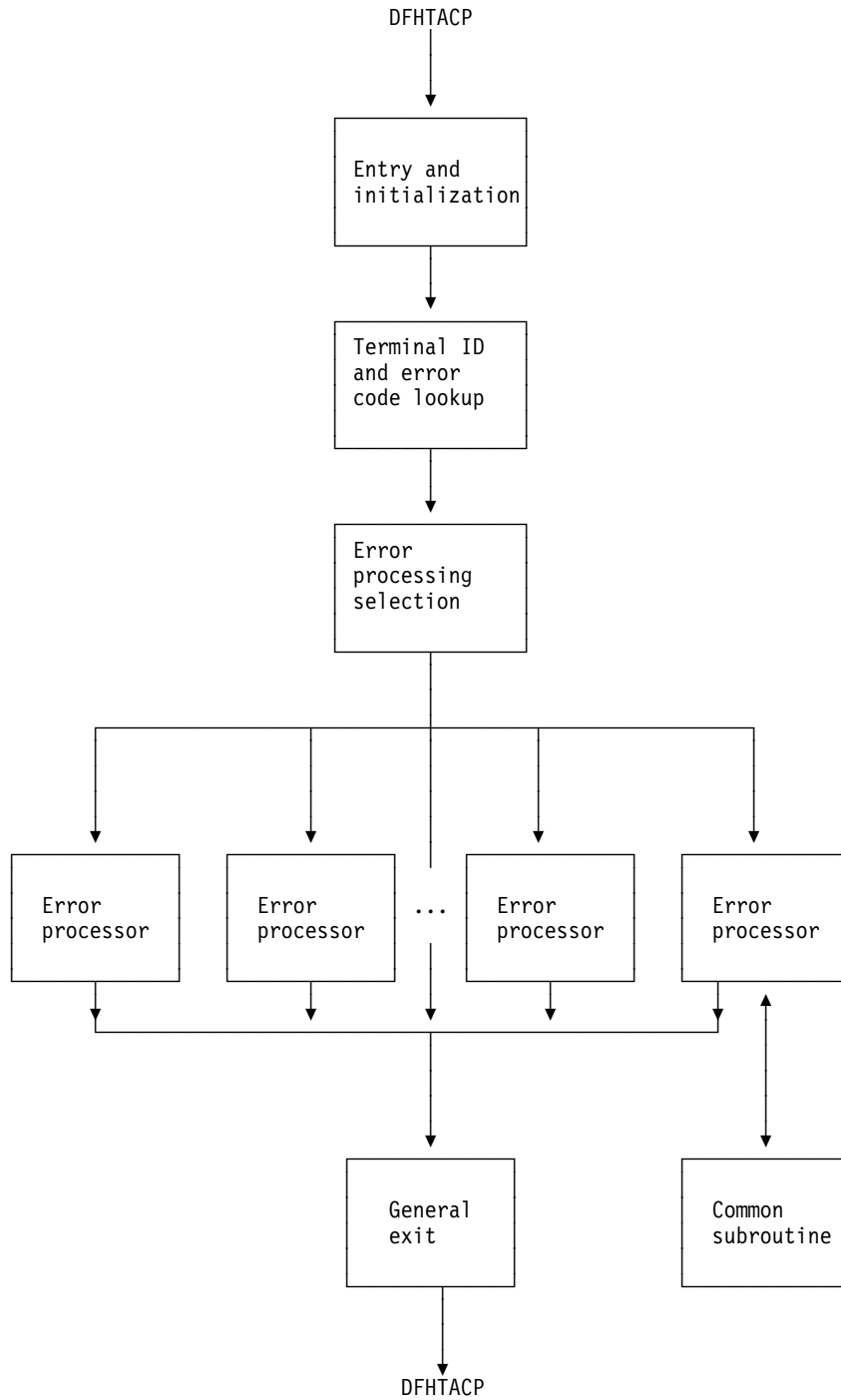


Figure 18. Overview of the sample terminal error program (DFHXTEP)

Sample terminal error program messages

The messages logged to the transient data destination CSMT (or, optionally, to the destination specified in the OPTIONS operand of DFHTEPM TYPE=INITIAL) are of six types, each identified by a unique message prefix. You can control the selection of each type of message by using the appropriate parameters specified on the PRINT operand of DFHTEPM TYPE=INITIAL.

These messages are:

DFHTEP, ERROR – error text

During DFHTEP module generation, the PRINT parameter specified ERRORS. This message can be suppressed by using the NOERRORS option. The error text is one of the following:

Unsupported error code, “xx”

The error code presented to DFHTEP by DFHTACP is unknown to DFHTEP.

“DFHTEPT” not defined in system

The DFHTEP table could not be loaded into storage.

Unknown error status message, “xxxx”

The error status message presented from a remote 3270 type device could not be decoded.

None of these errors should occur.

DFHTEP, ACTION – action flag names

During DFHTEP module generation, the PRINT parameter specified TACP ACTION or TEP ACTION or both. If both are specified, this message is logged twice each time DFHTEP is called. The first message indicates the action flags as set by DFHTACP on entry to DFHTEP. The second message indicates the action flags as returned to DFHTACP by DFHTEP after error processing. These messages can be suppressed by using the NOTACP ACTION and NOTEP ACTION options.

The action flag names and descriptions are listed below. For further information about the actions taken by DFHTACP, see the description of the TEPCA ACT field in “Addressing the contents of the communication area” on page 385.

Flag name	Description
LINEOS	Place line out of service
NONPRGT	Nonpurgeable task exists on terminal
TERMOS	Place terminal out of service
ABENDT	Abend task on terminal
ABORTWR	Abort write, free terminal storage
RELTTIOA	Release TCAM incoming message
SIGNOFF	Sign off terminal.

DFHTEP, TID – tid

During the DFHTEP module generation, the PRINT parameter specified TID. This message contains the symbolic terminal ID of the device associated with the error. This message can be suppressed by using the NOTID option.

DFHTEP, DECB – DECB information

During the DFHTEP module generation, the PRINT parameter specified DECB. This two-line message contains the DECB (printed in hexadecimal format) of the terminal causing the error. The DECB is contained in the TACLE (displacement +16 [decimal]). See the TACLE DSECT described in “User-written terminal error programs” on page 384. This message can be suppressed by using the NODECB option.

DFHTEP, TACLE – TACLE information

During the DFHTEP module generation, the PRINT parameter specified TACLE. This message (printed in hexadecimal format) contains the first 16 bytes of the TACLE passed to DFHTEP by DFHTACP. See the TACLE DSECT described in “User-written terminal error programs” on page 384. This message can be suppressed by using the NOTACLE option.

DFHTEP, ESE – ESE information

During the DFHTEP module generation, the PRINT parameter specified ESE. This message contains the error status element. The message can be suppressed by using the NOESE option.

An ESE is either 6 bytes or 12 bytes long, depending on whether the TIME option was specified when generating the TEP tables. The formats are as follows:

Table 104. Format of error status element on DFHTEP, ESE messages—NOTIME specified

NOTIME	Display	Length (bytes)	Significance
	0	2	Error threshold counter or weight value in binary format
	2	2	Current error count or weight value in binary
	4	1	Error code
	5	1	Not used.

Table 105. Format of error status element on DFHTEP, ESE messages—TIME specified

TIME	Display	Length (bytes)	Significance
	0	5	Error threshold counter or weight value in binary format
	5	3	Timed threshold value in hundredths of a second
	8	4	Time of first occurrence of this error. Time given as binary integer in hundredths of a second.

Generating the sample terminal error program

For information about how to generate the sample terminal error program and the sample terminal error table, refer to “Assembling and link-editing user-replaceable programs” on page 344.

The sample program and tables provide you with default error processing for terminal errors. If you want to replace the supplied error processors with user-written error processors, you must use the DFHTEPM and DFHTEPT macros to generate a sample error program and tables that include your user-written routines. Some of the parameters specified in the DFHTEPM and DFHTEPT macros are related and care must be taken to ensure compatibility. The parameters concerned are identified in the descriptions of the macros later in this chapter.

If you use the sample terminal error program (DFHXTEP), you can generate the required program and transaction definitions by using the CEDA INSTALL GROUP(DFHSTAND) command.

Job control for generating the sample terminal error program

The generation of the sample terminal error program consists of two separate assembly and link-edit steps, one to create the sample TEP module itself, and the other to create the TEP tables. The names under which the components must be link-edited are:

DFHTEP Sample TEP module, assembled from DFHXTEP
DFHTEPT Sample TEPT table, assembled from DFHXTEPT.

For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to “Assembling and link-editing user-replaceable programs” on page 344.

DFHTEPM—generating the sample DFHTEP module

The sample DFHTEP module is generated by the following macros:

- DFHTEPM TYPE=USTOR—to indicate the start of user storage definitions.
- DFHTEPM TYPE=USTOREND—to indicate the end of user storage definitions.
- DFHTEPM TYPE=INITIAL—to control the printing of CICS DSECTs, provide optional routines, and indicate the type of information to be logged when errors occur.
- DFHTEPM TYPE=ENTRY—to code a user “ENTRY” routine.
- DFHTEPM TYPE=EXIT—to code a user “EXIT” routine.
- DFHTEPM TYPE=ERRPROC—to allow you to replace the error processors supplied with the sample terminal error program with user-written versions.
- DFHTEPM TYPE=FINAL—to indicate the end of the sample DFHTEP module.

Note: You must code the translator options NOPROLOG and NOEPILOG in your error processors if you use these macros.

```
DFHTEPM TYPE=USTOR
```

the sample terminal error program

This macro indicates the start of user storage definitions. It must be followed by your storage definitions, and then by DFHTEPM TYPE=USTOREND. If you use DFHTEPM TYPE=USTOR to define storage, then both it and DFHTEPM TYPE=USTOREND must be coded **before** DFHTEPM TYPE=INITIAL.

```
DFHTEPM TYPE=USTOREND
```

This macro indicates the end of user storage definitions. Its use is mandatory if DFHTEPM TYPE=USTOR has been coded. If you use DFHTEPM TYPE=USTOR to define storage, then both it and DFHTEPM TYPE=USTOREND must be coded **before** DFHTEPM TYPE=INITIAL.

```
DFHTEPM TYPE=INITIAL
  [,DSECTPR={YES|NO}]
  [,OPTIONS=( [TD] (TD,destid) |NOTD]
              [,EXITS|,NOEXITS]
              [,TIME|,NOTIME]
              [,TCAM|,NOTCAM] ) ]
  [,PRINT=( [ERRORS|NOERRORS]
            [,TACPACTION|,NOTACPACTION]
            [,TEPACTION|,NOTEPACTION]
            [,TD|,NOTID]
            [,DECB|,NODECB]
            [,TACLE|,NOTACLE]
            [,ESE|,NOESE] ) ] ]
```

TYPE=INITIAL

establishes the beginning of the generation of the sample DFHTEP module itself.

DSECTPR={YES|NO}

controls the printing of CICS DSECTs on the sample DFHTEP assembly listing. Its purpose is to reduce the size of the listing. The default is DSECTPR=YES.

YES

Printing of the DSECTs is allowed.

NO

Printing of selected CICS DSECTs is suppressed. This parameter should not be used under Assembler F.

OPTIONS=optional-routines

includes or excludes optional routines in the DFHTEP module. The parentheses are required even when only one option is specified. If this operand is omitted, all default options are generated.

TD|(TD, destid)|NOTD

specifies whether information regarding the errors is to be written to a transient data destination.

TD

The transient data output routine is to be generated. The implied transient data destination is CSMT.

(TD, destid)

The transient data output routine is to be generated. The messages are sent to the destination specified by “destid”, which must be defined in the destination control table.

NOTD

No messages are to be written to a transient data destination.

EXITS|NOEXITS

specifies whether “ENTRY” and “EXIT” user routine support is to be included.

EXITS

Branches are taken to ENTRY and EXIT routines before and after error processing. Dummy routines are provided if user routines are not used.

NOEXITS

No branches are taken to user routines.

TIME|NOTIME

specifies whether threshold tests are to be controlled over prescribed time intervals. An example might be putting a terminal out of service if more than three instances of a given type of error occur in one hour. The parameter must be the same as the OPTIONS operand in the DFHTEPT TYPE=INITIAL macro.

TIME

This type of threshold testing is supported.

NOTIME

This type of threshold testing is not supported.

TCAM|NOTCAM

specifies whether optional TCAM support is to be included.

TCAM

TCAM error code ‘9F’ is supported.

NOTCAM

TCAM error code ‘9F’ is not supported.

PRINT=print-information

specifies which types of information are to be logged to the transient data destination each time an error occurs. If NOTD is specified on the OPTIONS operand, all PRINT parameters default to NO. All PRINT parameters require the transient data output routine. The parentheses are required even when only one parameter is specified.

ERRORS|NOERRORS

specifies whether unprocessable conditions detected by the sample DFHTEP are to be recorded on the transient data destination.

ERRORS

Error messages are to be logged.

NOERRORS

No error messages are to be logged.

TACPACTION|NOTACPACTION

specifies whether DFHTACP default actions are to be recorded on the transient data destination.

TACPACTION

The default actions are logged.

NOTACPACTION

No default actions are logged.

TEPACTION|NOTEPACTION

specifies whether the actions selected as a result of sample DFHTEP processing are to be recorded on the transient data destination.

TEPACTION

The final actions are logged.

NOTEPACTION

No final actions are logged.

TID|NOTID

specifies whether the symbolic terminal ID of the terminal associated with an error is to be recorded on the transient data destination.

TID

The terminal ID is to be logged.

NOTID

No terminal IDs are to be logged.

DECB|NODECB

specifies whether the DECB of the line associated with error is to be recorded on the transient data destination.

DECB

The DECB is logged. The hexadecimal representation of the DECB is logged as two 24-byte messages.

NODECB

No DECB logging occurs.

TACLE|NOTACLE

specifies whether the TACLE prefix is to be recorded on the transient data destination.

TACLE

The 16-byte TACLE prefix as received from DFHTACP is logged.

NOTACLE

No TACLE prefix logging occurs.

ESE|NOESE

specifies whether the ESE associated with the error is to be recorded on the transient data destination.

ESE

The ESE, after being updated, and before being deleted (if the action puts the terminal out of service) is logged.

NOESE

No ESE logging occurs.

DFHTEPM TYPE=ENTRY and EXIT—for user entry and exit routines

The sample DFHTEP provides guidance about how to prepare error processor routines, particularly with regard to register and subroutine linkage conventions. The routines must also observe the following restrictions:

- The error processor must be coded in assembler language.
- The first executable statement in the routine must be labeled TEPCDxx, where “xx” is the error code specified in the DFHTEPM TYPE=ERRPROC, CODE=errcode macro, which follows.
- Register usage conventions and restrictions are stated in the sample DFHTEP source.
- The error processor must exit to the sample DFHTEP symbolic label TEPRET.

The macro required for a user “ENTRY” routine is:

```
DFHTEPM TYPE=ENTRY
```

This macro must be immediately followed by user “ENTRY” routine code, starting with the label “TEPENTRY” and ending with a BR 14 instruction.

The macro required for a user “EXIT” routine is:

```
DFHTEPM TYPE=EXIT
```

This macro must be immediately followed by user “EXIT” routine code, starting with the label “TEPEXIT” and ending with a BR 14 instruction.

DFHTEPM TYPE=ERRPROC—replacing error processors

The macro necessary to replace error processors supplied with the sample DFHTEP with user-written error processors is:

```
DFHTEPM TYPE=ERRPROC
        ,CODE=errcode
        (followed by the appropriate error
         processor source statements)
```

TYPE=ERRPROC

indicates that a CICS-supplied error processor routine is to be replaced with the user-written error processor that immediately follows the macro. This macro is optional; if used, it must follow the DFHTEPM TYPE=INITIAL macro. One DFHTEPM TYPE=ERRPROC macro must precede each user-written error processor source routine.

the sample terminal error program

CODE=errcode

is used to identify the error code assigned to the appropriate error condition. These codes are listed in Figure 22 on page 389. For example, the TCAM invalid destination would be entered as code '9F'.

DFHTEPM TYPE=FINAL—ending the sample DFHTEP module

The macro to terminate the sample DFHTEP module is:

```
DFHTEPM TYPE=FINAL
```

This is followed by an END DFHTEPNA statement.

DFHTEPM macro examples

1. The following is an example of the minimum number of statements required to generate a sample DFHTEP module:

```
DFHTEPM TYPE=INITIAL  
DFHTEPM TYPE=FINAL  
END DFHTEPNA
```

This example generates a sample DFHTEP module with CICS-supplied error processors and all default options. This is equivalent to the CICS-supplied sample terminal error program.

2. Figure 19 on page 377 is an example of a more tailored sample DFHTEP module. In this example no 3270 support is generated, but TCAM support is provided. All default types of information except for TACP and TEP actions are to be logged to the TEPQ transient data destination. The CICS DSECTs are not printed on the sample DFHTEP assembler-language listing. There are two error processor routines (codes '87' and '9F' respectively).


```

* GENERATE USER STORAGE

    DFHTEPM    TYPE=USTOR
USORFLD  DS  F
    DFHTEPM    TYPE=USTOREND

* MODULE SPECIFICATIONS

    DFHTEPM    TYPE=INITIAL,
                OPTIONS=((TD,TEPQ),N03270,EXITS),
                PRINT=(NOTEPACTION,NOTACPATION),
                DSECTPR=NO
*
*
*
*
* USER-SUPPLIED ERROR PROCESSORS

    DFHTEPM    TYPE=ERRPROC, CODE=87
TEPCD81  DS  0H
        -
        - error processor "87" source statements
        -
        B  TEPRET

    DFHTEPM    TYPE=ERRPROC, CODE=9F
TEPCD9C  DS  0H
        -
        - error processor "9F" source statements
        -
        B  TEPRET

* USER "EXIT" EXIT CODE

    DFHTEPM    TYPE=EXIT
TEPEXIT  DS  0H
        -
        -
Additional user source statements to be executed after
error processing:
        -
        -
        BR  R14

* CONCLUDE MODULE GENERATION

    DFHTEPM    TYPE=FINAL
END DFHTEPNA

```

Figure 19. Example of DFHTEPM macros used to generate a sample DFHTEP module

DFHTEPT—generating the sample DFHTEP tables

The following macros are required to generate the terminal error program tables:

- DFHTEPT TYPE=INITIAL—to establish the control section.
- DFHTEPT TYPE=PERMTID—to define permanently reserved terminal error blocks (TEBs) for specific terminals.
- DFHTEPT TYPE=PERMCODE|ERRCODE—to define permanently reserved error status elements (ESEs).
- DFHTEPT TYPE=BUCKET—to define specific error conditions to be accounted for in the common error bucket.
- DFHTEPT TYPE=FINAL—to end the set of DFHTEPT macros.

DFHTEPT TYPE=INITIAL—establishing the control section

The DFHTEPT TYPE=INITIAL macro necessary to establish the control section for the TEP tables is:

```
DFHTEPT  TYPE=INITIAL
          ,MAXTIDS=number
          [,MAXERRS={25|number}]
          [,OPTIONS={TIME|NOTIME}]
```

TYPE=INITIAL

establishes the beginning of the generation of the TEP tables.

MAXTIDS=number

specifies the total number of permanent and reusable terminal error blocks to be generated in the TEP error table. Permanent entries are defined by the DFHTEPT TYPE=PERMTID macro described later in this section. Any entries not defined as permanent are reused when the terminal is taken out of service, or are deleted at the request of an error processor. If an error occurs, and no TEB space is available, the error is not processed, and DFHTACP default actions are taken. The minimum number of blocks is 1. A maximum number is not checked for but should be no greater than the number of terminals in your network.

MAXERRS=25|number

specifies the number of errors to be recorded for each terminal. This value determines the number of permanent and reusable error status elements in each TEB. The maximum number that can be specified is 25 (the default value). If more are requested, only the maximum are generated. If fewer are requested, one extra ESE is generated for each TEB. The extra ESE is the common error bucket. Permanently reserved ESEs are defined by the DFHTEPT TYPE=PERMCODE macro described later in this section. Any ESEs not defined as permanent are dynamically assigned on the first occurrence of a nonpermanent error type associated with the terminal. By defining a number less than the maximum, and allowing the sample DFHTEP to assign ESEs dynamically, you can minimize the size of the table and still control and account for the error types relevant to the network. The minimum number that can be specified is zero. In this case only a common error bucket is generated.

OPTIONS=(TIME|NOTIME)

specifies whether time threshold space is to be reserved in support of the TIME option specified in the DFHTEPM TYPE=INITIAL macro. The default is OPTIONS=TIME.

TIME

Time threshold space is reserved.

NOTIME

Time threshold space is not reserved.

DFHTEPT TYPE=PERMTID—assigning permanent terminal error blocks

The DFHTEPT TYPE=PERMTID macro to define permanently reserved terminal error blocks for specific terminals is:

```
DFHTEPT TYPE=PERMTID
        ,TRMIDNT=name
```

TYPE=PERMTID

defines permanently reserved terminal error blocks for specific terminals. Permanent TEBs are defined for terminals that are critical to system operation to ensure that error processors are always executed in the event of errors associated with that terminal. If no permanent TEBs are to be defined, this macro is not required. A separate macro must be issued for each permanently reserved TEB. The maximum number of permanent TEBs is the number specified in the MAXTIDS operand of the DFHTEPT TYPE=INITIAL macro.

TRMIDNT=name

is used to provide the symbolic terminal ID (1–4 characters) for a permanently defined TEB. Only one terminal can be specified in each macro.

DFHTEPT TYPE=PERMCODE|ERRCODE—defining error status elements

The DFHTEPT TYPE=PERMCODE|ERRCODE macro is used to change the default threshold constants of the sample DFHTEP, and to define permanently reserved error status elements:

```
DFHTEPT TYPE={PERMCODE|ERRCODE}
        ,CODE={errcode|BUCKET}
        [,COUNT=number]
        [,TIME=(number[,SEC|,MIN|,HRS])]
```

TYPE={PERMCODE|ERRCODE}

identifies whether the error code specified in the macro is to have a permanently reserved or a dynamically assigned ESE. These macros are required only if permanently reserved ESEs are to be defined, or if the sample DFHTEP default threshold constants are to be overridden. These are listed in Table 106 on page 382.

PERMCODE

Identifies the error code specified as having a permanently reserved ESE. Each permanently reserved ESE must be identified by a separate DFHTEPT TYPE=PERMCODE macro. All DFHTEPT TYPE=PERMCODE macros must precede all DFHTEPT TYPE=ERRCODE macros.

ERRCODE

Indicates that the error code specified does not require a permanently reserved ESE, but that the sample DFHTEP default threshold constants are to be changed. Each error code requiring a threshold constant change, other than those defined as permanently reserved, must be identified by a separate DFHTEPT TYPE=ERRCODE macro. All DFHTEPT TYPE=ERRCODE macros must follow all DFHTEPT TYPE=PERMCODE macros.

CODE={errcode|BUCKET}

identifies the error code referred to by the TYPE=PERMCODE|ERRCODE parameter. These codes are listed in Figure 22 on page 389. CODE=BUCKET is only applicable to the DFHTEPT TYPE=ERRCODE macro. It is used to override the default threshold constants established for the common error bucket.

COUNT=number

can be used in either the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macro to override the sample DFHTEP default count threshold (see Table 106 on page 382). When the number of occurrences of the error type specified reaches the threshold, an error processor normally takes a logic path that causes DFHTACP default actions to be taken. If the number of occurrences is less than the threshold, the error processor normally takes a logic path that overrides the DFHTACP default actions. The updating and testing of the current threshold counts are normally performed by a DFHTEP subroutine that sets a condition code that the error processor can test to determine whether the limit has been reached. **If you specify 0 as the number in the COUNT operand, you are not told when the threshold is reached.**

TIME=(number[,SEC|,MIN|,HRS])

can be used in either the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macros to override the sample DFHTEP default time threshold (see Table 106 on page 382). This operand is only applicable when OPTIONS=TIME is specified on both the DFHTEPM and DFHTEPT TYPE=INITIAL macros. When the number of occurrences reaches the threshold specified on the COUNT operand (above) within the interval specified on this parameter, an error processor normally takes a logic path that causes DFHTACP default actions to be taken. If the number of occurrences within the interval is less than the threshold, the error processor normally takes a logic path that overrides the DFHTACP default actions. If the time interval has expired, the sample DFHTEP subroutine that normally updates and tests the current threshold count resets the occurrence counts, and establishes a new expiration time. In this case, the condition code set by the subroutine indicates that the thresholds had not been reached.

Time control in the sample DFHTEP starts with the first occurrence of an error type. Subsequent occurrences of the same error type **do not** establish new starting times, but are accounted for as having occurred within the interval started by the first occurrence. This continues until an error count reaches the threshold within the interval started by the first occurrence, or until the interval

has expired. In the latter case, the error being processed becomes a first occurrence, and a new interval is started. A time interval of 0 means that the number of occurrences is to be accounted for and controlled without regard to a time interval. Zero is the implied time interval if the value of the COUNT operand is 0 or 1. It is also the implied time interval if the time options are not generated.

The time interval can be expressed in any one of four units; hours, minutes, seconds, or hundredths of a second. The maximum interval must be the equivalent of less than 24 hours. A practical minimum would be 1 to 2 minutes. This allows for access method retries and the time required to create the task to service each error. The four methods of expressing the threshold time interval are:

number

The interval in units of one hundredth of a second. Parentheses are not required if this method is used. The maximum number must be less than 864000 (24 hours).

(number,SEC)

The interval in whole seconds, which must be enclosed in parentheses. The maximum number must be less than 86400 (24 hours).

(number,MIN)

The interval in whole minutes, which must be enclosed in parentheses. The maximum number must be less than 1440 (24 hours).

(number,HRS)

The interval in whole hours, which must be enclosed in parentheses. The maximum number must be less than 24.

Table 106 on page 382 illustrates the default thresholds of the sample terminal error program, referred to in the TYPE, COUNT, and TIME operands of the DFHTEPT TYPE=PERMCODE|ERRCODE macro.

Table 106. Default thresholds of the sample TEP

CODE=	COUNT=	TIME=
81	3	(7,MIN)
84	1	0
85	1	0
87 (Note 1)	50 (Note 3)	0
88	1	0
8C	1	0
8D	1	0
8E	1	0
8F	1	0
90	0	0
91	0	0
94	7	(10,MIN)
95 (Note 2)	0	0
96	2	(1,MIN)
97 (Note 2)	0	0
99	1	0
9F (Note 2)	0	0
BUCKET	5	(5,MIN)

Notes:

1. For TCAM conditions without TACP defaults, TEP retries five times and releases the TIOA. Otherwise the default TACP actions are taken.
2. The error processor maintains an error count only. DFHTACP default actions are always taken regardless of the thresholds.
3. The error processor uses a threshold "weight" instead of a threshold count (see the source code of the sample DFHTEP).

DFHTEPT TYPE=BUCKET—using the error bucket for specific errors

The DFHTEPT TYPE=BUCKET macro is used to ensure that specific error conditions are always accounted for in the common error bucket:

```
DFHTEPT TYPE=BUCKET
        ,CODE=errcode
```

TYPE=BUCKET

generates the macro to account for specific error conditions in the common error bucket. If MAXERR=25 on the DFHTEPT TYPE=INITIAL macro, this macro cannot be used. This macro is not required if no error codes are to be specifically accounted for in the common error bucket. Each error code must be identified by a separate DFHTEPT TYPE=BUCKET macro.

CODE=errcode

identifies the error code to be specifically accounted for in the common error bucket. The error code must not be specified in the DFHTEPT TYPE=PERMCODE or TYPE=ERRCODE macro.

DFHTEPT TYPE=FINAL—terminating DFHTEPT entries

The DFHTEPT TYPE=FINAL macro terminates the generation of the DFHTEPT tables.

```
DFHTEPT TYPE=FINAL
```

DFHTEPT—examples of how the macros are used

1. The following is an example of the minimum number of statements required to generate the TEP tables:

```
DFHTEPT TYPE=INITIAL,MAXTIDS=10
DFHTEPT TYPE=FINAL
END
```

This example generates 10 reusable terminal error blocks, each capable of accounting for the maximum number of error types. Time threshold control is supported, and all threshold values are the defaults supported by the sample DFHTEPT. This is equivalent to the CICS-supplied sample terminal error program.

2. Figure 20 is an example of a customized TEP table (continuation character omitted).

* TABLE SPECIFICATIONS

```
DFHTEPT TYPE=INITIAL,MAXTIDS=10,
MAXERRS=5
```

* PERMANENT TERMINAL DEFINITIONS

```
DFHTEPT TYPE=PERMTID,TRMIDNT=TM02
```

* PERMANENT ERROR CODE DEFINITIONS

```
DFHTEPT TYPE=PERMCODE,CODE=81
DFHTEPT TYPE=PERMCODE,CODE=87,
COUNT=2,TIME=(1,MIN)
```

* OTHER THRESHOLD OVERRIDES

```
DFHTEPT TYPE=ERRCODE,CODE=BUCKET,
COUNT=3,TIME=(3,MIN)
```

* CONCLUDE TABLE GENERATION

```
DFHTEPT TYPE=FINAL
END
```

Figure 20. Example of the use of DFHTEPT macros to generate DFHTEPT tables

user-written terminal error programs

This example generates 10 terminal error blocks, one of which is reserved for the terminal whose symbolic ID is TM02, and the other nine are reusable. Each TEB has space for five error status elements plus a common error bucket. Of the five ESEs, two are reserved for error codes '81' and '87'; the remaining ESEs are available to be assigned dynamically. The thresholds for error code '87' and the common error bucket are being changed. No specific error code is to be accounted for in the common error bucket.

User-written terminal error programs

You can write your own terminal error program in any of the languages supported by CICS. However, CICS-supplied code is provided in assembler language only. The names of the supplied source files and macros, and the libraries in which they can be found, are listed in Table 107.

Name	Type	Description	Library
DFHXTEP	Source	Sample terminal error program (assembler-language)	CICS410.SDFHSAMP
DFHXTEPT	CSECT	Sample terminal error tables (assembler-language)	CICS410.SDFHSAMP
DFHTEPM	Macro	Sample TEP program generator (assembler-language)	CICS410.SDFHMAC
DFHTEPT	Macro	TEP table generator (assembler-language)	CICS410.SDFHMAC
DFHTEPCA	Macro	Assembler-language communication area	CICS410.SDFHMAC

The user-written DFHTEP receives control in the same manner as the CICS-supplied sample DFHTEP, described in “The communication area” on page 363. It should therefore use the communication area as its basic interface with DFHTACP.

Why write your own terminal error program?

- There are some situations in which CICS may try to send a message to an input-only terminal; for example, an 'invalid transaction ID' message, or a message wrongly sent by an application program. You should provide a terminal error program to reroute these messages to a system destination such as CSMT or CSTL or other destinations, by means of transient data or interval control facilities.
- There could be application-related activity to be carried out when a terminal error occurs. For example, if a message is not delivered to a terminal because of an error condition, it may be necessary to notify applications that the message needs to be redirected.
- Not all errors represent communication-system failures – for example, SAM end-of-data conditions.

Restrictions on the use of EXEC CICS commands

There are certain restrictions on the commands that a TEP can issue. **The use of any commands that require a principal facility causes unpredictable results, and should be avoided.** In particular, you should not use commands that invoke the following functions:

- Terminal control (“CEMT-type” commands, such as EXEC CICS INQUIRE TERMINAL, *are* permissible)
- BMS (except routing)
- ISC communication (including function shipping).

Addressing the contents of the communication area

After your terminal error program receives control from DFHTACP, it should obtain the address of the communication area by means of an EXEC CICS ADDRESS COMMAREA command.

You generate the communication area DSECT by coding DFHTEPCA TYPE=DSECT in your program. The layout of the communication area is as follows:

			IN/OUT PARM		
		0XL4		Standard Header	
TEPCALDS	DS	XL1	I	Function Code	Always '1'
TEPCAGDS	DS	XL2	I	Component Code	Always 'TC'
		XL1		Reserved	
TEPCATCA	DS	A	I	Address of TACLE being processed	
TEPCECIA	DS	A	I	Address of TCTUA	
TEPCECIL	DS	H	I	Length of TCTUA	
TEPCAACT	DS	XL1	I/O	User action byte	
TEPCATID	DS	CL4	I	Terminal identity	
TEPCATDB	DS	F	I	Current time of day binary	

Figure 21. The DFHTACP/DFHTEP communication area

The parameter list contains the following information:

TEPCALDS Function Code. The Function Code is a printable character representing the identity of the task within the TCP which invoked DFHTEP. It always has the value '1'.

TEPCAGDS Component Code. This always has the value 'TC', representing a component of the TCP.

TEPCATCA Contains the address of the TACLE being processed.

TEPCECIA Contains the address of the terminal control table user area (TCTUA).

TEPCECIL Contains the length of the TCTUA.

TEPCAACT The User action byte. One of the main uses of the communication area is to transmit the actions that are to be taken for a terminal. TEPCAACT contains the following flags, which can be reset within DFHTEP:

LINEOS (X'80')	Place line out of service
NONPRGT (X'40')	Nonpurgeable task exists on the terminal
TERMOS (X'20')	Place terminal out of service
ABENDT (X'10')	Abend the task on the terminal
ABORTWR (X'08')	Abend write, free terminal storage
RELTIOA (X'04')	Release TCAM incoming message
SIGNOFF (X'02')	Call sign-off program.

On entry to DFHTEP, the above flags represent the default actions set by DFHTACP. The write-abend bit (communication area field **ABORTWR**) and the abend-task bit (communication area field **ABENDT**) are always set if the place-line-out-of-service bit (X'80') is set; but both bits are suppressed if "dummy terminal" is indicated (see Resetting the flags in the user action byte, TEPCAACT).

On return to DFHTACP, the flags represent the actions as modified by DFHTEP.

TEPCATID Contains the identity of the terminal in error.

TEPCATDB Contains the time of day when the error occurred, in binary format.

Resetting the flags in the user action byte, TEPCAACT

The following factors should be considered when altering the action bits in TEPCAACT:

- You should consider how to preserve data security. For example, if a terminal is put out of service for some time (until the cause of the failure is removed) the signon information is still in the TCTTE when the terminal is put back into service, although the original operator may no longer be present. To prevent a possible security violation, you can set the **SIGNOFF** bit to sign off the terminal.
- For TCAM unsolicited input errors with either the terminal out of service or in receive-only state, a loop occurs if the default action of purging the incoming message does not occur and the status of the terminal is not altered.
- The dummy terminal indicator at TCTLEPF2 is set on errors from which no specific terminal is indicated. Therefore, if a dummy terminal is indicated, abend task and abend write are not set (see below). The dummy terminal is only used to identify the line.
- The abend-task bit (X'10' in TEPCAACT) is always associated with two other bits as part of TACP's abend transaction processing. These other bits are nonpurgeable task and abend write (X'40' and X'08' respectively, both in TEPCAACT).
- Abend write is always set on at the same time as abend task. It has the effect of clearing the TCTTE of the original write request indicators, if the error being processed occurred on a TC WRITE.
- Nonpurgeable task is set on if a transaction is currently associated with the terminal, and the transaction ID was specified with TPURGE=NO.

None of the `abend-task`, `abend-write`, or `nonpurgeable-task` bits is set if the dummy terminal indicator is on, even if DFHTACP would normally set `abend task` as the default for the error being processed. Therefore, the following remarks apply only to errors related to a real terminal.

- `Abend task` has no effect if no transaction is associated with the terminal; (except where a pseudoconversational task is associated with the terminal, in which case, the next `transid` is cleared). Otherwise, if `nonpurgeable task` is indicated, the transaction remains attached to the terminal (normally in `SUSPEND` state) and DFHTACP writes the 'DFHTC2522 INTERCEPT REQUIRED' message to CSMT; if the task is not marked `nonpurgeable`, it is abended with code 'AEXY' or, rarely, 'AEXZ'.
- `Abend write` has no effect if the `TCTTE` was associated with a `READ` request. In this case the normal result is that, if the line and terminal remain in service, the read is retried.
- For TCAM pooled terminals, the DFHTACP default action of `ABENDT` is inappropriate for unsolicited input and 'invalid TCAM destination' types of error. `Abend task` results in the pooled terminal entry being placed out of service, which may lead to line lockouts as available in-service pooled entries become exhausted. You should therefore set the `ABENDT` action bit off in the TEP.

Addressing the contents of the TACLE

The TACLE is created by the terminal control program when the error occurs, and contains all the I/O error information provided by TCAM and BSAM.

To address the contents of the TACLE, the user-written terminal error program should contain the `COPY DFHTACLE` and `COPY DFHTCTLE` statements, in that order. These define the complete DFHTCTLE DSECT. The symbolic names in this DSECT are used to address fields in both the TACLE and the real line entry associated with the error.

The TACLE consists of a 16-byte prefix (defined by `COPY DFHTACLE`) and a further 48-byte section, which is a modified copy of the DECB of the real line entry at the time the TACLE was created.

To address the TACLE, the user-written terminal error program should therefore contain the statements:

```
COPY DFHTACLE
COPY DFHTCTLE

L TCTLEAR,TEPCATCA      POINT TO TACLE
USING DFHTCTLE,TCTLEAR
```

Note that fields normally part of the real line entry DECB have offsets increased by 16 in the TACLE.

The following fields in the DECB copy in the TACLE do **not** represent data copies from the real line entry:

```
TCTLEDCB                (Offset 24 in TACLE,
                        8 in real TCTLE)
```

user-written terminal error programs

This field in the TACLE points to the real line entry; in the real line entry, it points to the TCAM/BSAM DCB for the line group.

```
TCTLECSW          (Offsets 46, 48 in TACLE,  
TCTLEALP          30, 32 in real TCTLE)
```

These are used in the TACLE for SAM error information.

The following statements give direct addressability to the **real** line entry:

```
COPY DFHTCTLE  
COPY DFHTCTTE
```

```
L      TCTLEAR,TEPCATCA      POINT TO TACLE  
USING DFHTCTLE,TCTLEAR  
L      TCTTEAR,TCTLEPTE      POINT TO ERROR TCTTE  
USING DFHTCTTE,TCTTEAR  
DROP  TCTLEAR  
L      TCTLEAR,TCTTELEA      POINT TO TCTLE  
USING DFHTCTLE,TCTLEAR
```

After you have carried out the required functions and, optionally, altered the default actions scheduled by DFHTACP, the user-written DFHTEP must return control to DFHTACP by issuing the EXEC CICS RETURN command. DFHTACP then performs the actions specified in the TACLE and causes the error processing task to terminate.

The format of the TACLE DSECT is shown in Figure 22 on page 389.

TERMINAL ABNORMAL CONDITION LINE ENTRY

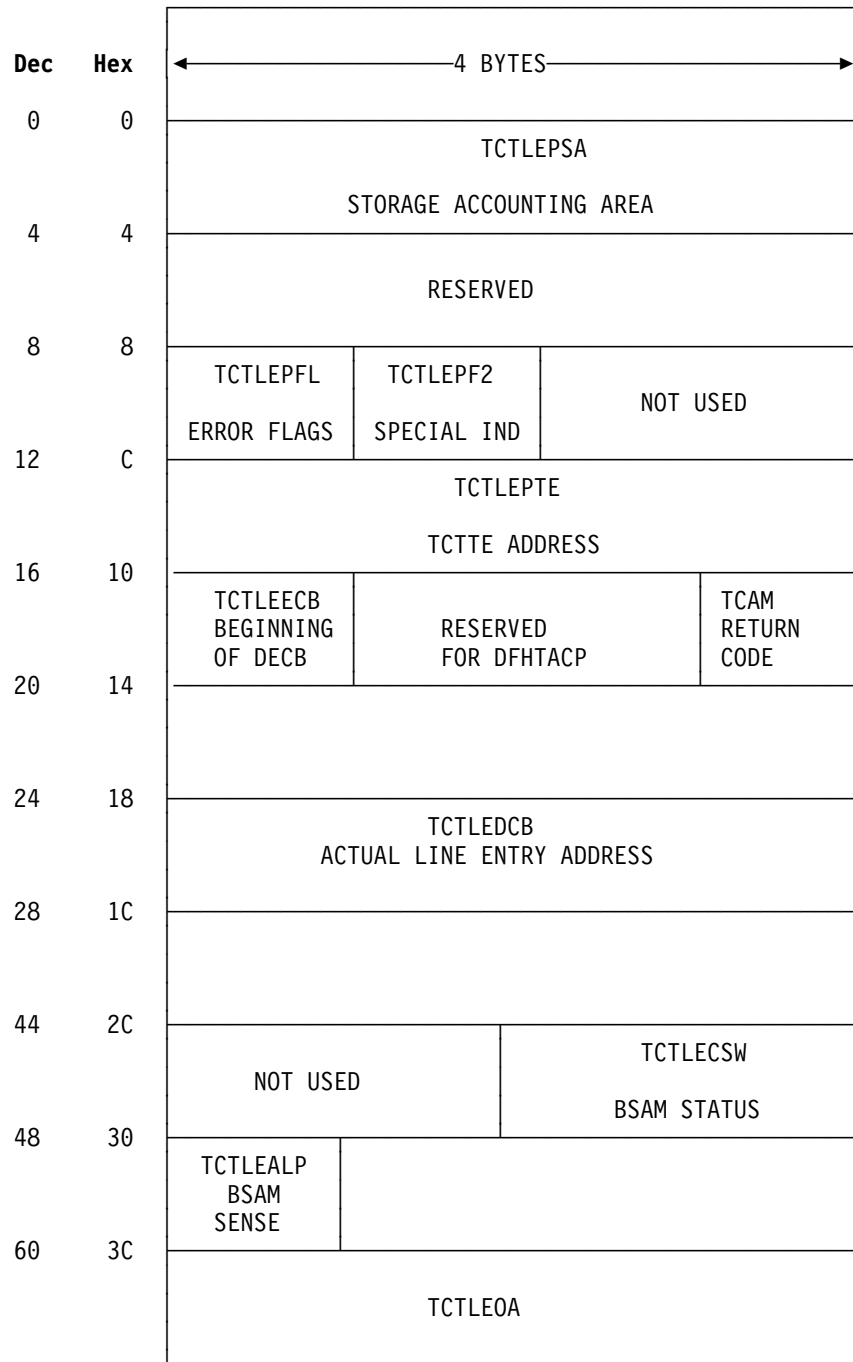


Figure 22 (Part 1 of 2). Format description of the TACLE DSECT

user-written terminal error programs

Displacement

Dec	Hex	Code	Bytes	Label	Meaning
0	0		4	TCTLEPSA	Storage accounting
				RESERVED	
8	8	81	1	TCTLEPFL	Error flags
					Message too long
		84			TCT search error
		85			Write not valid
		87			Unsolicited input
		88			Input event rejected
		8C			Output event rejected
		8D			Output length of zero
		8E			No output area
		8F			Output area exceeded
		94			Unit check
		95			Unit check (should not occur)
		96			Unit exception
		97			Unit exception (should not occur)
		99			Undetermined I/O error
		9F			Invalid destination (TCAM)
		.			
		.			(All codes not listed are reserved and are not intended for use by DFHTEP)
		.			
		.			
9	9	01	1	TCTLEPF2	Special indicator dummy terminal
12	C		4	TCTLEPTE	Address of terminal entry for terminal in error
16	10		4	TCTLEECB	DECBC/copy of line when error occurred
60	3C		4	TCTLEOA	For TCAM lines only. Address of the line I/O area containing the input or output message, or zero if none available.

Figure 22 (Part 2 of 2). Format description of the TACLE DSECT

Example of a user-written terminal error program

The “DFHTEP recursive retry routine” on page 392 is an example of the logic steps necessary to design a portion of the terminal error program. In Figure 23, 10 retries are provided for each terminal; however, the logic could be used for any number of retries. The following assumptions are made:

USER FIELD A (PCISAVE)

represents a 6-byte field in the process control information (PCI) area of the TCTTE. This field is used to preserve the count of input and output from the TCTTE when the first error occurs. These counts are contained in 3-byte fields located at TCTTENI and TCTTENI within the TCTTE.

USER FIELD B (PCICNT)

represents a user-defined field used to accumulate the count of recursive errors. It should be in the process control information (PCI) area of the TCTTE.

SYSTEM COUNT (TCTTENI)

represents the 6-byte field in the TCTTE that contains the terminal input and output counts (TCTTENI+TCTTENI). In the example, these two adjacent fields are considered as one 6-byte field.

Because this example requires access to the TCT terminal entry (TCTTE) to examine the SYSTEM COUNT and to locate the process control information (PCI) area, the DFHTCTTE symbolic storage definition is included so that fields can be symbolically referenced.

Note that the code in Figure 23 is intended only as an illustration of a recursive error handling technique and of the steps necessary to establish addressability to the applicable control blocks.

DFHTEP recursive retry routine

```

*ASM      XOPTS(NOPROLOG NOEPILOG SP)
*****
*
*                      DFHTEP RECURSIVE RETRY ROUTINE
*
*****
          DFHEISTG
          DFHEIEND
          DFHTEPCA TYPE=DSECT      COMMAREA passed by TACP
          COPY  DFHA06DS           Statistics DSECT
          USING DFHA06DS,STATBAR
PCIAREA  DSECT
PCISAVE  DS   XL6                  User Field A
PCICNT   DS   PL2                  User Field B
*
TCTLEAR  EQU   2                   Pointer to TACLE
STATBAR  EQU   4                   Pointer to statistics DSECT
TCTUABAR EQU   5                   Pointer to TCTUA
COMMABAR EQU  12                   Pointer to COMMAREA passed by TACP
          EJECT
DFHTEP   CSECT
*****
*      Establish addressability
*
*****
          DFHEIENT
*
          EXEC CICS ADDRESS EIB(11)
*
          EXEC CICS ADDRESS COMMAREA(COMMABAR)
*
          USING DFHTEPCA,COMMABAR
          L      TCTLEAR,TEPCATCA   Load TACLE address
*
          USING PCIAREA,TCTUABAR
          L      TCTUABAR,TEPCECIA  Load TCTUA address
*
*****
*      Start processing
*
*****
          TM      PCICNT+1,X'0C'    Has User Field B been initialized
*                                     to a packed decimal number?
          BO      CKCOUNT          YES .... so compare the system count
*                                     with the existing count in Field B
          RESET   DS      0H
          MVC     PCICNT,=PL2'+0'  NO .... so initialize field B to
*                                     packed zero.
*

```

Figure 23 (Part 1 of 2). DFHTEP recursive retry routine


```

EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*
*           Get statistics for this terminal
*           using TERMID passed in Commarea
*
MVC  PCISAVE,A06TENI  Save the current system counts. This
*
*           is a new error, or first time
*           through.
INCR  DS  0H
      AP  PCICNT,=P'1'  Increment the number of times this
*
*           error has occurred (recursive count)
*
      CP  PCICNT,=P'10'  Has the maximum recursive error
*
*           limit been reached?
      BNE  RETRY        NO .... set action
*
      ZAP  PCICNT,=P'0'  Clear and reset user fields for next
*
*           error set
EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*
*           Get statistics for this terminal
*           using TERMID passed in COMMAREA
*
MVC  PCISAVE,A06TENI  Get current system counts
      B   NORETRY      Action indicators for no retry
*
CKCOUNT DS  0H
EXEC CICS COLLECT STATISTICS TERMINAL(TEPCATID) SET(STATBAR)
*
*           Get statistics for this terminal
*           using TERMID passed in COMMAREA
*
      CLC  PCISAVE,A06TENI  Has system count changed since last
*
*           entry to TEP?
      BNE  RESET        YES .... this is a new error since
*
*           some I/O activity has occurred on
*           terminal.
      B   INCR        NO .... this is a recursive error,
*
*           so increment the recursive count and
*           check for retry.
RETRY  DS  0H
*
*           The user would include here the code
*           necessary to alter the flags in the
*           COMMAREA so that a retry can be
*           performed on the terminal.
NORETRY DS  0H
*
*           The user would include here the code
*           necessary to allow DFHTACP to take
*           final actions on the terminal; that
*           is, abend task, put line out of
*           service, and others.

LTORG ,
END

```

Figure 23 (Part 2 of 2). DFHTEP recursive retry routine

user-written terminal error programs

Chapter 9. Writing a node error program

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

This chapter contains information about the node error program (NEP) of CICS/ESA 4.1. Node error programs, not terminal error programs, must be used for terminals and logical units supported via the ACF/VTAM interface.

The chapter is divided into the following sections:

1. **“Background to CICS-VTAM error handling”** is an overview. If you are not familiar with the node error program, you should read this section. If you are familiar with NEPs, you may be able to go straight to the detailed information in the following sections, and look at the subjects that particularly interest you.
2. **“When an abnormal condition occurs”** on page 403 describes the CICS components that are invoked when an abnormal condition is detected from a VTAM logical unit.
3. **“The sample node error program”** on page 411 describes the CICS-supplied sample NEP.
4. **“User-written node error programs”** on page 421 discusses the factors you need to consider when writing your own node error program.
5. **“Using the node error program with XRF or persistent sessions”** on page 427.

Notes:

1. Like the terminal error program for non-VTAM devices, the node error program for VTAM-attached terminals is available in three forms:
 - a. The default node error program
 - b. The CICS-supplied sample node error program
 - c. User-written versions.

All three types are discussed in the following sections.
2. In this chapter, “VTAM 3270” refers to a non-SNA 3270 connected through VTAM, and “3270 compatibility mode” refers to an SNA 3270 connected through VTAM.
3. If you code an EXEC CICS HANDLE CONDITION TERMERR command in your application program, it is **sometimes** possible for the application program to handle exceptional cases, rather than using a node error program. The ‘TERMERR’ condition is driven if the node abnormal condition program (DFHZNAC) actions an ABTASK (‘ATNI’ abend). Note that ‘TERMERR’ is application-related, and is not an alternative to the node error program, which must be used for session-related problems. Dealing with errors in the application program is particularly useful in an intersystem communication (ISC) environment. For further information, refer to the *CICS/ESA Intercommunication Guide*.

Background to CICS-VTAM error handling

In general, errors detected by CICS-VTAM terminal control are queued for handling by a special task, the CICS node error handler (transid CSNE). (Note that CICS finds it convenient to use the same technique for some housekeeping work, such as sending “good morning” messages, and logging session starts and ends, which are not errors at all.)

In a few cases, exceptions signaled to CICS by VTAM are not treated as errors, and are not passed to the node error handler. For example, CICS often sends an SNA BID command as part of automatic transaction initiation. Rejection of the BID with exception code ‘0813’ (wait) is a standard response, and CICS handles the retry in terminal control without calling this an error. In the rest of this description, only the errors are considered.

The CSNE task runs as a “background” task, meaning that it is not associated with any one CICS terminal. At any time, there is at most one such task, working on the single node error queue.

All node errors on the queue are analyzed in turn by a table-driven, CICS-supplied program called DFHZNAC (node abnormal condition program). It is not intended that you should ever modify this.

DFHZNAC links to a module called DFHZNEP (if present in the CICS system) when processing most node errors. (It does **not** link to DFHZNEP for errors that are not related to a specific node—for example, those caused by a VTAM shutdown.) The interface for this link is described in “When an abnormal condition occurs” on page 403. This formal DFHZNAC–DFHZNEP interface gives you the opportunity to supply your own code to analyze error conditions, change default actions by setting various “action flags”, and take additional actions specific to your applications.

CICS supplies a pregenerated default DFHZNEP, which simply sets the “print TCTTE” action flag if a VTAM storage problem is detected, and returns control to DFHZNAC. Because it leaves all other action flags unchanged, DFHZNAC’s default actions are not otherwise affected. (DFHZNAC’s default actions for different error conditions are listed in Appendix B, “Default actions of the node abnormal condition program” on page 697.)

Why use a NEP to supplement CICS default actions?

The following list gives some of the reasons why you might want to write your own node error program to add to the default actions provided by CICS and VTAM.

- Not all errors represent communication system failures. Some errors (such as trying to write zero-length data) may reflect special situations in applications, needing special action.
- You might want to output extra data, in addition to the error messages sent by DFHZNAC. (Note that you cannot use the node error program to suppress messages from DFHZNAC.) All data output from DFHZNAC and DFHZNEP is written to the transient data queue CSNE.
- In other cases, you might want to change the amount of diagnostic information produced by CICS: the default varies with the error type. For example, the

VTAM RPL associated with an error may be printed when you do not want it, or not printed when you do.

- There could be application-related activity to be performed when a node error occurs. For example, if a message fails to be delivered to a terminal, it may need redirecting to another. With messages sent with exception-response only, CICS may not have the data available to send it again, but the requesting application might be able to re-create it. For example, if an error were signaled during the sending of a document to a printer, it might be able to restart from the beginning, or from a specific page.
- Some devices, such as the 3650 Retail Store System, return application-type data in “User Sense Data” fields. This can only be retrieved in a NEP. The NEP has to catch and save data for further application programs.

An overview of writing a NEP

Your DFHZNEP module must conform to the defined interface: that is, it must be a linked-to program that uses defined communication area fields to analyze an error and then returns to DFHZNAC. The source code of the default NEP provided by CICS can be used as a skeleton on which to build a single NEP.

CICS also provides macros to help you generate more complex sample NEPs. These are **aids** to help you develop your own NEPs; you do not have to use any of them.

Your error-handling logic can be written as a number of modules, but the one that receives control from DFHZNAC must be called DFHZNEP.

DFHZNEP code can use standard CICS functions (LINK, XCTL) to invoke other user modules. Each module thus requested must, of course, have an installed CSD program definition. Installed definitions are also needed for DFHZNAC and DFHZNEP themselves; these are contained in the supplied CSD group DFHVTAM.

The key features of the DFHZNAC–DFHZNEP interface are as follows:

- DFHZNEP can be written in any of the CICS-supported languages.

Note: CICS-supplied NEP code is provided in assembler language only. The communication area parameter list is supplied in assembler-language and C versions.
- DFHZNEP is linked-to separately for each node-related error on the queue. (Note that, because sense codes are always associated with an error, DFHZNEP is not linked-to separately for these.)
- Communication between the two modules is through a communication area (DFHNEPCA).

The structure of the communication area is described in “The communication area” on page 404.

On each DFHZNEP invocation, one field in the communication area contains a 1-byte internal error code, assigned by DFHZNAC, which identifies the type of error. Other fields identify the CICS TCTTE (LU) associated with the error, and any SNA sense codes. There are also fields for DFHZNEP to pass back user messages for subsequent logging by DFHZNAC.

background to VTAM error handling

Further fields contain “action flags”. Each flag represents an action that DFHZNAC may take when DFHZNEP returns control to it. These actions are of different types:

- Reporting (dumps of control blocks, actions taken)
- Status changes (for example, of TCTTE)
- Clean-up work (cancel any associated transaction, end the VTAM session).

The action flags can be set or reset within DFHZNEP.

The action flags set by DFHZNAC for specific error codes and sense codes are listed in Appendix B, “Default actions of the node abnormal condition program” on page 697.

The default NEP

The CICS-supplied default NEP, DFHZNEP, sets the “print TCTTE” action flag (TWAOTCTE in the user option byte TWAOPT1—see page 408) if a VTAM storage problem is detected; otherwise it performs no processing, leaves the action flags set by DFHZNAC unchanged, and returns control to DFHZNAC.

The sample NEP

The CICS sample node error program is a generalized program structure for handling errors detected from logical units. None of its components is generated as part of the standard CICS generation process, but instead may be optionally generated as described in this section and in “The sample node error program” on page 411.

The sample NEP that CICS provides is designed with two main features:

- It assumes that you want to invoke separate user-supplied error processors to handle different “groups” of error types. You specify which of the DFHZNAC internal error codes are to be regarded as a “group” for processing by any one routine, and then supply the code for that routine. CICS has some standard cases to help you. More information is given about them below.
- The supplied error processors may work in association with a separately generated module called a node error table. This can be used to build up statistics for each error group that the NEP processes. This table is analogous to the terminal error table, DFHTEPT, used by the equivalent CICS–TCAM sample terminal error program.

Some of the CICS-supplied error processors use the node error table—for example, that for errors affecting 3270 LUs (GROUP=1) (see “DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs” on page 416).

The node error table

To understand the sample NEP, first look at the node error table structure in more detail.

Node error table is often abbreviated to NET. You should not confuse this acronym with “net” (as in “network”), or with a NETNAME.

You can generate a node error table using the CICS macro DFHSNET. See “Node error table” on page 413 and “DFHSNET—generating the node error table” on page 418. You choose how complex this table is to be.

The node error table must be defined as a RESIDENT program. This makes it easy for the NEP to find it (using a CICS LOAD request), and ensures that any counters are not reset by reloading. You can give the table any name you like. The default is DFHNET.

The table consists of sets of error-recording areas. Each set is called a node error block (NEB) and is used to count node errors relating to a single LU. You can dedicate specific NEBs to specific LUs throughout a CICS run; and you can leave other, reusable NEBs for general use. If you expect to accumulate error statistics about 10 LUs concurrently, you need 10–12 NEBs.

Each NEB may contain multiple recording areas, one being used for each group of errors you want to distinguish. The error groups correspond to those in the NEP. That is, they are groups of error types requiring separate processing logic.

Each recording area is known as an error status block (ESB). You specify the space reserved for each ESB, and it typically includes space to count the errors, or record when the first of the present series occurred. Note that in any one NEB the counting is for one LU only.

Finally, you can specify a threshold count and a time limit in the table. These are constants that can be used by code in the NEP to test an ESB, to see if a given type of error has occurred more than the threshold number of times in the stated interval. The time limit also affects the interval between using a general NEB for one LU and then reusing it for another.

A minimal NET would simply consist of a handful of NEBs, each with just one ESB, grouping together all types of error that are of interest.

Coding the sample NEP

The sample NEP is coded using the macro DFHSNEP. The basic form is as follows:

```
DFHSNEP TYPE=INITIAL
```

Specific error handling code. For example:

```
DFHSNEP TYPE=DEF3270
```

```
DFHSNEP TYPE=FINAL  
END      DFHNEPNA
```

By default, this generates a module called DFHZNEP, which works with a node error table called DFHNET. If you want to use another table, you could code NETNAME=MYTABLE after TYPE=INITIAL. Details of the DFHSNEP macro are given in “Generating the sample node error program” on page 414.

To understand the sample code, generate a standard NEP, as with TYPE=DEF3270, shown in “DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs” on page 416, and look at the resulting assembler-language listing. Here is a description of the code.

The INITIAL and FINAL macros generate the basic skeleton of the NEP. This comprises some initialization code and some common routines. All the code is built round the assumption that you have a node error table as previously described.

background to VTAM error handling

The initial code first tests the internal error code passed from DFHZNAC to see if it belongs to a group that the NEP needs to handle. (The groups are identified by the code you supply between the DFHSNEP INITIAL and FINAL macros. This is described in “Generating the sample node error program” on page 414.) If the particular error code is not of interest to the NEP, control is returned at once to DFHZNAC, to take default actions.

Otherwise, the relevant node error table is located by a CICS LOAD request. (As previously explained, this table should be resident in virtual storage.) The NEP code will then locate the correct ESB within a selected NEB. The latter may be permanently dedicated to the LU in error (a named NEB), or may be one taken from the general pool.

The initial code then invokes the appropriate user logic for that error group. The initial code also sets up pointers to the communication area, the NEB, and the ESB. For details, see “Generating the sample node error program” on page 414.

The common routines in the NEP provide common services for your own logic. They count and time stamp errors in the ESB, and test whether error thresholds have been exceeded. They are not documented outside the sample listings. You can generate a NEP without them if you prefer.

Your own code is inserted between the DFHSNEP TYPE=INITIAL and TYPE=FINAL macros.

Note: If the user code you insert between the DFHSNEP macros contains EXEC
CICS commands, you must translate the commands, and enter the *translated code*
between the DFHSNEP macros.

Each section of user logic, intended to handle a particular group of error types, is headed by a macro of the type:

```
DFHSNEP TYPE=ERRPROC, CODE=(ab,cd,...), GROUP=n
```

where X'ab', X'cd',... are the DFHZNAC internal error codes you want to process, and n is the number of the error group, and therefore also of the corresponding ESB, within a NEB, in the node error table. Successive DFHSNEP TYPE=ERRPROC macros should use groups 1, 2, 3, and so on.

The DFHSNEP TYPE=ERRPROC macros serve several purposes. They:

- Inform the NEP generation how many error groups there are
- Show which error types are to be included in each group
- Introduce the code for each group.

Note that any one DFHZNAC error code should only figure in one error group, and that any code not mentioned is simply ignored by the NEP. You follow each DFHSNEP TYPE=ERRPROC macro with your own logic. This should begin with standard code to save registers, or set up addressability, which is best copied from sample NEP listings.

CICS provides some standard error processors to handle specific errors on two different types of LU. These are for non-SNA 3270s (BSC 3270s attached to CICS-VTAM), and for interactive SNA logical units like a 3767. More information is given in “When an abnormal condition occurs” on page 403.

The code for non-SNA 3270s can be generated by coding

```
DFHSNEP TYPE=DEF3270
```

where you would otherwise code a DFHSNEP TYPE=ERRPROC macro plus logic of your own. In effect, TYPE=DEF3270 defines two error groups, and associates each with an error processor. The first group comprises the four DFHZNAC error codes X'D9', X'DC', X'DD', and X'F2'. The second group contains only error code X'42', corresponding to the 'unavailable printer' condition, a specific exception condition signaled when CICS cannot allocate a printer in response to a 3270 print request.

The 3270 sample code is not intended to cover all error conditions. Note that the code is not suitable for SNA 3270s (LU session type 2). Error conditions arising from these result in different DFHZNAC error codes and may require different handling.

You may find that the CICS-supplied code is not sufficient for other, application-related, reasons. Perhaps you want to try to reacquire lost sessions after a time interval. The code supplied for the 3767 covers only one error group with one DFHZNAC error code, X'DC', which may occur under contention protocol.

You can use these CICS-supplied error processors to generate a valid DFHZNEP listing, for tutorial purposes, without having to write any user code.

You should be aware of the following limitations of this NEP design:

- Any error types you have not allowed for are ignored by the NEP, and not accumulated into error buckets.
- You may want to handle a particular situation whenever it arises, even though DFHZNAC may assign it different error codes in different situations. For example, on an SNA 3270, switching in and out of TEST state generates status X'082B' (presentation-space integrity lost). This might result in one of several DFHZNAC error codes.

In the sample NEP structure, you would need either to test for this last case in separate error processors, or group all the DFHZNAC error codes together. If you wrote your own NEP code from scratch, you would simply, on entry to your NEP, test the communication area field containing the status.

Multiple NEPs

CICS allows you to define a NEP transaction class that applies to every transaction that uses a particular profile, session, or terminal-type. To do this you use the NEPCCLASS option of the CEDA DEFINE PROFILE, CEDA DEFINE SESSIONS, or CEDA DEFINE TYPETERM command. (Note that any value of NEPCCLASS that you specify using CEDA DEFINE PROFILE overrides any specified using CEDA DEFINE SESSIONS or CEDA DEFINE TYPETERM.) NEPCCLASS is a 1-byte binary field containing a value in the range 0–255. The purpose of NEPCCLASS is that, while a transaction is running on the LU, you can obtain a special version of node error handling, suitable for that transaction. (This is sometimes called a "transaction-class error routine".) The default value NEPCCLASS(0) indicates that no NEPCCLASS is in effect.

background to VTAM error handling

The DFHZNEP that gets control from DFHZNAC must test the NEPCCLASS in effect at that time for the LU associated with the error. Then it either transfers control to a suitable module (the actual NEP), or branches to a specific bit of code within itself.

The DFHZNEPI macros (see “DFHZNEPI macros” on page 424) generate a DFHZNEP module that is purely a routing module. This inspects the NEPCCLASS in effect for the node error passed by DFHZNAC, and transfers control (links) to another module, the real NEP, according to a NEPCCLASS/name routing table built up by the macros.

If no NEPCCLASS is in effect (equivalent to CEDA DEFINE PROFILE NEPCCLASS(0)), or the NEPCCLASS is not in the routing table, a default module is invoked. You must specify the name of this in the DFHZNEPI TYPE=INITIAL macro. (See “DFHZNEPI TYPE=INITIAL—specifying the default routine” on page 425.) If you do not specify the name, no module is invoked.

You also have to provide the sub-NEPs for the various NEP transaction classes, including, of course, one for the default NEPCCLASS(0). Each of these sub-NEPs needs a separate program definition. You have the same choice in coding each sub-NEP as you had when there was just one; you can code your own, or use the CICS sample macro DFHSNEP. If you use DFHSNEP, note that there is another operand on the DFHSNEP TYPE=INITIAL macro, NAME=, which means that the generated module can be given any name you choose (to match the DFHZNEPI routing). You can use a different node error table with each sub-NEP.

Before you start using NEP routing, consider the following:

- The association of an LU (TCTTE) with a transaction NEPCCLASS is only valid for about the time that the CICS task exists. Errors detected after a CICS task has ended (for example, because of a problem with a delayed output message) may not be associated with the NEPCCLASS of the creating transaction.

Another problem can occur when CICS is about to start a new task for the LU as a result of an internal request from another CICS task (by an EXEC CICS START request, for example). This is usually called automatic transaction initiation. Before the task is started, CICS has to open a fresh session if none exists, by issuing a VTAM SIMLOGON request, and then, as mentioned earlier, send a BID command. The intended task is not attached until all this is completed successfully. The NEPCCLASS is not picked up from the transaction definition until then. This means that any errors arising in the ATI process (perhaps an error on BIND or BID) occur before the NEPCCLASS is correctly set, so they may get routed to the default NEP and not the one for the NEPCCLASS. This complicates the total node error handling for the application.

As an example, consider an application that contacts unattended programmable controllers overnight in order to read in the day's input. Recovery design in such an application is fundamental, and has to allow for errors both in ATI and in file transmission. To separate these into two NEPs could be an unnecessary complication.

- The extra development effort for a NEP split on a NEPCCLASS basis might not be justified. Generally, if logic is to be split, it is on an LU basis (programmable controllers may be running applications other than 3270).

To conclude this overview, remember that the CICS sample NEPs are a good source of ideas for you to write your own NEPs, but they might not be the ideal framework for your particular needs. It is recommended that you to write straightforward NEPs at first.

When an abnormal condition occurs

The following CICS components are involved when an abnormal condition is detected from a logical unit:

- The terminal control program VTAM section: DFHZCA, DFHZCB, DFHZCC, DFHZCP, DFHZCQ, DFHZCW, DFHZCX, DFHZCY, and DFHZCZ.
- The node abnormal condition program, DFHZNAC.
- The CICS-supplied default node error program, DFHZNEP, or your own version of it.

For logical units, all information concerning the processing state of the terminal is contained in the TCTTE and the request parameter list (RPL). Consequently, when a terminal error must be handled for a logical unit, the TCTTE itself is placed onto the system error queue.

DFHZNAC assumes that system sense codes are available upon receipt of an exception response from the logical unit. Thus, analysis is performed to determine the reason for the response. Decisions, such as which action flags to set and which requests are needed, are made based upon the system sense codes received. If sense information is not available, default action flags are set, and DFHZEMW is scheduled to send a negative response, if a response is outstanding, with an error message to the terminal.

The action flags set by DFHZNAC on receipt of specific inbound system sense codes are listed in Appendix B, “Default actions of the node abnormal condition program” on page 697.

Before executing the specified routines, DFHZNAC links to DFHZNEP. You can use DFHZNEP to perform additional error processing beyond that performed by DFHZNAC; or to alter the default actions previously set by DFHZNAC. You need to code a node error program only if you want to do either of these things.

The action flags, set by DFHZNAC to assist the node error program, are in field TWAOPTL of the communication area.

If you want to modify DFHZNAC’s actions following an abnormal situation, DFHZNEP can interrogate field TWAOPTL and modify the bit settings. If you agree with DFHZNAC’s proposed actions, field TWAOPTL is left unaltered.

In most cases, DFHZNEP can modify DFHZNAC’s proposed actions. The only time that DFHZNAC overrides DFHZNEP’s modification of field TWAOPTL is when a logical unit is to be disconnected from CICS; that is, when DFHZNAC determines that the abnormal situation requires that CICS issue the ACF/VTAM CLSDST macro for a logical unit. In such a case, DFHZNAC disconnects the terminal and abnormally terminates the task, even if DFHZNEP tries to block such actions.

when an abnormal condition occurs

Resetting of the task termination flag by the node error program is also ignored if a negative response has been sent to a logical unit, or if DFHZEMW is to write an error message to the logical unit.

When the node error program has performed its functions, it returns control to DFHZNAC by an EXEC CICS RETURN command.

When control is returned from DFHZNEP, DFHZNAC performs the actions specified in field TWAOPTL (except when disconnecting logical units, as noted above), issuing messages and setting error codes, as necessary.

The communication area

After DFHZNEP receives control from DFHZNAC, it obtains the address of the communication area by means of an EXEC CICS ADDRESS COMMAREA command. Figure 24 illustrates the general structure of the communication area.

Header
Error_being_processed
User option bytes
VTAM information
Additional information for NEP
Additional system parameters
XRF parameters

Figure 24. General structure of the communication area

The significance of each section of the communication area is described below:

Header	A 4-byte header common to all user-replaceable programs.
Error_being_processed	Identifiers of the error code and the terminal associated with the error.
User option bytes	Flags that indicate the default actions set by DFHZNAC, and that may be reset within DFHZNEP.
VTAM information	Sense and RPL codes.
Additional info. for NEP	Other useful information for the NEP.
Additional system parameters	Locations of indirect parameters, such as the TCTTE, and other system information.
XRF parameters	Recovery notification data.

A detailed listing of the communication area is given in Figure 25.

```

*****
**                               Header                               **
**                               These fields are READ ONLY          **
*****
NEPCAHDR DS   0XL4                Standard Header
NEPCAFNC DS   XL1                 Function Code   Always '1'
NEPCACMP DS   XL2                 Component Code  Always 'ZC'
           DS   XL1                 Reserved
*****
**                               Error_being_processed             **
** Identity of terminal and the error code associated with it      **
**                               These fields are READ ONLY          **
*****
TWAEC   DS   XL1                Error Code
           DS   CL3                Reserved
TWANID  DS   CL4                Terminal identity
TWANETN DS   CL8                Netname
*****
**                               User option bytes                 **
**                               Initially set to the default actions. **
**                               DFHZNEP can change the defaults.     **
*****
TWAOPTL DS   0XL3                User option bytes
TWAOPT1 DS   XL1                 User option byte 1
TWAOPT2 DS   XL1                 User option byte 2
TWAOPT3 DS   XL1                 User option byte 3
           DS   XL1                 Reserved

```

Figure 25 (Part 1 of 3). The DFHZNAC/DFHZNEP communication area

when an abnormal condition occurs

```

*****
**      VTAM information - Any VTAM sense and RPL codes      **
**      These fields are READ ONLY                          **
*****
TWAVTAM DS    0XL12          VTAM information
TWARPLCD DS    H            VTAM RPL feedback codes
          DS    H            Reserved
TWASENSS DS    0F          Sense codes to be sent
TWASS1 DS    XL1          System sense byte No 1
TWASS2 DS    XL1          System sense byte No 2
TWAUS1 DS    XL1          User sense byte No 1
TWAUS2 DS    XL1          User sense byte No 2
*
TWASENSR DS    0F          Sense codes received
TWASR1 DS    X            System sense byte No 1
TWASR2 DS    X            System sense byte No 2
TWAUR1 DS    X            User sense byte No 1
TWAUR2 DS    X            User sense byte No 2
*
*****
**      Additional information for the NEP                    **
**Except for TWANPFW, TWANLD, and TWANLDL these fields are READ ONLY**
*****
TWAADINF DS    0XL22
          DS    F            Reserved
TWACTLB DS    X            General use control byte
* EQU    X'80'          Reserved
* EQU    X'40'          Reserved
TWACSC EQU    X'20'          Clear sense code indicator
TWAPSC EQU    X'10'          Print VTAM sense codes
TWATIOA EQU    X'08'          Print portion of I/O area
* EQU    X'04'          Reserved
TWAVTRTC EQU    X'02'          VTAM return code available
TWANEPR DS    XL1          NEP return code byte
TWANPFW EQU    X'80'          Retry write with FORCE=YES
TWAREASN DS    XL1          VTAM reason code
TWASTAT DS    XL1          VTAM status code
TWAXRSN DS    H            Exception response seq number recd
TWAR EQU    *
TWAPFLG DS    XL1          CLSDST pass flag
TWAPIP EQU    X'80'          CLSDST pass in progress
TWANEPC DS    XL1          NEP class flag
TWAEISAB DS    XL1          Stand-alone begin bracket indicator
TWAESAB EQU    X'04'          Stand-alone begin bracket
          DS    XL3          Reserved
TWANLD DS    A            Address of data to be logged
TWANLDL DS    H            Length of data to be logged

```

Figure 25 (Part 2 of 3). The DFHZNAC/DFHZNEP communication area

```

*****
**                               Additional system parameters                               **
**Except for TWAPNETN, TWAPNTID, TWAUPRRRC these fields are READ ONLY**
*****
TWASYSM DS      0XL68
TWATCTA DS      AL4                               Address of TCTTE being processed
TWARPL  DS      AL4                               Address of VTAM RPL
TWATIOAA DS     AL4                               Address of data portion of TIOA
TWATIOAL DS     H                                 Length of data portion of TIOA
TWACOMML DS     H                                 Length of commarea data for TCTTE
TWACOMMA DS     CL4                              Address of commarea data for TCTTE
TWATECIA DS     AL4                              Address of TCTTE user area
TWATECIL DS     H                                 Length of TCTTE user area
TWAPPNTN DS     CL8                              Primary 3270 printer netname
TWAPPTID DS     CL4                              Primary 3270 printer termid
TWAPPELG DS     X                                 Primary printer eligible indicator
TWAPPELY EQU    X'01'                            Primary printer is eligible flag
TWASPNTN DS     CL8                              Secondary 3270 printer netname
TWASPTID DS     CL4                              Secondary 3270 printer termid
TWASPELG DS     X                                 Secondary printer eligible indicator
TWASPELY EQU    X'01'                            Secondary printer is eligible flag
TWAPNETN DS     CL8                              Selected 3270 printer netname
TWAPNTID DS     CL4                              Selected 3270 printer termid
TWAUPRRC DS     B                                 Unavailable Printer return code
TWAUPRNP EQU    X'00'                            No printer selected
TWAUPRPS EQU    X'01'                            Printer selected
TWAUPRDD EQU    X'FF'                            Data disposal complete
TWAUPRPE EQU    X'FE'                            Error on Put request
TWAERRF1 DS     B                                 Error flag byte 1
TVALXS  EQU     X'80'                            Logon crossed simlogon
          DS     XL2                              Reserved

*****
**                               XRF parameters                               **
**                               XRF recovery notification data                       **
**                               DFHZNEP can change these default actions           **
*****
TWAXRNOT DS     X                                 Recovery notification options
TWAXRNON EQU    X'80'                            Recov notification = none
TWAXRMSG EQU    X'40'                            Recov notification = message
TWAXRTRN EQU    X'20'                            Recov notification = transact.
          DS     XL3                              Reserved
TWAXMSTN DS     CL8                              Recovery mapset name
TWAXMAPN DS     CL8                              Recovery map name
TWAXTRAN DS     CL4                              Recovery transaction ID
*

```

Figure 25 (Part 3 of 3). The DFHZNAC/DFHZNEP communication area

The next sections describe fields in the parameter list that can be reset within DFHZNEP. See also “Coding for the 3270 ‘unavailable printer’ condition” on page 422, which describes the use of the flags in the “unavailable printer return code” field, and “Using the node error program with XRF or persistent sessions” on page 427, which describes how the flags in the XRF part of the parameter list can be manipulated.

The user option bytes (TWAOPTL)

TWAOPTL contains the user option bytes TWAOPT1, TWAOPT2, and TWAOPT3, each of which contains action flags. On entry to DFHZNEP, these flags represent the default actions previously set by DFHZNAC. They can be reset by DFHZNEP.

TWAOPT1

User option byte 1. TWAOPT1 contains flags which are principally debugging aids. The first five flags cause DFHZNAC to write the desired information to the CSNE log if the appropriate bit is set. Setting the sixth flag (TWAODNTA) on causes CICS to take a system dump when there is no task attached to the terminal at the time of error detection, if the flag TWAOAT in TWAOPT2 is also set on.

The flags are:

TWAOAF (X'80')	Print action flags
TWAORPL (X'40')	Print VTAM RPL
TWAOTCTE (X'20')	Print TCTTE
TWAOTIOA (X'10')	Print TIOA
TWAOBIND (X'08')	Print BIND area
TWAODNTA (X'04')	System dump if no task attached.

TWAOPT2

User option byte 2. TWAOPT2 contains flags which are task-related.

The NEP can abend the task by setting TWAOAT, or cancel it by setting TWAOCT. The difference is that abend task does not take effect until the task requests or completes a terminal control operation: cancel task takes effect as soon as system and data integrity can be maintained. Setting TWAOAT to abend the task is normally sufficient, except where the task performs lengthy processing (such as a database browse) between terminal requests. If both TWAOAT and TWAOCT are set, TWAOCT (cancel task) takes priority.

If the task is to be abnormally terminated, sends and receives are purged. If TWAOGMM is set, the next transid is cleared and any communication area associated with the terminal is released—except in the case of permanent transids (specified on the TERMINAL definition as TRANSACTION(name)), when the communication area is not released. If the TYPETERM of the terminal indicates that the “good morning” message is supported (LOGONMSG(YES)), if TWAONINT is off, and if the terminal is not in a BMS paging session, then the “good morning” message transaction is initiated (the transaction specified by the system initialization parameter GMTRAN).

The flags are:

TWAOAS (X'80')	Abandon any SEND for this terminal
TWAOAR (X'40')	Abandon any RECEIVE for this terminal
TWAOAT (X'20')	Abend any task attached to TCTTE
TWAOCT (X'10')	Cancel any task attached to TCTTE
TWAOGMM (X'08')	“good morning” message to be sent
TWAOPBP (X'04')	Purge any BMS pages for this session
TWAOASM (X'02')	SIMLOGON required.

Notes:

1. If a definite response SEND has been performed, CICS has to issue a RECEIVE in order to obtain the response. If the response is negative, DFHZNAC is entered and sets flags TWAOAS (abandon the SEND) and TWAOAR (abandon the RECEIVE). TWAOAR must be left on to ensure that the RECEIVE for the response is abandoned.
2. If the request is to be retried, and the break connection action flag is off (that is, if TWAOCN in TWAOPT3 is off), then one or more of TWAOAS, TWAOAR, and TWAONEGR must be off as well as TWAOAT.
3. The abend code returned as a result of setting TWAOCT is unpredictable.
4. TWAOGMM forces TWAOAT only if set on by the node error program.
5. TWAOPBP forces TWAOAT to be set on.
6. For non-pipeline terminals, TWAOAT acts as a cancel request (TWAOCT) if the task has not yet been dispatched for the first time.

TWAOPT3

User option byte 3. TWAOPT3 contains flags which are node-related.

The flags are:

TWAOINT (X'80')	Internally generated logons (INTLOGs) allowed
TWAONINT (X'40')	No internally generated logons allowed ⁵
TWAONCN (X'10')	Normal CLSDST (no reset allowed)
TWAOSCN (X'08')	Normal CLSDST (reset allowed)
TWAONEGR (X'04')	Send negative response
TWAOOS (X'02')	Keep node out of service
TWAOCN (X'01')	CLSDST node. ⁵

TWAONINT forces TWAOCN.

TWAONEGR forces TWAOAR and TWAOAT.

TWAOOS forces TWAOCN.

TWAOCN forces TWAOAR, TWAOAS, and TWAOAT.

TWAOOS indicates that no further processing is to be done for this node. The node is logically out of service.

For an LU6.1 intersystem communication session, TWAOOS or TWAONINT causes the system entry to be put out of service if, as a result of the specified action, there are no allocatable sessions left. (A session can also be put out of service because of either an unknown modename being passed to VTAM during an attempt to bind an APPC session, or an invalid logmode name for a VTAM 3270-type terminal. However, the CICS default action resulting from this condition cannot be overridden in the NEP.)

If TWAOCN is set, the task is abnormally terminated and communication with the node is lost. Note that the NEP cannot reset this flag.

TWAOSCN provides the same function as TWAONCN, but the NEP can reset it if the session is not to be closed.

⁵ Do not set this flag when processing error code X'49' (TCZCLSIN).

when an abnormal condition occurs

If DFHZNAC is scheduled because of the receipt of an exception response, the sense information in the TCTTE is available to DFHZNAC and DFHZNAP to determine any necessary actions.

If DFHZNAC is scheduled because of loss of the connection between CICS and a logical unit, DFHZNAC abnormally terminates any transaction in progress at the time of the failure. DFHZNAP and transaction-class error routine analysis and processing are permitted, but you should not attempt to retry the message.

However, if the application program handles the 'TERMERR' condition, the transaction is not abended. Control is returned to the program. In this circumstance, no further use can be made of the failed session.

Additional information for the NEP (TWAADINF)

Fields TWANPFW, TWANLD, and TWANLDL can be reset by the NEP. For information about the use of TWANPFW, see the supplied sample node error program, and "Optional error processor for interactive logical units" on page 414.

TWANLD and TWANLDL — using the DFHZNAC logging facility: You can use the logging facility available in DFHZNAC to aid in retrieving information. You specify the address of the data that you want to examine in field TWANLD of the communication area, and the length of the data in field TWANLDL. The data is logged to the CSNE transient data queue for future inspection.

Note: No data in excess of 220 bytes is logged.

You can also send user-written messages to the CSNE log using the transient data facility. To write your messages, you must code the EXEC CICS WRITEQ TD instruction directly into the node error program.

TWAPIP — and application routing failure: The EXEC CICS ISSUE PASS command passes control from CICS to another named VTAM application. For programming information about the EXEC CICS ISSUE PASS command, see the *CICS/ESA Application Programming Reference* manual. The ISSUE PASS command in turn invokes the VTAM macro CLSDST with OPTCD=PASS, and, in addition, if NOTIFY has been specified on the CLSDSTP system initialization parameter, with PARMS=(THRDPTY=NOTIFY). CICS is then notified of the outcome of any CLSDST request.

This notification results in an informative message being issued, and causes DFHZNAC to invoke your NEP, whether the CLSDST request has failed or succeeded. The NEP can discover that a CLSDST OPTCD=PASS request is in progress by examining field TWAPFLG for the pass-in-progress indicator, TWAPIP. The success or failure of the CLSDST OPTCD=PASS request can be determined by examining the error code at TWAEC.

If the pass operation fails, DFHZNAC sets up a default set of recovery actions that can be modified by your NEP. A possible recovery, when, for example, the target application program is not active, would be to reestablish the session with the initial application using a SIMLOGON request and for CICS to send its "good morning" message to the terminal. The default action is to leave the session disconnected and to make it NOCREATE.

If CLSDSTP=NONOTIFY has been specified, and autoinstall is being used, CICS takes no action, even if the ISSUE PASS fails.

If persistent sessions support is active, autoinstall terminals are deleted after the AIRDELAY, so any expected NEP processing as a result of CLSDSTP=NOTIFY being coded does not take place.

The additional system parameters (TWASYSPM)

If a data element referenced in this section of the parameter list (for example, the TIOA) does not exist when the NEP is driven, its address and length fields are set to zero.

Fields TWAPNETN, TWAPNTID, and TWAUPRRRC can be reset by the NEP. The use of these fields is discussed in “Coding for the 3270 ‘unavailable printer’ condition” on page 422.

XRF parameters (TWAXRNOT)

These fields can be reset by the NEP. See “Using the node error program with XRF or persistent sessions” on page 427.

The sample node error program

The sample node error program provides a general environment for the execution of error processing routines (error processors), each of which is specific to certain error codes generated by the node abnormal condition program. Sufficient optional error processors for normal operation of VTAM 3270 or interactive logical unit networks are provided; these can be easily supplemented or replaced by user-supplied error processors.

There are three types of error that may occur in a VTAM network:

- Errors in the host system
- Communication errors, such as session failures
- Abnormal conditions at the terminal, such as intervention required and invalid requests.

A sample node error program is supplied with CICS, and can be used as the basis of each subsequent node error program that you write. This provides you with:

- A general environment within which your error processing programs can be added
- The default node error program in a system that has several node error programs.

The CICS-supplied sample node error program is described in greater detail below.

Compatibility with the sample terminal error program

Receipt of sense or status codes corresponds to error codes X'D9', X'DC', X'DD', and X'F2'. Weighted counts of these messages are maintained against numeric and time thresholds. If the numeric threshold is exceeded, default actions are taken. If the time threshold is reached, the count is reset. This is equivalent to the function in the sample TEP, except that sense or status arising out of the “from” device on a COPY command is now presented to the node error program as an error on the “to” device, thus exceeding the threshold, which causes the request to be terminated, although the terminal remains in service. Some of the weights for errors that occur on the 3270 display device have been revised, but otherwise the

the sample node error program

weight and threshold values are the same as the defaults used in the sample TEP. Time threshold maintenance for the sample NEP is mandatory, and not optional as in the sample TEP.

For further information about time and threshold count limits, see the information about the sample terminal error program in Chapter 8, "Writing a terminal error program" on page 361.

The 3270 message 'unavailable printer' corresponds to error code X'42' (interval control PUT request has failed). The algorithm used for printer selection differs in VTAM support. The retry algorithm in the sample node error program is similar to this new selection algorithm.

Components of the sample node error program

The sample node error program comprises the following components:

- An entry section.
- The routing mechanism.
- The node error table.
- Optional common subroutines.
- Optional error processors for 3270 or interactive logical units. A node error program cannot be generated with both 3270 and interactive logical unit error processors.

The components are described in the sections that follow.

Entry section

On entry, the sample NEP uses DFHEIENT to establish base registers and addressability to the EXEC interface. It uses an EXEC CICS LOAD PROGRAM command to establish addressability to the node error table (NET) and, if included, the common subroutine vector table (CSVV). It uses an EXEC CICS ADDRESS COMMAREA command to obtain addressability to the communication area passed by DFHZNAC, and an EXEC CICS ADDRESS EIB command to obtain addressability to the EXEC interface block. If time support has been generated, the error is time-stamped for subsequent processing.

Routing mechanism

The routing mechanism invokes the appropriate error processor depending on the error code provided by the node abnormal condition program.

Groups of one or more error codes are defined in the DFHSNEP macro (see below). Each group is associated with an index (in the range X'01' through X'FF') and an error processor. A translate table is generated and the group index is placed at the appropriate offset for each error code. Error codes not defined in groups have a zero value in the table. An error processor vector table (EPVT) contains the addresses of the error group processors, positioned according to their indexes. The vector table extends up to the maximum index defined; undefined intermediate values are represented by zero addresses.

The error code is translated to obtain the error group index. A zero value causes the node error program to take no further action; otherwise the index is used to obtain the address of the appropriate error processor from the EPVT. A zero address causes the node error program to take no further action; otherwise a call is made to the error processor. This is entered with direct addressability to the NET

and CSVT areas. When the error processor has been executed, the node error program returns control to the node abnormal condition program.

Node error table

The node error program may use a node error table (NET) that comprises the node error blocks (NEBs) used to maintain error status information for individual nodes (see Figure 26). Some or all of the NEBs can be permanently reserved for specific nodes; others are dynamically assigned to nodes when errors occur. Dynamically assigned NEBs are used exclusively for the nodes to which they are assigned until they are explicitly released. All the NEBs have an identical structure of error status blocks (ESBs). Each ESB is reserved for one error processor and associated with it by means of the appropriate error group index. The ESB length and format can be customized to the particular error processor that it serves.

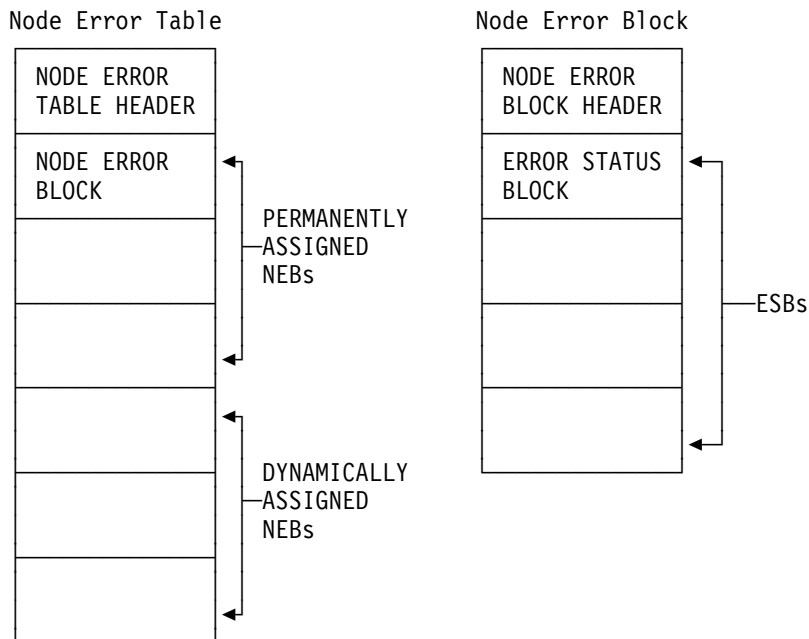


Figure 26. Format of node error table and node error block

Optional common subroutines

The common subroutines are addressed via the CSVT and provide error processors with the following functions:

- Locate or assign NEBs and ESBs on the basis of node ID and error group index.
- Time stamp an error, update an error count, and test an error count against numeric and time threshold values.
- Release a dynamically assigned NEB from a particular node.

Optional error processors for 3270 logical units

Two error processors are supplied for 3270 LUs, as follows:

1. Group index 1, error codes X'D9', X'DC', X'DD', and X'F2'.

These error codes correspond to the receipt of sense or status bytes in the user sense fields of the RPL. The error processor locates an ESB of the

the sample node error program

standard format and updates a weighted error count. The weight, threshold, and timer values are based on those used by the sample terminal error program 3270 except as noted in the previous section. If the threshold is not exceeded, the abend SEND, abend RECEIVE, abend transaction flags, and all the print action flags are turned off. Otherwise the default actions are taken and the NEB is released if it is reusable.

2. Group index 2, error code X'42'.

This code means that no 3270 printer was available to satisfy a print request made at a 3270 screen. The error processor examines the printers defined for this screen to determine why they were unavailable. If either is busy on a previous PRINT or COPY request (that is, a task is attached with a transaction ID of CSPP or CSCY) or is no longer unavailable, that printer address is returned to the node abnormal condition program which retries the print request with an IC PUT command. Otherwise the default actions are taken. (For more details, see the section "Coding for the 3270 'unavailable printer' condition" on page 422.)

Optional error processor for interactive logical units

Only one error processor is supplied for interactive LUs: group index 1, with error code X'DC'.

This error code, in combination with a user sense value of X'081B', indicates a 'receiver in transmit mode' condition. The action flags in TWANPFW are manipulated to allow the failing SEND request to be retried.

Generating the sample node error program

The routing mechanism, common subroutines, CICS-supplied error processors, and user-supplied error processors are generated by means of DFHSNEP macros.

The sample node error program and table need to be translated, assembled, and link-edited. For information about the job control statements required to assemble and link-edit user-replaceable programs, refer to "Assembling and link-editing user-replaceable programs" on page 344.

Note that you should code the translator options NOPROLOG and NOEPILOG in your node error program.

Note also that an extra 24 bytes are required for the common subroutines register save area, and further space is required for the error processor save area. The CICS sample processors use 4 bytes of this area.

The DFHSNEP macro to generate the sample node error program has seven types, as follows:

TYPE=USTOR

to indicate the start of user storage definitions.

TYPE=USTOREND

to indicate the end of user storage definitions.

TYPE=INITIAL

to generate the routing mechanism and, optionally, the common subroutines.

TYPE=DEF3270

to generate the default CICS-supplied error processors for 3270 devices.

TYPE=DEFILU

to generate the default CICS-supplied error processor for interactive logical units operating in contention mode.

TYPE=ERRPROC

to indicate the start of a user-supplied error processor.

TYPE=FINAL

to indicate the end of the sample node error program.

DFHSNEP TYPE=USTOR and USTOREND—defining user storage

The DFHSNEP TYPE=USTOR macro has the following format:

```
DFHSNEP TYPE=USTOR
```

This macro indicates the start of user storage definitions. It must be followed by your storage definitions, and then by DFHSNEP TYPE=USTOREND. If you use DFHSNEP TYPE=USTOR to define storage, then both it and DFHSNEP TYPE=USTOREND must be coded **before** DFHSNEP TYPE=INITIAL.

The DFHSNEP TYPE=USTOREND macro has the following format:

```
DFHSNEP TYPE=USTOREND
```

This macro indicates the end of user storage definitions. Its use is mandatory if DFHSNEP TYPE=USTOR has been coded. If you use DFHSNEP TYPE=USTOR to define storage, then both it and DFHSNEP TYPE=USTOREND must be coded **before** DFHSNEP TYPE=INITIAL.

DFHSNEP TYPE=INITIAL—generating the routing mechanism

One DFHSNEP TYPE=INITIAL macro must appear immediately **after** DFHSNEP TYPE=USTOR and DFHSNEP TYPE=USTOREND (if they are coded) and **before** the remaining macros.

```
DFHSNEP TYPE=INITIAL
      [,CS=NO]
      [,NAME=name]
      [,NETNAME=netname]
```

TYPE=INITIAL

indicates the start of the sample node error program and causes the routing mechanism to be generated.

CS=NO

specifies that the generation of the common subroutines is to be suppressed.

NAME=name

specifies the name of the node error program module identifier. The name must be a string of 1 through 8 characters. This operand is optional, and the default is DFHZNEP0. If you allow the NAME operand to default, you can use the examples on page 346 to create link-edit statements, but if you specify a different NAME, you must change the link-edit statements accordingly. If the interface module DFHZNEP (generated by the DFHZNEPI macro) is used, this operand must be specified (with a name other than DFHZNEP).

NETNAME=netname

specifies the name of the node error table to be loaded at initialization. The name must be a string of 1 through 8 characters. This operand is optional, and the default is DFHNET.

DFHSNEP TYPE=DEF3270—including error processors for 3270 LUs

The DFHSNEP TYPE=DEF3270 macro has the following format:

```
DFHSNEP TYPE=DEF3270
```

TYPE=DEF3270

specifies that the CICS-supplied error processors for 3270 logical units are to be included in the node error program. This macro causes the following source code to be generated:

```
DFHSNEP TYPE=ERRPROC,GROUP=1,CODE=(D9,DC,DD,F2)  
Sense/status error processor code.
```

```
DFHSNEP TYPE=ERRPROC,GROUP=2,CODE=42  
Unavailable printer error processor code.
```

DFHSNEP TYPE=DEFILU—including error processors for INTLUs

The DFHSNEP TYPE=DEFILU macro has the following format:

```
DFHSNEP TYPE=DEFILU
```

TYPE=DEFILU

specifies that the CICS-supplied error processor for interactive logical units is to be included in the node error program. This macro causes the following source code to be generated:

```
DFHSNEP TYPE=ERRPROC,GROUP=1,CODE=DC  
(receiver in transmit mode error processor code)
```

DFHSNEP TYPE=FINAL—terminating DFHSNEP entries

One DFHSNEP TYPE=FINAL macro must follow all the other DFHSNEP macros. It has the following format:

```
DFHSNEP TYPE=FINAL
```


TYPE=FINAL

indicates the end of the node error program and causes the error processor vector table (EPVT) to be generated.

DFHSNEP TYPE=ERRPROC—specifying a user error processor

The DFHSNEP TYPE=ERRPROC macro is used to indicate the start of a user-supplied error processor. The actual error processor code should immediately follow this macro. The assembly should be terminated by the statement: END DFHNEPNA.

The following operands can be used on the DFHSNEP TYPE=ERRPROC macro:

```
DFHSNEP  TYPE=ERRPROC
          ,CODE=(error-code,...)
          ,GROUP=error-group-index
```

TYPE=ERRPROC

indicates the start of a user-supplied error processor.

CODE=(error-code,...)

specifies the error codes that make up the error group, and which are therefore handled by the error processor supplied. The operand is coded as a sublist of 2-character representations of 1-byte hexadecimal codes. (The parentheses can be omitted for a single code.) For each code specified, the error group index is placed at the equivalent offset in the translate table. Thus, when this code occurs, the appropriate error processor can be identified.

GROUP=error-group-index

specifies an error group index for the error processor. This index is used to name the error processor, locate its address from the error processor vector table (EPVT), and optionally associate it with an ESB in each NEB. The index specified must be a 2-character representation of a 1-byte hexadecimal number in the range X'01' through X'FF' (a leading zero can be omitted). The error processor name has the form NEPROCxx, where "xx" is the error group index. A CSECT statement of this name is generated, which causes the error processor code to be assembled at the end of the node error program module and to have its own addressability.

If you intend to add your own error processors to the sample node error program, you should consider the following factors:

- The layout of the communication area. The communication area is described in detail in Figure 25 on page 405.
- The fact that certain functions cannot be used within DFHZNEP. (See "Restrictions on the use of EXEC CICS commands" on page 421.)
- The register conventions used by the sample node error program. These are described in Table 108 on page 418.

the sample node error program

Register	Use
0	Work register
1	Address of the EXEC parameter list
2	NEB base register (DFHSNEP only)
3	ESB base register (DFHSNEP only) NEP error class register (DFHZNEPI only)
4	NEP name pointer register (DFHZNEPI only)
5	NEP interface base register (DFHZNEPI only)
6	Work register
7	Work register
8	Work register
9	Work register
10	Code base register
11	Address of the EIB
12	Address of the communication area
13	Address of DFHEISTG storage
14	CSVT base and error processor link register Common subroutine link register
15	Error processor branch register Common subroutine branch register.

Notes:

1. Register 14 must be saved for return from error processors. The common subroutine vector table (CSVT) is coded after the BALR to the error processor and so this register is also the CSVT base.
2. Registers 1, 10, 12, 13, 14, and 15 are set up on entry to error processors.
3. Registers 14 through 11 can be saved by error processors in an area reserved in EXEC interface storage at label NEPEPRS. Registers 15 through 11 do not need to be restored before return from error processors.
4. Registers 4 through 9 can be saved by common subroutines in an area reserved in EXEC interface storage at label NEPCRSR. They must be restored before return from the subroutines.

DFHSNET—generating the node error table

The DFHSNET macro is used to generate a node error table. Each node error table that you generate must be defined to CICS.

```
DFHSNET [NAME=DFHNET|name]
        [,COUNT=100|threshold]
        [,ESBS=1|(index,length,...)]
        [,NEBNAME=(name,...)]
        [,NEBS=10|number]
        [,TIME=(7,MIN)|(interval,units)]
```

NAME=DFHNET|name

specifies the identifier to be included in the NET header. It must be a string of one through eight characters. This operand is optional, and the default is DFHNET.

COUNT=100|threshold

specifies the error count threshold that is to be stored in the NET header for use by the common subroutines to update standard ESBs. If the threshold is exceeded, the error processor that invoked the subroutine is informed by a return code. The maximum value is 32 767. This operand is optional, and the default is 100.

ESBS=1|(index,length,...)

specifies the ESB structure for each NEB. This operand is coded as a sublist. Each element of the sublist comprises two values: "index" specifies an error group index for which an ESB is to be included in the NEB; "length" specifies the status area length, in bytes, for that ESB. The parentheses can be omitted for a single element. The "index" must be specified as a 2-character representation of a 1-byte hexadecimal number in the range X'01' through X'FF' (a leading 0 can be omitted). The "length" is constrained only because an 8-byte NEB header plus a 4-byte header for each ESB must be contained within the maximum NEB length of 32 767 bytes. If a null value is specified, a standard ESB with a status area length of 10 bytes is assumed. This is suitable for use by the common subroutines in maintaining a time-stamped error count.

This operand is optional and defaults to 1. This causes each NEB to be generated with one ESB for error group 1 with a status area length of 6 bytes.

NEBNAME=(name,...)

specifies the names of nodes that are to have a permanently assigned NEB. The names specified are assigned, in the order specified, to the set of NEBs requested by the NEBS operand. Any remaining NEBs are available for dynamic allocation to other nodes as errors occur. The name must be a string of 1 through 4 characters. The parentheses can be omitted for a single name. This operand is optional and has no default.

NEBS=10|number

specifies the number of NEBs required in the NET. The maximum valid number is 32 767; the default is 10.

TIME=(7,MIN)|(interval,units)

specifies the time interval that is to be stored in the NET header for use by the common subroutines to maintain error counts in standard ESBs. If the threshold specified in the COUNT operand is not exceeded before this time interval elapses, the error count is reset to 0. Specify "units" as SEC, MIN, or HRS. The maximum values for "interval" are as follows: (86400,SEC), (1440,MIN), or (24,HRS). This operand is optional, and the default is set to (7,MIN).

DSECTS

The following DSECTS are provided:

Node Error Table Header: This contains the table name and common information relevant for all the node error blocks (NEBs) in the table.

DFHNETH	DSECT		
NETHNAM	DS	CL8	Table name
NETHNBN	DS	H	Number of NEBs in table
NETHNBL	DS	H	Length of NEBs in table
NETHTIM	DS	PL8	Error count time interval
NETHECT	DS	H	Error count threshold
NETHFLG	DS	X	Flag byte
NETHINI	EQU	X'01'	Table initialized
	DS	X	Reserved
NETHFNB	DS	0F	First NEB

Node Error Block: The table contains node error blocks that are used for recording error information for individual nodes. These can be permanently assigned to specific nodes or dynamically assigned at the request of error processors.

DFHNETB	DSECT		
NEBNAM	DS	CL4	Node name
NEBFLG	DS	X	Flag byte
NEBPERM	EQU	X'01'	Permanently assigned NEB
	DS	XL3	Reserved
NEBFESB	DS	0X	First NEB

Error Status Block: The NEBs can contain error status blocks. These are reserved for specific error processors and are identified by the corresponding error group index. An ESB can have a format defined by you, or can have a standard format suitable for counting errors over a fixed time interval.

DFHNETE	DSECT		
ESBEGI	DS	X	Error group index
ESBFLG	DS	X	Flag byte
ESBSTAN	EQU	X'01'	Standard format ESB
ESBTTE	EQU	X'02'	Time threshold exceeded
ESBCTE	EQU	X'04'	Count threshold exceeded
ESBSLEN	DS	XL2	Status area length
ESBHLEN	EQU	*-DFHNETE	ESB header length
ESBSTAT	DS	0X	Status area

The following fields apply to the standard format:

ESBTIM	DS	PL8	Time stamp
ESBEC	DS	XL2	Error count

Common Subroutine Vector Table: The CSVT provides error processors with addressability to the common subroutines. The error processor link register gives addressability to the CSVT and so the first section of the DSECT overlies the code required to branch around the actual table.

DFHNEPC	DSECT		
	DS	F	Load instruction
	DS	F	Branch instruction
CSVNNEP	DS	A	Node error program base address
CSVNESBL	DS	A	NEPESBL - ESB locate routine
CSVNEBD	DS	A	NEPNEBD - NEB delete routine
CSVTECUP	DS	A	NEPECUP - error count update routine

User-written node error programs

You can write your own NEP in any of the CICS-supported languages. However, CICS-supplied NEP code is provided in assembler language only. The communication area parameter list is supplied in assembler-language and C versions. The names of the supplied source files, copy books, and macros, and the libraries in which they can be found, are listed in Table 109.

Table 109. Supplied source files, copy books, and macros

Name	Type	Description	Library
DFHZNEP0	Program	Default node error program (assembler language)	CICS410.SDFHSAMP
DFHZNEPX	Source	Default NEP (embedded by DFHZNEP0 via COPY statement)	CICS410.SDFHSAMP
DFHSNEP	Macro	Sample NEP program generator (assembler language)	CICS410.SDFHMAC
DFHZNEPI	Macro	NEP interface generator (for multiple NEPs)	CICS410.SDFHMAC
DFHNEPCA	Macro	Assembler-language communication area	CICS410.SDFHMAC
DFHNEPCA	Copy book	C-language communication area	CICS410.SDFHC370

If you code in assembler language, you can use the sample NEP as a framework on which to construct your own node error program.

Restrictions on the use of EXEC CICS commands

There are certain restrictions on the commands that a NEP can issue. **The use of any commands which require a principal facility causes unpredictable results, and should be avoided.** In particular, you should not use commands which invoke the following functions:

- Terminal control (“CEMT-type” commands, such as EXEC CICS INQUIRE TERMINAL, are permissible)
- BMS (except routing)
- ISC communication (including function shipping). This includes START requests for remote transactions. Such requests are not recommended because CSNE (Node Abnormal Condition task) might become suspended while doing an ALLOCATE to the remote system.

If you need to start a remote transaction, start a local transaction which start a remote one in turn.

user-written node error programs

You should also note that you cannot use the NEP to suppress DFHZNAC messages.

Entry and addressability

On entry, your NEP should issue the commands:

```
EXEC CICS ADDRESS COMMAREA  
EXEC CICS ADDRESS EIB
```

These commands provide addressability to the communication area passed by DFHZNAC, and to the EXEC interface block, respectively.

If you write your node error program in assembler language, you generate the communication area DSECT by coding:

```
DFHNEPCA TYPE=DSECT
```

If you write your program in C/370, you include the communication area definitions by coding:

```
#include <dfhnepca>
```

Coding for the 3270 'unavailable printer' condition

The 'unavailable printer' condition arises when a print request is made using the 3270 print request facility, and there are no printers on the control unit, or when the printers are in one of the following conditions:

- Out of service
- Not in TRANSCIVE or RECEIVE status for automatic transaction initiation
- With a task currently attached
- Busy on a previous operation
- Requiring intervention.

The procedure is applicable to 3270 logical units or to the 3270 compatibility mode logical unit when using the PRINTER and ALTPRINTER operands of the CEDA DEFINE TERMINAL command.

The terminal control program recognizes this condition, and issues a READ BUFFER request to collect the data into a terminal I/O area. The TIOA is of the same format as it is when an application program has issued a terminal control read buffer request.

The terminal control program VTAM section (DFHZCP) then queues the TCTTE to the node abnormal condition program with error code X'42' (TCZCUNPRT). The node abnormal condition program (DFHZNAC) writes to the CSNE transient data queue:

- DFHZC2497 UNAVAILABLE PRINTER (device types 3270P and LUTYPE3)
- DFHZC3493 INVALID DEVICE TYPE FOR A PRINT REQUEST (all other printer device types).

Before linking to the node error program, DFHZNAC inserts the primary and secondary printer netnames and terminal IDs into the communication area, indicating also whether either printer is eligible for a print request. DFHZNAC links to the node error program with no default actions set.

On return from the node error program, DFHZNAC checks the additional system parameter TWAUPRRRC in the communication area (see Figure 25 on page 405) and, based on its contents, performs one of the following actions:

- If your NEP sets TWAUPRRRC to X'FF' (-1), DFHZNAC assumes that the node error program has disposed of the data to be printed and therefore takes no further action.
- If your NEP sets TWAUPRRRC to zero, DFHZNAC assumes that no printer is available and takes no further action.
- If your NEP sets TWAUPRRRC to neither zero nor -1, DFHZNAC assumes that one of either field TWAPNETN or field TWAPNTID is set. (If both are set, TWAPNTID(termid) takes precedence.) An interval control PUT is performed to the provided terminal. The transaction to be initiated is CSPP (print program), and the time interval is zero.
 - If an error occurs on the interval control PUT, DFHZNAC writes the 'DFHZC2496 IC FAILURE' message to the destination CSNE. DFHZNAC then links to the node error program again with the TWAUPRRRC field set to -2. This is done to give the node error program a last chance to dispose of the data. On the second return from the node error program to DFHZNAC, the latter reexamines TWAUPRRRC. If TWAUPRRRC is -1, then the node error program has disposed of the data.
 - If no error occurs on the interval control PUT, DFHZNAC checks for the following printer conditions:
 - 'Out of service'
 - 'Intervention required'
 - Any condition other than RECEIVE or TRANSCEIVE status.

If one of these conditions is true, DFHZNAC issues the 'DFH2495 PRINTER OUTSERV/IR/INELIGIBLE-REQ QUEUED' message to the destination CSNE.

Finally, DFHZNAC terminates any print requests on the originating terminal and performs normal action flag processing on that terminal.

Coding for session failures

Following some categories of error associated with logical unit or path failures, the session between CICS and the logical unit may be lost. The default action taken by DFHZNAC may be to put the TCTTE out of service.

A method of automatically reacquiring the session is for your node error program to alter the default DFHZNAC actions and to keep the TCTTE in service. Your node error program can then issue an EXEC CICS START TERMID(name) command against that TCTTE for a transaction written in a similar manner to the CICS "good morning" signon message (CSGM). When the transaction is initiated using automatic transaction initiation (ATI), CICS tries to reacquire the session. If the session fails again, DFHZNAC is reinvoked and the process is repeated.

The time specified on the EXEC CICS START command is determined by installation-dependent expected-mean-time-to values.

If used in this way, the initiated transaction can write an appropriate signon message when the session has been acquired. Note, however, that if

user-written node error programs

LOGONMSG=YES is specified on the CEDA DEFINE TYPETERM command, the CICS “good morning” message is also initiated when the session is opened. Refer to “Restrictions on the use of EXEC CICS commands” on page 421.

Coding for specific VTAM sense codes

Figure 27 shows how your NEP error processors could test for the presence of specific VTAM sense codes.

```
TEST1  EQU  *
        CLC  TWASENSR(2),SNS1          SENSE CODE EQUAL TO NNNN
        BNE  TEST2                     NO, THEN NEXT TEST
        NI   TWAOPT1,TWAOAF            PRINT ACTION MESSAGES ONLY
        OI   TWAOPT2,TWAOAS+TWAOAR+TWAOAT ABANDON SEND,RECEIVE AND TASK
        NI   TWAOPT2,255-TWAOASM       SET SIMLOGON OFF
        OI   TWAOPT3,TWAOINT          SET INTLOG NOW ALLOWED
        NI   TWAOPT3,255-TWAOININT     OR RESET NOINTLOG
        B    END
        .
        .
        .
SNS1    DC   X'NNNN'
```

Figure 27. Sample code, showing how your node error program could test for specific VTAM sense codes

Writing multiple NEPs

You can write several node error programs, as described in “Multiple NEPs” on page 401. When an error occurs, the node abnormal condition program passes control to an interface module, DFHZNEPI, which determines the transaction class and passes control to the appropriate node error program.

If only one node error program is used, the interface module (DFHZNEPI) is not required. If the node error program is named DFHZNEP, the node abnormal condition program branches directly to that. If more than one node error program is used, the interface module (DFHZNEPI) is required. In this case, the node error programs must be given names other than DFHZNEP. There must be an installed program definition for every node error program generated.

DFHZNEPI macros

The following macros are required to generate the node error program interface module (DFHZNEPI):

- DFHZNEPI TYPE=INITIAL to specify the name of the default transaction-class routine.
- DFHZNEPI TYPE=ENTRY to associate a transaction-class with a transaction-class error handling routine.
- DFHZNEPI TYPE=FINAL to end the DFHZNEPI entries.

The DFHZNEPI interface module must be generated when you require the node abnormal condition program to pass control to the appropriate user-written node error program for resolution of the error.

DFHZNEPI TYPE=INITIAL—specifying the default routine

The DFHZNEPI TYPE=INITIAL macro specifies the name of the default transaction-class routine to be used for the DFHZNEPI module.

```
DFHZNEPI TYPE=INITIAL
          [,DEFAULT=name]
```

DEFAULT=name

specifies the name of the default transaction-class routine to be used. A link is made to this default routine if you specify for the transaction (using the CEDA DEFINE PROFILE, CEDA DEFINE SESSIONS, or CEDA DEFINE TYPETERM command) a NEPCCLASS value of 0 (the default) or higher than 255, or if you do not specify a transaction-class routine using the DFHZNEPI TYPE=ENTRY macro for the class specified on the NEPCCLASS operand.

If either of the preceding conditions is true, but you do not code the DEFAULT operand, then no routine is invoked.

The DFHZNEPI TYPE=INITIAL macro must always be specified, and must be placed before any other forms of the DFHZNEPI macro. Only one TYPE=INITIAL macro can be specified.

DFHZNEPI TYPE=ENTRY—specifying a transaction-class routine

You use the DFHZNEPI TYPE=ENTRY macro to associate a transaction class (NEPCCLASS) with a transaction-class error handling routine. The format of this macro is:

```
DFHZNEPI TYPE=ENTRY
          ,NEPCLAS=integer
          ,NEPNAME=name
```

NEPCLAS=integer

specifies the transaction-class, and must be in the range 1 through 255. No value should be specified that has been specified in a previous DFHZNEPI TYPE=ENTRY instruction.

NEPNAME=name

specifies a name for the transaction-class routine to be associated with the specified transaction-class. An error condition results if the name is specified either as DFHZNEP, or is longer than 8 characters.

Note: You can use the sample node error program (with a name other than DFHZNEP) as a transaction-class routine for the interface module, DFHZNEPI.

DFHZNEPI TYPE=FINAL—terminating DFHZNEPI entries

```
DFHZNEPI TYPE=FINAL
```

TYPE=FINAL

completes the definition of module DFHZNEPI and must be specified last. The assembly should be terminated by an END statement with no entry name specified, or by the statement: END DFHZNENA.

Handling shutdown hung terminals in the node error program

```
# Error Code: X'6F'
# Symbolic Name: TCZSDAS
# Message Number: DFHZC2351
```

```
# For error code X'6F', DFHZNAC passes the setting of TCSACTN and the
# DFHZC2351 reason code to DFHZNEP, and DFHZNEP can modify the force-close
# action for the current terminal.
```

How DFHZNAC passes the setting of TCSACTN to DFHZNEP

```
# For error code X'6F', DFHZNAC passes the setting of the TCSACTN system
# initialization parameter to DFHZNEP by setting TWAOSCN. TWAOSCN off (B'0')
# indicates TCSACTN=NONE, and TWAOSCN on (B'1') indicates
# TCSACTN=UNBIND.
```

How DFHZNAC passes the DFHZC2351 reason code to DFHZNEP

```
# For error code X'6F', the DFHZC2351 reason code is passed to DFHZNEP in the
# NEP communications area (NEPCA) field TWATRSN. TWATRSN is a 1-byte code
# field. Note that, currently, TWATRSN overlays TWAREASN (also a 1-byte field).
# The codes, and their meaning, are:
```

```
# 01 Request in progress    06 Waiting for RTR
# 02 Task still active      07 BID in progress
# 03 Waiting for SHUTC      08 Other TC work pending
# 04 Waiting for BIS        99 (X'63') Undetermined
# 05 Waiting for UNBIND
```

```
# See Terminal Control message DFHZC2351 for further details.
```

How DFHZNEP can modify the force-close action for the current terminal

```
# For error code X'6F', DFHZNEP can modify the force-close action, for the current
# terminal, by setting TWAOSCN. If DFHZNEP sets TWAOSCN off (B'0'),
# DFHZNAC will not attempt to force-close the terminal (TCSACTN=NONE),
# however, if DFHZNEP sets TWAOSCN on (B'1'), DFHZNAC will attempt to
# force-close the terminal (TCSACTN=UNBIND). Internally, DFHZNAC achieves this
# by converting the TWAOSCN normal close to a TWAOCN forced close. DFHZNEP
# cannot modify either of the system initialization parameters TCSWAIT or
# TCSACTN.
```

Using the node error program with XRF or persistent sessions

This section contains guidance information about the NEP in an XRF or persistent sessions environment for CICS/ESA 4.1.

The node error program in an XRF environment

The CICS extended recovery facility (XRF) is described in the *CICS/ESA 3.3 XRF Guide*. If you are using XRF, a VTAM failure in your active CICS system may cause a takeover by the alternate CICS system. Each terminal from the failing system that is switched to the alternate system is passed to DFHZNAC for “conversation-restart” processing. This is similar to “session opened” processing for a normal session start.

One of the steps in the conversation-restart process is to link to the node error program with error code X'3F'. If you want to be able to change any of the system-wide recovery notification options (whether terminal users are notified of a recovery, the recovery message, or the recovery transaction) for some terminals, you should write your own error processor to handle code X'3F'. (For details of the recovery notification parameters passed to the NEP, see the listing of communication area fields in Figure 25 on page 405.)

The node error program with persistent session support

Persistent session support is described in the *CICS/ESA Recovery and Restart Guide*.

One of the steps in the conversation-restart process is to link to the node error program with error code X'FD'. If you want to be able to change any of the system-wide recovery notification options (whether terminal users are notified of a recovery, the recovery message, or the recovery transaction) for some terminals, you should write your own error processor to handle code X'FD'.

When using persistent sessions, note the following:

- When a session has been recovered, it may be a good idea to run NEP processing equivalent to your normal “session started” (code X'48') processing, because code X'48' is not passed on session recovery when persistent sessions are used.
- In certain situations where sessions have persisted over a failure but have been unbound on restart (for example, a COLD start occurs after a CICS failure), the NEP is not driven. (In systems without persistent session support, the NEP is always driven with code X'49', “session terminated”, when a VTAM session terminates.) Conditions leading to the issuing of the following messages do not drive the NEP. The messages appear on the system console:

```
DFHZC0120      DFHZC0124
DFHZC0121      DFHZC0129
DFHZC0122      DFHZC0130
DFHZC0123
```

Conditions leading to the issuing of messages DFHZC0125 and DFHZC0131 drive the NEP with codes X'FB' and X'FC' respectively. It is recommended that you run NEP processing equivalent to your normal “session terminated” NEP processing for these conditions.

- If zero is specified on the AIRDELAY system initialization parameter, autoinstalled terminals are not recovered after a restart. Similarly, if the delay period specified on AIRDELAY expires before an autoinstalled terminal is used after a restart, the terminal definition is deleted. In these circumstances, any expected NEP processing as a result of CLSDSTP=NOTIFY being coded does not take place.

Changing the recovery notification

The method of recovery notification for a terminal is defined using the RECOVNOTIFY option of the TYPETERM definition, which is described in the *CICS/ESA Resource Definition Guide*. This is the most efficient way to specify the recovery notification method for the whole network, because CICS initiates the notification procedure during the early stages of takeover.

You can use the node error program to change the recovery notification method for some of the switched terminals. For example, you may want most terminals of a particular type to receive the recovery message at takeover, but the rest to get no notification that service has been restored. To achieve this, you could code RECOVNOTIFY(MESSAGE) in the TYPETERM definition, and then use the node error program to change the recovery notification to NONE for the relatively few terminals that are not to be notified.

Changing the recovery message

If you define a terminal with RECOVNOTIFY(MESSAGE) in its TYPETERM definition, a recovery message is sent to the terminal after takeover. By default, for an XRF takeover, this is the following CICS-supplied message in BMS map DFHXRC1 of map set DFHXMSG:

```
CICS/ESA has recovered after a system failure.  
Execute recovery procedures.
```

For a persistent sessions recovery, BMS map DFHXRC3 is used; this map prefixes the above message with CICS/ESA message number DFHZC0199. You can specify your own map set in the node error program if you want to change the recovery message for some of the switched terminals. This could be useful, for example, if signon security is in force and you want to tell terminal users to sign on again. The map set that you specify must have an installed program definition. If you choose to change the recovery message for all terminals, it would be more efficient to replace the CICS-supplied map with your own.

Changing the recovery transaction

The recovery transaction, to be started at a terminal after takeover, is specified using the RMTRAN system initialization parameter. This is the most efficient way of specifying a recovery transaction for the network. You can specify a different transaction for some of the switched terminals by overwriting field TWAXTRAN in the communication area. The transaction that you specify must have an installed transaction definition, and the terminal must be defined with the option ATI(YES).

If the transaction specified in TWAXTRAN does not exist, CICS tries to start the CSGM transaction. If this also fails, CICS terminates the session.

Using the node error program with VTAM generic resources

The EXEC CICS ISSUE PASS command can be used (either from an application program, or by means of CECI) to disconnect a terminal from CICS, and transfer it to the VTAM application specified on the LUNAME option. For example, to transfer a terminal from this CICS to another terminal-owning region, you could issue the command:

```
CECI ISSUE PASS LUNAME(applid)
```

where `applid` is the APPLID of the TOR to which the terminal is to be transferred.

If your TORs are members of a VTAM generic resource group, you can transfer a terminal to any member of the group by specifying LUNAME as the generic resource name. For example:

```
CECI ISSUE PASS LUNAME(grname)
```

where `grname` is the generic resource name. VTAM chooses the most suitable group member to which to transfer the terminal. (If you need to transfer a terminal to a specific TOR within the CICS generic resource group, you must specify LUNAME as the member name—that is, the CICS APPLID, as in the first example.)

Note that, if the system that issues an `ISSUE PASS LUNAME(grname)` command is the *only* CICS currently registered under the generic resource name (for example, the others have all been shut down), the `ISSUE PASS` command does **not** fail with an INVREQ. Instead, the terminal is logged off and message DFHZC3490 is written to the CSNE log.

You may want to code your node error program to deal with the situation when message DFHZC3490 (DFHZNAC error code X'C3') is issued.

the node error program and generic resources

Chapter 10. Writing a program to control autoinstall of terminals

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

+ This chapter describes how to write a program to control the automatic installation
 + of locally-attached terminals (including APPC single-session devices). For
 + information about controlling the automatic installation of local APPC connections
 + that are initiated by BIND requests, see Chapter 11. For information about
 + controlling the installation of shipped terminals and connections, see Chapter 12.

The chapter is divided into the following sections:

1. **“Preliminary considerations”**.
2. **“The autoinstall control program at INSTALL”** on page 433. This contains:
 - “The communication area at INSTALL for terminals” on page 434
 - “How CICS builds the list of autoinstall models” on page 436
 - “Returning information to CICS” on page 437
 - “CICS action on return from the control program” on page 440.
3. **“The autoinstall control program at DELETE”** on page 441.
4. **“Naming, testing, and debugging your autoinstall control program”** on page 442.
5. **“The sample programs and copy books”** on page 444.

Preliminary considerations

You use the CEDA DEFINE command to define VTAM-connected resources to your CICS system. This command puts your definitions onto the CICS system definition file (CSD). Having defined your resources, you can allow CICS to create entries in the TCT automatically, whenever VTAM resources request connection to CICS. A particular advantage of automatic installation (autoinstall) is that the resource occupies storage in the TCT only while it is connected to CICS.

You use the autoinstall control program to select some of the data needed to automatically install your terminals—notably the CICS terminal name and the model name to be used in each instance. You can use the CICS-supplied autoinstall program, or extend it to suit your own purposes.

For an overview of autoinstall, see the *CICS/ESA Resource Definition Guide*. You should also read the sections in the same manual that describe the CEDA commands that create the environment in which your control program can work.

If you choose automatic installation for some or all of your terminals, you must:

- Create some model TERMINAL definitions.

the autoinstall control program for terminals

- Define the terminals to VTAM, so that their definitions in VTAM match the model TERMINAL definitions in CICS.
- If you are using model terminal support (MTS), define the MTS tables to VTAM.
- Use the default autoinstall control program for terminals (DFHZATDX), or write your own program, using the source-code of the default program and the customization examples in this chapter as a basis. (You can write an entirely new program if the default program does not meet your needs, but you are recommended to try a default-based program first.) You can write your program in any of the languages supported by CICS—the source of the default program is provided in assembler language, COBOL, PL/I, and C. You can rename your user-written program.

Notes:

1. Documentation for APAR PN 83855 added 11th June 1996.

If you use the COBOL version of the supplied autoinstall control program, DFHZCTDX, and compile it with the VS COBOL II compiler, you must run the program under the Language Environment.

- #### 2. You can have only one active autoinstall control program to handle both terminals and APPC connections. You specify the name of the active program on the AEXIT system initialization parameter. The DFHZATDY program described in Chapter 11 provides the same function for terminal autoinstall as DFHZATDX, but in addition provides function to autoinstall APPC connections initiated by BIND requests. Both DFHZATDX and DFHZATDY provide function to install shipped terminals and connections. So, for example, if you want to autoinstall APPC connections as well as VTAM terminals, you should use a customized version of DFHZATDY, rather than DFHZATDX.

Coding entries in the VTAM LOGON mode table

CICS uses the logmode data in the VTAM LOGON mode table when processing an autoinstall request. Autoinstall functions properly only if the logmode entries that you define to VTAM have matches among the model TERMINAL definitions that you specify to CICS.

The tables in Appendix A, “Coding entries in the VTAM LOGON mode table” on page 669 show, for a variety of possible terminal devices, what you must have coded on the VTAM MODEENT macros that define, in your logmode table, the terminals that you want to install automatically. Between them, the tables show the values that must be specified for each of the operands of the MODEENT macro.

Some of the examples in the appendix correspond exactly to entries in the IBM-supplied logon mode table called ISTINCLM. Where this is so, the table gives the name of the entry in ISTINCLM.

Using model terminal support (MTS)

CICS/ESA supports the model terminal support (MTS) function of VTAM 3.3 and above.

Using MTS, you can define the model name, the printer (PRINTER), and the alternate printer (ALTPRINTER) for each terminal in a VTAM table. This

information is sent by VTAM in an extended CINIT RU. CICS captures it as part of autoinstall processing at logon, and uses it to create a TCTTE for the terminal.

If you are using MTS, you must use the CICS/ESA 4.1 version of DFHZATDX. If you write your own autoinstall control program, you must ensure that it takes account of the extended data pointed to in the communication area. (See “The autoinstall control program at INSTALL.”)

Coding entries for MTS

You need to define model names (MDLTAB, MDLENT, and MDLPLU macros) and printer and associated printer names (ASLTAB, ASLENT, and ASLPLU macros) to VTAM.

| The autoinstall control program for terminals, DFHZATDX

In addition to managing your resource definition, your autoinstall control program can perform any other processes that you want at this time. Its access to the command-level interface is that of a normal, nonterminal user task. Some possible uses are listed on page 444.

The control program is invoked when:

1. An autoinstall INSTALL request is being processed
2. An autoinstall DELETE request has just been completed
3. An autoinstall request has previously been accepted by the user program, but the subsequent INSTALL process has failed.

On each invocation of the autoinstall control program, a parameter list is passed (using a communication area), describing the function being performed (INSTALL or DELETE), and providing data relevant to the particular event. (In case 3 above, the control program is invoked as if for DELETE).

The INSTALL and DELETE events are now described in detail.

The autoinstall control program at INSTALL

| If autoinstall is operative, the autoinstall control program is invoked at INSTALL for:

- + • Local VTAM terminals
- + • Local APPC single-session connections initiated by a CINIT
- + • Local APPC parallel-session connections initiated by a BIND
- + • Local APPC single-session connections initiated by a BIND
- + • Shipped terminals and connections.

| On each invocation, CICS passes a parameter list to the control program by means of a communication area addressed by DFHEICAP. The parameter list passed at INSTALL of local APPC connections initiated by BIND requests is described in “The communication area at INSTALL for APPC connections” on page 454. The parameter list passed at INSTALL of shipped terminals and connections is described in “The communications area at INSTALL for shipped terminals” on page 462. This section describes only INSTALL of local terminals (including APPC single-session connections initiated by a CINIT).

The control program is invoked at INSTALL for terminals (and APPC single-session connections initiated by a CINIT) when both:

autoinstall control program at INSTALL

- A VTAM logon request has been received from a resource eligible for automatic installation whose NETNAME is not in the TCT.
- Autoinstall processing has been completed to a point where information (a terminal identifier and autoinstall model name) from the control program is required to proceed.

| The communication area at INSTALL for terminals

The layout of the communication area is shown in Figure 28.

Fullword 1	Standard Header	
Byte 1	Function Code	(X'F0' for INSTALL)
Bytes 2 - 3	Component Code	Always 'ZC'
Byte 4	Reserved	Always X'00'
Fullword 2	Pointer to NETNAME_FIELD	
Fullword 3	Pointer to MODELNAME_LIST	
Fullword 4	Pointer to SELECTED_PARMS	
Fullword 5	Pointer to CINIT_RU	

Figure 28. Autoinstall control program's communication area at INSTALL

The parameter list contains the following information:

1. Standard Header. Byte 1 indicates the request type (this is character '0' for INSTALL).
2. Pointer to a 2-byte length field, followed by the NETNAME of the resource requesting LOGON.
3. Pointer to an array of names of eligible autoinstall models. The array is preceded by a 2-byte field describing the number of 8-byte name elements in the array. If there are no elements in the array, the number field is set to zero.
4. Pointer to the area of storage that you use to return information to CICS, and where the MTS information from the VTAM CINIT is stored.
5. Pointer to VTAM LOGON data (the CINIT request unit). The data is preceded by a 2-byte length field, indicating the length of the CINIT request unit, and includes the 3-character NS header. The format of the CINIT request unit is described in the *SNA Network Product Formats* manual.

CICS passes a list of eligible autoinstall models in the area addressed by fullword 3 of the parameter list.

If the model name is not supplied by MTS, the control program must select a model from this list that is suitable for the device logging on, and move the model name to the first 8 bytes of the area addressed by fullword 4 of the parameter list.

For example, if a 3270 printer attempts to autoinstall, the subset of matching models includes all the types in VTAM category 2 that you have defined as models. This subset could include any of the following:

- DEVICE(3270) TERMMODEL(2)
- DEVICE(3270) TERMMODEL(1)
- DEVICE(3270P) TERMMODEL(2)
- DEVICE(3270P) TERMMODEL(1)
- DEVICE(3275) TERMMODEL(2)
- DEVICE(3275) TERMMODEL(1).

The control program selects one model from this list, and CICS uses this model to build the TCTTE for the device. The default autoinstall control program, DFHZATDX, always selects the first model name in the list.

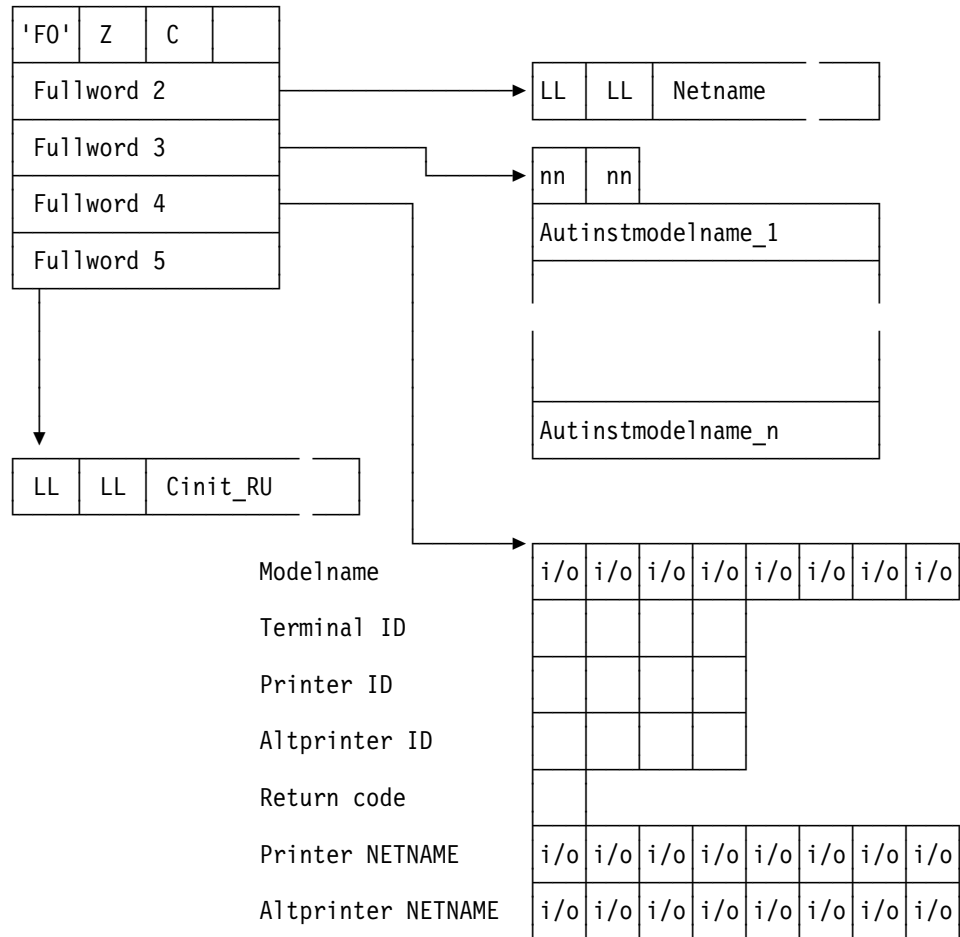
If you are not using MTS but need a printer ID or NETNAME (or an alternative printer ID or NETNAME) associated with this terminal, then your control program can supply this in the area addressed by fullword 4.

If you are using MTS, CICS passes the control program the printer and alternative printer NETNAMEs specified on the VTAM ASLTAB macro.

Before returning to CICS, the control program must supply a CICS terminal name for the device logging on, and must set the return code field to X'00' if the autoinstall request is to be allowed.

Figure 29 on page 436 shows all of these fields in their required order.

autoinstall control program at INSTALL



Note: i/o designates an input/output field. The other fields in SELECTED_PARMS are output only. Input may be supplied by MTS from the MTS CINIT.

Figure 29. Autoinstall control program's parameter list at INSTALL

How CICS builds the list of autoinstall models

+
+
+
+
+

APAR PN67608
Documentation for PN67608 added on 22 March 1995.

If CICS finds an MTS model name (and the model is defined to CICS and is compatible with the VTAM information describing the resource), it puts the model name into the model name list (Autinstmodelname_1), and also into the model name field (Modelname) in the selection list addressed by fullword 4 of the parameter list.

|
|
|

If CICS is unable to find an MTS model name in the MTS Control Vector, or the named model does not exist or is invalid, it builds the list of autoinstall models by selecting from the complete list of terminal models those models that are compatible with the VTAM information describing the resource. The complete list of autoinstall models available to CICS at any time comprises all the definitions with AUTINSTMODEL(YES) and AUTINSTMODEL(ONLY) that have been installed, both by the GRPLIST on CICS cold start, and by INSTALL GROUP commands issued

by CEDA. The *CICS/ESA Resource Definition Guide* describes the definition of models.

Table 132 on page 671 gives you the information to work out which model types could be included in the subset of models passed to the autoinstall control program when a particular terminal attempts to install. The subset is determined by the VTAM characteristics of the device attempting to log on. The number in the right-hand column of the figure indicates the selection of the subset from the full list. When a terminal with a given combination of DEVICE, SESSIONTYPE, and TERMMODEL values attempts to logon, the subset of matching models passed to the control program includes all the models with DEVICE, SESSIONTYPE, and TERMMODEL values that have a corresponding VTAM category number in the right-hand column of the table.

If CICS finds no model that exactly matches the BIND, and if the return code in the area addressed by fullword 4 of the parameter list is nonzero, then CICS issues error message 'DFHZC6987'. This message contains a "best failure" model name, which is provided for diagnostic purposes only. It is described in detail in "CICS action on return from the control program" on page 440, and in the *CICS/ESA Messages and Codes*.

Returning information to CICS

At the INSTALL event, the autoinstall control program is responsible for allowing or denying the connection of a new terminal resource to the CICS system. This decision can be based on a number of installation-dependent factors, such as security, or the total number of connected terminals. CICS takes no part in any such checking. You decide whether any such checking takes place, and how it is done.

If the INSTALL request is to proceed, the control program **must** do the following:

- Return an autoinstall model name in the first 8 bytes of the area addressed by fullword 4 of the parameter list, unless this has already been set by MTS support.

If the control program returns a model name not in the subset passed to it by CICS, CICS cannot guarantee what will happen when further processing takes place. It is the user's responsibility to determine the effect of associating any particular logon request with a particular model name, because no interface is provided to the in-storage "model" objects.

- Supply a CICS terminal name (TERMID) in the next four bytes of the return area.

DFHZATDX takes the last four nonblank characters of the NETNAME (addressed by fullword 2 of the parameter list) as the terminal name, so you must code your own autoinstall program if this does not match your installation's naming conventions. See "Setting the TERMINAL name" on page 439 for information on this.

Note that when processing an AUTOINSTALL request for an LU6.2 single session terminal the four byte terminal identifier returned by the user program is used to name a CONNECTION. It should therefore conform to the naming standards for a CONNECTION (rather than a TERMINAL) as defined in the *CICS/ESA Resource Definition Guide*. The user program could identify an LU6.2 AUTOINSTALL request in one of the following ways:

autoinstall control program at INSTALL

- Use a MODEL naming convention and examine the model name pointed to by fullword 3.
- Test bytes 14 and 15 of the CINIT BIND which is pointed to by fullword 5 for X'0602' (LU6.2).
- Set the return code to X'00'.

On entry to the autoinstall control program, the return code always has a nonzero value. If you do not change this, the autoinstall request is rejected.

If you are not using MTS, your control program can also supply or change any of the optional values, such as PRINTER and ALTPRINTER IDs or NETNAMEs, before returning to CICS. If you need information about the formats and acceptable character ranges for any of the return values, refer to the *CICS/ESA Resource Definition Guide*.

If you are using MTS, then VTAM supplies the PRINTER and ALTPRINTER NETNAMEs, if specified.

The printers need not be installed at this stage; however, they must be installed before you use Print Key support. PRINTER and ALTPRINTER IDs override PRINTER and ALTPRINTER NETNAMEs.

Note that TERMID, PRINTER, and ALTPRINTER are the only attributes of the TERMINAL definition that can be set by the autoinstall control program; all other attributes must come from one of these sources:

- The VTAM LOGMODE entry (MODEENT)
- The autoinstall model TERMINAL definition
- The TYPETERM definition that it refers to
- The QUERY function
- Model names from VTAM MDLTAB MDLENT and printers' NETNAMEs from VTAM ASLTAB ASLENT (if you are using MTS).

Notes:

1. The QUERY function overrides any extended attributes specified in the TYPETERM definition.
2. You cannot override information in the LOGMODE entry, with the model TERMINAL and TYPETERM; they must match.

If your control program decides to reject the INSTALL request, it should return to CICS with a nonzero value in the return code.

Having completed processing, the control program must return to CICS by issuing an EXEC CICS RETURN command.

Selecting the autoinstall model

If you are using model terminal support to supply the model name (and the named model exists and is valid), CICS passes the model name to your autoinstall control program—you do not need to make any further selection.

As a general rule, all the models in the list passed to your program match the VTAM data for the terminal. That is to say, a viable TCT entry usually results from the use of any one of them. (The exception to this rule involves the VTAM RUSIZE; if this value is incompatible, CICS issues an error message.) The default

autoinstall control program merely picks the first model in the list. However, this model may not provide the attributes required in all cases. For instance, you do not want a 3270 display device definition for a 3270 printer. Your control program must be able to select the model that provides the characteristics you require for this terminal—for example, security characteristics.

To save on storage, you should try to minimize the number of different models available to the control program, and the number of different TYPETERM definitions referenced by those models. If you are migrating your definitions from DFHTCT macros, look carefully at them and eliminate those that are unnecessarily different from others. Use the QUERY function for all devices that can support it. For bisynchronous devices, which do not support QUERY, one approach is to make the definition as straightforward as possible, with no special features.

If you need special models for special cases, you can use a simple mapping of, for example, NETNAME (generic or specific) to AUTINSTNAME. Your control program could go through a table of special case NETNAMEs, choosing the specified model for each. The default model would be used for any terminal not in the table. (Note that the list of models presented to the control program is in alphabetical order with one exception which is described in the notes to Table 133 on page 674.)

Setting the TERMINAL name

The TERMINAL name must be unique, and one through four characters long. For a list of the acceptable characters, see the *CICS/ESA Resource Definition Guide*. (The TERMINAL name is the identifier CICS uses for the terminal. The NETNAME is the identifier VTAM uses for the terminal.)

You may have transactions that depend on the terminals from which they are initiated, or to which they will be attached, having particular TERMINAL names. Some transactions are restricted to particular terminals and others behave in different ways, depending on the terminal. In some cases, the transaction may gather statistics about terminal use, using the TERMINAL name as a reference. The TERMINAL name may have meaning to those managing, using, or maintaining the network: it may, for instance, denote geographical location or departmental function.

The NETNAME is really more suitable for these purposes than the TERMINAL name, because it is eight characters in length. If you can use the NETNAME, the TERMINAL name can be randomly assigned by the autoinstall control program, and it does not matter if a terminal has a different TERMINAL name every time the user logs on. The control program is required, in this case, only to make the TERMINAL name unique within the system in which the terminal is to be autoinstalled. If the control program attempts to install a TCT entry for a TERMINAL name that already has a TCT entry, the installation is rejected, despite the fact that the terminal is eligible and a suitable model has been found. (By contrast, if the NETNAME already has a TCT entry, the terminal uses it and autoinstall can never be invoked.)

The default autoinstall control program creates the TERMINAL name from the last four nonblank characters of the NETNAME. This may not satisfy the requirement for uniqueness. One way of overcoming this problem is to use the EXEC CICS INQUIRE command from the control program, to determine whether the TERMINAL name is already in use. If it is, modify the last character and check again.

autoinstall control program at INSTALL

However, you may be in a situation where you must continue to use unique and predictable TERMINAL names for your terminals. Your control program must be able to assign the right TERMINAL name to each terminal, every time the user logs on. Two possible approaches to this problem are:

- Devise another algorithm to generate predictable TERMINAL names from NETNAMEs
- Use a table or file to map TERMINAL names to NETNAMEs.

Devising an algorithm avoids the disadvantages of using a table or a file, but it might be difficult to ensure both uniqueness and predictability. If some of the information in the NETNAME is not needed by CICS, it can be omitted from the TERMINAL name. An algorithm is probably most appropriate in this situation.

Using a table has two disadvantages, each of which loses you some of the benefits of autoinstall: it takes up storage and it must be maintained. You could create a table in main temporary storage, so that it is placed in extended storage, above the 16MB line. You could use a **VSAM file** rather than a table, to avoid the storage problem. However, this might be slower, because of the I/O associated with a file. The table or file can contain information such as PRINTER and ALTPRINTER, and you can add information such as AUTINSTNAME for devices that need particular autoinstall models. (See “Selecting the autoinstall model” on page 438.)

CICS action on return from the control program

When CICS receives control back from the autoinstall control program, it examines the return code field. If this is zero, and if the other required information supplied is satisfactory, CICS schedules the new resource for OPNDST in order to complete the logon request. If the installation process fails, then the control program is driven again, as though a DELETE had occurred. (See the section “The autoinstall control program at DELETE” on page 441 for details.) This is necessary to allow the program to free any allocations (for example, terminal identifiers) made on the assumption that this INSTALL request would succeed.

If the return code is not zero, then CICS rejects the connection request in the same way as it rejects an attempt by an unknown terminal to log on to CICS when autoinstall is not enabled.

For all autoinstall activity, messages are written to the transient data destination CADL. If an INSTALL fails, a message is sent to CADL, with a reason code. You can therefore check the output from CADL to find out why an autoinstall request failed.

If an autoinstall attempt fails for lack of an exact match, then details of the “best failure” match between a model and the BIND image are written to the CADL transient data destination.

The message takes the following form:

```
DFHZC6987 BEST FAILURE FOR NETNAME: nnnnnnnn,  
        WAS MODEL_NAME: mmmmmmmm,  
        CINIT BIND: ccccccc...,  
        MODEL BIND: bbbbbbbb...,  
        MISMATCH BITS: xxxxxxxx...
```


where

- 'nnnnnnnn' is the netname of the LU which failed to log on.
- 'mmmmmmm' is the name of model that gave the best failure. (That is, the one that had the fewest bits different from the BIND image supplied by VTAM.)
- 'ccccccc...' is the CINIT BIND image.
- 'bbbbbbb...' is the model BIND image.
- 'xxxxxxx...' is a string of hexadecimal digits, where 'xx' represents one byte, and each byte position represents the corresponding byte position in the BIND image. A bit set to '1' indicates a mismatch in that position between the BIND image from VTAM and the BIND image associated with the model.

A suggested course of action is as follows :

1. Determine whether a model such as 'mmmmmmm' is suitable. If there are several models that have identical BIND images, differing only in end-user options, then only the first such model is named in the above message. It will be up to your control program to make the choice, when the logmode table entry is corrected.
2. Identify the VTAM logmode table entry that is being used.
3. Check that this logmode table entry is not successfully in use with other applications, so that to change it might cause this other use of it to fail.
4. Amend the logmode table entry by switching the bits corresponding to 1-bits in the mismatch string. That is, if the bit in the VTAM BIND image corresponding to the bit position set to '1' in 'xxxxxxx...' above is '1', set it to '0'; if it is '0', set it to '1'.

More information about the meaning of the bits in a BIND image, and some more references, may be found in *ACF/VTAM Version 3 Programming*.

The autoinstall control program at DELETE

To provide symmetry of control over the autoinstall process, the autoinstall control program is also invoked when:

- A session with a previously automatically-installed resource has been ended
- An autoinstall request was accepted by the user program, but the subsequent INSTALL process failed for some reason.

To make it easier for you to write your control program, these two events can be considered to be identical. (There is no difference in the environment that exists, or in the actions that might need to be performed.)

Invoking the control program at DELETE enables you to reverse the processes carried out at the INSTALL event. For example, if the control program at INSTALL incremented a count of the total number of automatically installed resources, then the control program at DELETE would decrement that count.

The communication area at DELETE for terminals

Input to the program is via a communication area, addressed by DFHEICAP. The layout of the communication area is shown in Figure 30.

Fullword 1	Standard Header	
Byte 1	Function Code	(X'F1' for DELETE)
Bytes 2 - 3	Component Code	Always 'ZC'
Byte 4	Reserved	Always X'00'
Fullword 2	Terminal ID of terminal to be deleted	
Fullword 3	NETNAME of terminal to be deleted	
Bytes 1-2	Delete netname length	
Bytes 3-4	Start of Delete netname ID	
Next 15 bytes	Remainder of Delete netname ID	

Figure 30. Autoinstall control program's communication area at DELETE

The parameter list contains the following information:

1. Standard Header. Byte 1 indicates the request type (this is character '1' for DELETE).

Note: A value of X'FA' or X'FB' represents the deletion of a shipped terminal or connection.

2. The terminal identifier of the deleted resource, as shown in Table 110.

	1st byte	2nd byte	3rd byte	4th byte
First fullword	'F1'	'Z'	'C'	Reserved
Second fullword	ID of terminal to be deleted			
Third fullword	Length of netname to be deleted		First two bytes of netname	
Next 15 bytes	Remainder of netname			

Note that the named resource has been deleted by the time the control program is invoked, and is not therefore found by any TC LOCATE type functions.

Naming, testing, and debugging your autoinstall control program

Naming

The supplied, user-replaceable autoinstall control program for terminals and APPC single-session connections initiated by CINIT is named DFHZATDX. If you write your own version, you can name it differently.

After the system has been loaded, to find the name of the autoinstall control program currently identified to CICS, use either the EXEC CICS INQUIRE AUTOINSTALL command or the CEMT INQUIRE AUTOINSTALL command.

The default is DFHZATDX.

To change the current program:

- Use the AIXIT system initialization parameter. For guidance information about how to do this, refer to the *CICS/ESA System Definition Guide*.
- Make the change online using either the EXEC CICS SET AUTOINSTALL command or the CEMT SET AUTOINSTALL command. For further information about these commands, refer to the *CICS/ESA System Programming Reference* manual, and the *CICS/ESA CICS-Supplied Transactions* manual, respectively.

Testing and debugging

To help you test the operation of your autoinstall control program, you can run the program as a normal terminal-related application. Define your program and initiate it from a terminal. The parameter list passed to the program is described in “The autoinstall control program at INSTALL” on page 433. You can construct a dummy parameter list in your test program, upon which operations can be performed. Running your program on a terminal before you use it properly means that you can use the EDF transaction to help debug your program. You can also make the program interactive, sending and receiving data from the terminal.

If you find that CICS does not offer any autoinstall models to your program, you can create a test autoinstall program that forces the model name (AUTINSTNAME) you want. With a VTAM buffer trace running, try to log the device on to CICS. If CICS does not attempt to send a BIND, check the following:

- Does the model TERMINAL refer to the correct TYPETERM? (Or alternatively, is the TYPETERM in question referred to by the correct TERMINAL definition?)
- Is the TERMINAL definition AUTINSTMODEL(YES or ONLY)?
- Have you installed the group containing the autoinstall models (TERMINAL and TYPETERM definitions)?

If CICS attempts to BIND, compare the device’s CINIT RU to the CICS BIND, and make corrections accordingly.

It is very important that you ensure that the VTAM LOGMODE table entries for your terminals are correct, rather than defining new autoinstall models to fit incorrectly coded entries. Bear in mind, while you are testing, that CICS autoinstall does not work if a LOGMODE entry is incorrectly coded.

Note that you cannot **force** device attributes by specifying them in the TYPETERM definition. For autoinstall, the attributes defined in the LOGMODE entry must **match** those defined in the model; otherwise the model will not be selected. You cannot define a terminal in one way to VTAM and in another way to CICS.

If your control program abends, CICS does not, by default, cause a transaction dump to be written. To cause a dump to be taken after an abend, your program must issue an EXEC CICS HANDLE ABEND command.

The sample programs and copy books

The CICS-supplied default autoinstall program is an assembler-language command-level program, named DFHZATDX. The source of the default program is provided in COBOL, PL/I, and C, as well as in assembler language. The names of the supplied programs and their associated copy books, and the CICS410 libraries in which they can be found, are summarized in Table 111. Note that the COBOL, PL/I, and C copy books each have an alias of DFHTCUDS.

Language	Member name	Alias	Library
Programs:			
Assembler	DFHZATDX	—	SDFHSAMP
COBOL	DFHZCTDX	—	SDFHSAMP
PL/I	DFHZPTDX	—	SDFHSAMP
C/370	DFHZDTEX	—	SDFHSAMP
Copy books:			
Assembler	DFHTCUDS	—	SDFHMAC
COBOL	DFHTCUDO	DFHTCUDS	SDFHCOB
PL/I	DFHTCUDP	DFHTCUDS	SDFHPL1
C/370	DFHTCUDH	DFHTCUDS	SDFHC370

The module generated from the assembler-language source program is part of the pregenerated library shipped in CICS410.SDFHLOAD. You can use it without modification, or you can customize it according to your own requirements. If you choose to alter the code in the sample program, take a copy of the sample and modify it. After modification, use the DFHEITAL procedure to translate, assemble, and link-edit your module. Then put the load module into a user library that is concatenated before CICS410.SDFHLOAD in the DFHRPL statement. (This method applies to completely new modules as well as modified sample modules.) For more guidance information about this procedure, refer to the *CICS/ESA System Definition Guide*. Do not overwrite the sample with your customized module, because subsequent service may overwrite your module. You must install a new resource definition for a customized user program.

The default action of the sample program, on INSTALL, is to select the first model in the list, and derive the terminal identifier from the last four nonblank characters of the NETNAME, set the status byte, and return to CICS. If there are no models in the list, it returns with no action.

The default action, on DELETE, is to address the passed parameter list, and return to CICS with no action.

You can customize the sample program to carry out any processing that suits your installation. Examples of customization are given in “Customizing the sample program” on page 446. Generally, your user program could:

- Count and limit the total number of logged-on terminals.
- Count and limit the number of automatically installed terminals.
- Keep utilization information about specific terminals.
- Map TERMINAL name and NETNAME.

- Do general logging.
- Handle special cases (for example, always allow specific terminals or users to log on).
- Send messages to the operator.
- Exercise network-wide control over autoinstall. A network-wide, global autoinstall control program can reside on one CICS system. When an autoinstall request is received by a control program on a remote CICS system, this global control program can be invoked and data transferred from one control program to another.

Customizing the sample program

Here are three pieces of code that customize the sample program.

Assembler language

Figure 31, in assembler language, limits logon to netnames L77A and L77B. The model names utilized are known in advance. A logon request from any other terminal, or a request for a model which cannot be found, is rejected.

```

* REGISTER CONVENTIONS = *
* R0 used by DFHEICAL expansion *
* R1 -----"-----"-----"----- *
* R2 Base for Input parameters *
* R3 Base for code addressability *
* R4 Base for model name list *
* R5 Base for output parameter list *
* R6 Work register *
* R7 -----"----- *
* R8 -----"----- *
* R9 free *
* R10 Internal subroutine linkage return *
* R11 Base for EIB *
* R12 free *
* R13 Base for dynamic storage *
* R14 used by DFHEICAL expansion *
* R15 -----"-----"-----"----- *
*
* SELECT MODEL
*
*          LH R6,TABLEN          Number of valid netnames
*          LA R7,TABLE          Address the table
*
* LOOP1   CLC NETNAME(4),0(R7)    Is this netname in table?
*          BE VALIDT
*
*          LA R7,16(R7)          Next table entry
*          BCT R6,LOOP1
*
*          Now we know its not a valid netname
*          simply return and the logon is rejected
*
*          B RETURN
*

```

Figure 31 (Part 1 of 2). Example of how to customize the DFHZATDX sample program

```

*
VALIDT  CLI  SELECTED_MODELNAME,C' '  R7 now points to model name
        BNE  VALIDM1                  MTS model name supplied?
        LH   R6,MODELNAME_COUNT       Yes
        LTR  R6,R6                    Count of models
        BZ   RETURN                   Were any presented?
        LA   R8,MODELNAME             No
        LA   R8,MODELNAME             First model name
*
LOOP2   CLC  8(8,R7),0(R8)            Is this model name here?
        BE  VALIDM
*
        LA  R8,L'MODELNAME(R8)        Next model name
        BCT R6,LOOP2
*
*
*      Now we know the required model name was not presented
*      to this exit by CICS, a return rejects the logon
*
        B   RETURN
*
*
*      At this point the model name was found in those presented
*      It is given to CICS and the new termid is
*      the netname
*
VALIDM  MVC  SELECTED_MODELNAME,0(R8)  R8 was left pointing at
*                                           model name
VALIDM1 DS   0H
        MVC  SELECTED_TERM_ID,NETNAME  Use netname for termid
*                                           (4 chars)
*
*
* SELECTIONS COMPLETE, RETURN
*
        MVI  SELECTED_RETURN_CODE,X'00'  Indicate all OK
        B    RETURN                      Exit program
*
*
*      Table of netnames allowed to log on and the model name
*      necessary for the logon to be successful
*
*      Format of table :
*      Bytes 1 to 8   Netname allowed to log on
*      Bytes 9 to 16  Model required for netname
*
        DS   0D
TABLE   DC   CL8'L77A',CL8'3270064'
        DC   CL8'L77B',CL8'3270065'
TABLEN DC   Y((*-TABLE)/16)
*

```

Figure 31 (Part 2 of 2). Example of how to customize the DFHZATDX sample program

COBOL

Figure 32, in COBOL, redefines the NETNAME, so that the last four characters are used to select a more suitable model than that selected in the sample control program.

```

      .
*
* Redefine the netname so that the last 4 characters (of 7)
* can be used to select the autoinstall model to be used.
*
* The netnames to be supplied are known to be of the form:
*
*       HVMXNNN
*
* HVM is the prefix
* X  is the system name
* NNN is the address of the terminal
*
01 NETNAME-BITS.
   02 FIRST-CHRS PIC X(3).
   02 NEXT-CHRS.
      03 NODE-LETTER PIC X(1).
      03 NODE-ADDRESS PIC X(3).
   02 LAST-CHR PIC X(1).
      .
      .
PROCEDURE DIVISION.
      .
      .
*
* Select the autoinstall model to be used according to the
* node letter (see above). The models to be used are user
* defined.
*
* (It is assumed that the netname supplied in the commarea by CICS
* has been moved to NETNAME-BITS).
*
* If the node letter is C then use model AUTO2
* If the terminal netname is HVMC289 (a special case) then use
* model AUTO1.
* Otherwise (node letters A,B,D...) use model AUTO3.
*
      IF NODE-LETTER = 'C' THEN MOVE 'AUTO2' TO SELECTED-MODELNAME.
      IF NEXT-CHRS = 'C289' THEN MOVE 'AUTO1' TO SELECTED-MODELNAME.
      IF NODE-LETTER = 'A' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
      IF NODE-LETTER = 'B' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
      IF NODE-LETTER = 'D' THEN MOVE 'AUTO3' TO SELECTED-MODELNAME.
      .
      .

```

Figure 32. Example of how to customize the DFHZCTDX sample program

PL/I

Figure 33, in PL/I, extracts information from the VTAM CINIT RU, which carries the BIND image. Part of this information is the screen presentation services information, such as the default screen size and alternate screen size. The alternate screen size is used to determine the model of terminal that is requesting logon. The presented models are searched for a match, and if there is no match, the first model from those presented is used.

```

DCL 1 CINIT                                BASED(INSTALL_CINIT_PTR),
      2 CINIT LENG                          FIXED BIN(15),
      2 CINIT RU                            CHAR(256);
DCL  SAVE_CINIT                          CHAR(256);
                                          /* Temp save area for CINIT RU */
DCL 1 SCRNSZ                              BASED(ADDR(SAVE_CINIT)),
      2 SPARE                              CHAR(31),
                                          /* Bypass first part of CINIT and reach */
                                          /* into BIND image carried in CINIT */
      2 DHGT                              BIT(8),
                                          /* Screen default height in BIND PS area */
      2 DWID                              BIT(8),
                                          /* Screen default width in BIND PS area */
      2 AHGT                              BIT(8),
                                          /* Screen alternate height in BIND PS area */
      2 AWID                              BIT(8);
                                          /* Screen alternate width in BIND PS area */
DCL  NAME                                CHAR(2);
                                          /* Used to work up a screen model type */
DCL  TERMID                              PIC'9999' INIT(1) STATIC;
                                          /* Used to work up a unique termid */
DCL  ENQ                                CHAR(8) INIT('AUTOPRG');
                                          /* Used to prevent multiple access to termid */
/* If model name supplied by MTS, bypass model name selection */
IF SELECTED_MODELNAME ^= ' '
  THEN GO TO MODEL_EXIT;
/* Clear the CINIT save area and move in the VTAM CINIT RU.*/
/* This is useful if you fail to recognize the model */
/* of terminal; provide a dump and analyze this data */
SAVE_CINIT = LOW(256);
SUBSTR(SAVE_CINIT,1,CINIT_LEN) = SUBSTR(CINIT_RU,1,CINIT_LEN);

```

Figure 33 (Part 1 of 2). Example of how to customize the DFHZPTDX sample program

sample autoinstall programs

```
/* Now access the screen PS area in the portion of the BIND
   image presented in the CINIT RU */
/* using the screen alternate height as a guide to the model
   of terminal attempting logon. If this cannot be determined
   then default to the first model in the table */
SELECT (AHGT);          /* NOW GET SCRN ALTERNATE HEIGHT */
WHEN (12) NAME = 'M1';  /* MODEL 1 */
WHEN (32) NAME = 'M3';  /*   3 */
WHEN (43) NAME = 'M4';  /*   4 */
WHEN (27) NAME = 'M5';  /*   5 */
OTHERWISE NAME = 'M2';  /*   2 */
END;
/* Search the model entries for a matching entry. */
/* The criterion here is that a model definition should*/
/* contain the chars M2 for a model 2, and so on. */
/* For example, L3270M2, L3270M5 */
/*   TERMM2, TERMM5 */
IF MODELNAME_COUNT = 0
THEN GO TO EXIT;
DO I = 1 TO MODELNAME_COUNT;
  IF INDEX(MODELNAME(I),NAME)
  THEN GO TO FOUND_MODEL;
END;
NO_MODEL: /* Matching entry was not found, default to first model*/
SELECTED_MODELNAME = MODELNAME(1);
GO TO MODEL_EXIT;
FOUND_MODEL: /* Move the selected model name to the return area */
SELECTED_MODELNAME = MODELNAME(I);
MODEL_EXIT: /* ENQ to stop multiple updates of counter. */
/* A simple counter is used to generate unique */
/* terminal identities, so concurrent access to */
/* this counter is denied to ensure no two get */
/* the same identifier or update the counter. */

/* To use this method the program must be defined as resident.*/
EXEC CICS ENQ RESOURCE(ENQ);
SELECTED_TERMID = TERMID; /* Set SELECTED_TERMID to
                           count value */
TERMID = TERMID + 1; /* Increase the count value by 1 */
IF TERMID = 9999 THEN TERMID = 1; /* Reset if too large*/
EXEC CICS DEQ RESOURCE(ENQ);
NAME_EXIT:
INSTALL_RETURN_CODE = LOW(1);
/* Set stat field to X'00' to allow
   logon to be processed */

GO TO EXIT;
END INSTALL;
```

Figure 33 (Part 2 of 2). Example of how to customize the DFHZPTDX sample program

Chapter 11. Writing a program to control autoinstall of APPC connections

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

This chapter describes how to write a program to control the automatic installation of local APPC connections. For information about controlling the automatic installation of local terminals, see Chapter 10. For information about controlling the installation of shipped terminals and connections, see Chapter 12.

The chapter is divided into the following sections:

1. “Preliminary considerations”
2. “The autoinstall control program at INSTALL” on page 453
3. “The autoinstall control program at DELETE” on page 456
4. “The sample autoinstall control program for APPC connections” on page 456.

Note: In this chapter, “connection” and “session” are used as general terms when explaining autoinstall. The names “CONNECTION” and “SESSIONS” are used to indicate the CICS resource types used to create the definitions.

Preliminary considerations

Before the introduction of autoinstall, all resources had to be defined individually to CICS before they could be used. With the introduction of autoinstall for terminals in CICS/OS/VS 1.7, VTAM terminals could be defined dynamically on their first usage, thereby saving on storage for terminal entries, and on time spent creating the definitions.

As well as local terminals (including single-session APPC devices), you can autoinstall:

- Local APPC parallel-session connections initiated by incoming bind requests. (By “incoming” we mean that the request is initiated by the partner system.)
- Local APPC single-sessions connections initiated by incoming bind requests.
- Shipped terminals and connections.

Local APPC single-session connections initiated by a CINIT

Autoinstall of local APPC single-session connections that are initiated by CINIT requests works in the same way as autoinstall for terminals. You must provide a TERMINAL—TYPETERM model pair, and a customized version of one of the supplied autoinstall control programs, DFHZATDX or DFHZATDY. See Chapter 10, “Writing a program to control autoinstall of terminals” on page 431.

Note: If you define an APPC single-session connection using a TERMINAL/TYPETERM pair, the connection name must use the restricted set of characters permitted on a CONNECTION definition. The permitted characters are listed in the *CICS/ESA Resource Definition Guide*

Local APPC parallel-session and single-session connections initiated by a BIND

If autoinstall is enabled, and an incoming APPC BIND request is received for an APPC service manager (SNASVCMG) session (or for the only session of a single-session connection), and there is no matching CICS CONNECTION definition, a new connection is created and installed automatically.

Like autoinstall for other resources, autoinstall for APPC connections requires model definitions. However, unlike the model definitions used to autoinstall terminals, those used to autoinstall APPC links do not need to be defined explicitly as models. Instead, CICS can use any previously-installed connection definition as a “template” for a new definition. In order for autoinstall to work, you must have a template for each kind of connection you want to be autoinstalled.

Autoinstall templates for APPC connections

The purpose of a template is to provide CICS with a definition that can be used for all connections with the same properties. You customize the supplied autoinstall control program, DFHZATDY, to select an appropriate template for each new connection, depending on the information it receives from VTAM.

A template consists of a CONNECTION definition and its associated SESSIONS definitions. You should have a definition installed for each different set of session properties you are going to need.

Any installed connection definition can be used as a template, but for performance reasons, your template should be an installed connection definition that you do not actually use. The definition is locked while CICS is copying it, and if you have a very large number of sessions autoinstalling, the delay may be noticeable.

Benefits of autoinstall

Autoinstall support is likely to be beneficial if you have large numbers of APPC parallel session devices with identical characteristics. For example, if you had 1000 PS/2s, all with the same characteristics, you would set up one template to autoinstall all of them. If 500 of your PS/2s had one set of characteristics, and 500 had another set, you would set up two templates to autoinstall them.

Restart of any kind should be noticeably faster, especially when large numbers of terminals are involved.

Savings can also be made on systems management overheads, and on storage, as autoinstalled resources do not occupy space before they are used.

Requirements for autoinstall

Autoinstall of APPC connections works with any supported release of ACF/VTAM.

You can have only one active autoinstall control program for terminals and connections. You must specify the name of the active program on the AIEEXIT system initialization parameter. As well as providing function to autoinstall APPC connections initiated by BIND requests, the sample program, DFHZATDY, provides the same function for terminal autoinstall as the default control program, DFHZATDX, described in Chapter 10. Thus, you can use a customized version of DFHZATDY to autoinstall both terminals and APPC connections.

+ **Note:** Both DFHZATDX and DFHZATDY provide function to install shipped
 + terminals and connections.

| You may find the supplied version of DFHZATDY adequate for your purposes. If
 | not, you can write a customized version of the supplied program, or create your
 | own program to provide enhanced function.

| **The autoinstall control program for APPC connections**

| The purpose of the autoinstall control program is to provide CICS with any extra
 | information it needs to complete an autoinstall request. For APPC connections, the
 | control program selects the template to be used, and provides a name for the new
 | connection.

| If autoinstall is enabled, when CICS receives an APPC BIND request for an
 | SNASVCMG session (or for the only session of a single-session connection), if
 | there is no matching CONNECTION definition, CICS passes the partner's VTAM
 | NETNAME to the autoinstall control program. The control program uses
 | information from the BIND, which is passed in the communications area, to select
 | the most appropriate template on which to base a new connection.

| The control program needs to know the NETNAME or SYSID of all the templates,
 | in order to return the name of the most suitable one. If it attempts to use an
 | unsuitable template, message DFHZC6922 is issued, explaining why the template
 | is unusable.

| If the template is usable, CICS makes a copy of the definitions within it and
 | attempts to install the new CONNECTION definition. If the installation is not
 | successful, message DFHZC6903 is issued.

| **Recovery and restart**

| Autoinstalled connections are not cataloged in CICS/ESA 4.1, so they are not
 | recovered at an emergency restart or a warm restart.

| **The autoinstall control program at INSTALL**

| The autoinstall control program is invoked at INSTALL for:

- + • Local VTAM terminals
- + • Local APPC single-session connections initiated by a CINIT
- + • Local APPC parallel-session connections initiated by a BIND
- + • Local APPC single-session connections initiated by a BIND
- + • Shipped terminals and connections.

| On each invocation, CICS passes a parameter list to the control program by means
 + of a communication area addressed by DFHEICAP. The parameter list passed at
 + INSTALL of local terminals and APPC single-session connections initiated by CINIT
 + is described in "The communication area at INSTALL for terminals" on page 434.
 + The parameter list passed at INSTALL of shipped terminals and connections is
 + described in "The communications area at INSTALL for shipped terminals" on
 + page 462. This section describes only INSTALL of local APPC connections
 + initiated by BIND requests.

The communication area at INSTALL for APPC connections

The communications area is mapped by the DSECT for the assembler version of DFHZATDY, which is supplied in CICS410SDFHMAC.

```

*-----*
* APPC Install parameter list - Functions 2, 3, and 4 *
*-----*
INSTALL_APPC_COMMAREA      DSECT      Install Parameter List
*
INSTALL_APPC_STANDARD      DS  F        Standard field
                                ORG INSTALL_APPC_STANDARD
INSTALL_APPC_EXIT_FUNCTION DS  XL1      Install request type
INSTALL_APPC_PS_CINIT      EQU X'F2'   Install PS via CINIT
INSTALL_APPC_PS_BIND       EQU X'F3'   Install PS via BIND
INSTALL_APPC_SS_BIND       EQU X'F4'   Install SS via BIND
INSTALL_APPC_EXIT_COMPONENT DS  CL2     Component ID 'ZC'
                                DS  XL1   Reserved
*
                                ORG ,
INSTALL_APPC_NETNAME_PTR   DS  A        -> NETNAME      Input
INSTALL_APPC_CINIT_PTR    DS  0A       -> CINIT_RU     Input
INSTALL_APPC_BIND_PTR     DS  A        -> BIND        Input
INSTALL_APPC_SELECTED_PTR DS  A        -> Return fields Output
INSTALL_APPC_SYNCLEVEL_PTR DS  A        -> Sync level   Input
*
INSTALL_APPC_TEMPLATE_NETNAME_PTR DS  A -> Template NETNAME Output
INSTALL_APPC_TEMPLATE_SYSID_PTR DS  A -> Template SYSID Output
INSTALL_APPC_SYSID_PTR    DS  A        -> New SYSID    Output
*
TEMPLATE_NETNAME          DS  CL8      Put netname here
TEMPLATE_SYSID             DS  CL4      Put sysid here
SYSID                      DS  CL4      Put new name here
SYNCLEVEL                  DS  XL2      Synclevel of new connection

```

Figure 34. Autoinstall control program's communications area at INSTALL. For APPC connections initiated by BIND requests.

INSTALL_APPC_STANDARD header

A fullword input field comprising the following information:

INSTALL_APPC_EXIT_FUNCTION

A 1-byte field that defines the install request type. The equated values are:

INSTALL_APPC_PS_CINIT

X'F2' represents an install request for an APPC parallel-session connection from a secondary node via a CINIT request.

Note: These requests cannot be received by CICS/ESA 4.1.

INSTALL_APPC_PS_BIND

X'F3' represents an install request for an APPC parallel-session connection via a BIND.

INSTALL_APPC_SS_BIND

X'F4' represents an install request for an APPC single-session connection via a BIND.

Note: The values X'F0' and X'F1' represent, respectively, install and delete requests for terminals (including APPC single-session devices).

INSTALL_APPC_EXIT_COMPONENT

A 2-byte component code, which is set to 'ZC'.

INSTALL_APPC_NETNAME_PTR

A fullword pointer to a 2-byte length field, followed by the NETNAME to be installed (input field).

INSTALL_APPC_CINIT_PTR

A fullword pointer to an input field containing the incoming CINIT, if the incoming session is a secondary.

Note: Not applicable to CICS/ESA 4.1.

INSTALL_APPC_BIND_PTR

A fullword pointer to an input field containing the incoming BIND.

INSTALL_APPC_SELECTED_PTR

A fullword pointer to the return fields. These are in the same format as those for autoinstall of terminals.

Note that for APPC autoinstall (functions X'F3' and X'F4') only the return code is used. You return other information for APPC in other fields defined in the communications area.

INSTALL_APPC_SYNCLEVEL_PTR

A fullword pointer to a 2-byte input field specifying the syncpoint level for the connection, which is extracted from the BIND. The possible values are:

- X'0000' Synclevel 0
- X'0001' Synclevel 1
- X'0002' Synclevel 2.

INSTALL_APPC_TEMPLATE_NETNAME_PTR

A fullword pointer to an 8-byte output area that your control program can use to specify the NETNAME of the template. If the name is less than 8 bytes, it must be padded with trailing blanks. If, alternatively, you specify a CONNECTION name (in the SYSID field), the 8-byte area should be filled with blanks.

INSTALL_APPC_TEMPLATE_SYSID_PTR

A fullword pointer to a 4-byte output area that your control program can use to specify the SYSID (connection name) of the template. If the name is less than 4 bytes, it must be padded with trailing blanks. If, alternatively, you specify a NETNAME, the 4-byte area should be filled with zeros.

INSTALL_APPC_SYSID_PTR

A fullword pointer to a 4-byte output area in which your program must put the SYSID for the new autoinstalled connection. The name you supply must be unique. You can use the same or similar logic to create it that you use for creating a terminal ID. If the name is less than 4 bytes, it must be padded with trailing blanks.

If you are using recoverable resources, the SYSID chosen for a connection after a restart must be the same as that chosen in the previous CICS run.

The autoinstall control program at DELETE

There is no delete function for APPC connections equivalent to that for autoinstalled terminals.

The sample autoinstall control program for APPC connections

The sample control program for autoinstall of APPC connections is DFHZATDY. The source code, in assembler-language only, is in library CICS410.SDFHSAMP.

As well as providing function to autoinstall APPC connections initiated by BIND requests, DFHZATDY provides the same function for terminal autoinstall as the DFHZATDX program described in Chapter 10. Thus, you can use a customized version of DFHZATDY to autoinstall both terminals and APPC connections.

Default actions of the sample program

The role of DFHZATDY in installing APPC connections is to choose the template to be used (by supplying its NETNAME or SYSID), and to supply the name (SYSID) of the new connection. The actions taken by the supplied version of the program are to:

1. Examine the request type passed in the INSTALL_APPC_EXIT_FUNCTION field:

X'F0'

An incoming CINIT for a terminal or APPC single-session device. Proceed as for DFHZATDX. See Chapter 10, "Writing a program to control autoinstall of terminals" on page 431.

X'F1'

A delete request for a terminal or APPC single-session device. Proceed as for DFHZATDX. See Chapter 10, "Writing a program to control autoinstall of terminals" on page 431.

INSTALL_APPC_PS_CINIT (X'F2')

An incoming CINIT for an APPC parallel-session connection. Specify a template by setting the field pointed to by INSTALL_APPC_TEMPLATE_SYSID to 'CCPS'.

Note: This type of request cannot be received by CICS/ESA 4.1.

INSTALL_APPC_PS_BIND (X'F3')

An incoming BIND for an APPC parallel-session connection. Specify a template by setting the field pointed to by INSTALL_APPC_TEMPLATE_SYSID to 'CBPS'.

INSTALL_APPC_SS_BIND (X'F3')

An incoming BIND for an APPC single-session connection. Specify a template by setting the field pointed to by INSTALL_APPC_TEMPLATE_SYSID to 'CBSS'.

2. Specify a name for the new connection by copying the last 4 non-blank characters of the input NETNAME pointed to by INSTALL_APPC_NETNAME_PTR to the field pointed to by INSTALL_APPC_SYSID_PTR.
3. Indicate that a selection has been made by setting the return code to RETURN_OK.

Resource definitions

CICS supplies a resource definition group called DFHAI62, which defines DFHZATDY, and contains CONNECTION definitions for CCPS, CBPS, and CBSS. If you want to use the supplied version of DFHZATDY, you should append DFHAI62 to your CICS startup group list. However, if you customize DFHZATDY you will probably need to create your own definitions.

DFHZATDY is defined as follows in DFHAI62:

```
DEFINE PROGRAM(DFHZATDY)
DESCRIPTION(Assembler definition for sessions autoinstall control program)
GROUP(DFHAI62)
LANGUAGE(ASSEMBLER) RELOAD(NO) RESIDENT(NO)
USAGE(NORMAL) STATUS(ENABLED) CEDF(NO)
DATALOCATION(ANY) EXECKEY(CICS) EXECUTIONSET(FULLAPI)
```

sample autoinstall programs

+ Chapter 12. Writing a program to control autoinstall of shipped terminals

+ Considerations common to all user-replaceable programs

+ Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

+ This chapter describes how to write a program to control the installation of shipped terminals and connections. Both the supplied autoinstall control programs, DFHZATDX and DFHZATDY, provide function to install shipped definitions of remote terminals and connections. You can therefore base your customized control program on either DFHZATDX or DFHZATDY.

+ Just as you can use an autoinstall user program in a terminal-owning region (TOR) to control the automatic installation of *local* terminals and connections, so you can use a similar program in an application-owning region (AOR) to control the installation of shipped terminals and connections. (Bear in mind when reading this chapter that it assumes that your user program is installed in an AOR—or in a combined AOR/TOR—rather than in a TOR.)

+ Installing shipped terminals and connections

+ In releases of CICS before 4.1, the terminal identifiers (TERMIDs) of shippable terminals had to be unique in the transaction routing network. That is, you could not ship a terminal definition to an AOR on which a remote terminal of the same name was already installed. This restriction no longer applies. Because your autoinstall control program is invoked for shipped terminals and connections, you can use it to reset the TERMINAL (or CONNECTION) attribute of a shipped definition to an **alias**, thereby avoiding conflicts with names of remote terminals and connections already installed in the AOR. There is no need to reset the REMOTENAME attribute, which remains set to the name by which the terminal is known in the TOR; and autoinstall model names are not applicable to shipped definitions.

+ **Note:** If the name of a shipped definition clashes with the name of a *local* terminal or connection installed in the AOR, the install is rejected, and the autoinstall control program is not invoked.

+ For more information about using aliases on remote definitions, see the *CICS/ESA Intercommunication Guide*.

+ CICS-generated aliases

+ The autoinstall control program is invoked once for each shipped terminal or connection definition to be installed.

+ If CICS detects that the name on a shipped definition clashes with the name of a remote terminal or connection already installed in the AOR, it generates an alias TERMID and passes it to the control program in field SELECTED_SHIPPED_TERMID of the communications area.

the autoinstall control program for shipped terminals

+ If CICS detects that there is no clash of names, it passes in
+ SELECTED_SHIPPED_TERMID the name by which the terminal or connection is
+ known in the TOR—that is, the value of the TERMINAL or CONNECTION attribute
+ on the shipped definition.

+ Your control program can accept the passed TERMID, change it, or reject the
+ installation of the shipped definition.

+ CICS-generated aliases consist of a 1-character prefix and a 3-character suffix.
+ The prefix is always '{'. The suffix can have the values 'AAA' through '999'.
+ That is, each character in the suffix can have the value 'A' through 'Z' or '0'
+ through '9'. The first suffix generated by CICS has the value 'AAA'. This is
+ followed by 'AAB', 'AAC', ... 'AAZ', 'AA0', 'AA1', and so on, up to '999'.

+ Each time that it needs to create an alias, CICS generates a 3-character suffix that
+ it has not recorded as being in use. If your autoinstall control program overrides a
+ CICS-generated TERMID, CICS does not record the suffix as being in use, and
+ supplies the same suffix for the next alias.

+ Resetting the terminal identifier

+ You need to think about the algorithm by which your control program allocates alias
+ TERMIDs.

+ You must consider the consequences of a definition being deleted by the
+ CICS/ESA 4.1 timeout delete mechanism, and subsequently being re-shipped and
+ re-installed. You must decide whether your autoinstall program should allocate the
+ *same* TERMID as before (which implies a file mapping the name by which the
+ terminal is known in the TOR to the alias allocated by the AOR), or whether
+ allocation of a different TERMID is acceptable—in which case you could use the
+ default aliases generated by CICS. This decision may depend on several factors.
+ For example:

+ • How your application programs allocate temporary storage queue names. If
+ they derive them from the TERMID (so as to associate the queue with a
+ particular end-user), problems of data mismatch could occur if the queue is not
+ emptied by transaction end (possibly due to a failure), and TERMIDs are not
+ allocated to the same terminals consistently.

+ The best solution is for your application programs always to check before
+ creating a temporary storage queue whether a queue of the same name
+ already exists, and, if so, to delete it. This dispenses with the need for your
+ autoinstall program to allocate TERMIDs consistently.

+ However, if your application programs do not already implement this check, it
+ may not be possible to correct them all. In this case, your autoinstall program
+ may need to use a mapping file, as described above.

+ • Whether your application programs record TERMIDs for later use. For
+ example, an application might issue an EXEC CICS START TERMID
+ command, with a time interval after which the transaction is to be initiated
+ against the named terminal. If, during the delay interval, the terminal definition
+ is deleted, re-shipped, and re-installed with a different local TERMID, the
+ started transaction could fail because the TERMID no longer exists.

+ If your application programs record TERMIDs in this way, your autoinstall
+ program may need to use a mapping file.

+ **Example:** Assume that you have two terminal-owning regions, TORA and TORB,
+ and that they use the same set of terminal identifiers, T001 through T500. TORA
+ and TORB route transactions to the same application-owning region, AOR1. To
+ prevent naming conflicts when terminals are shipped to AOR1, your control
+ program in AOR1 could:

- + • Accept the TERMIDs allocated by TORA. That is, leave the TERMINAL
+ attribute of the remote definition set to the same as the REMOTENAME
+ attribute.
- + • Create aliases for the TERMIDs allocated by TORB. That is, reset the
+ TERMINAL attribute of the remote definition, using a mapping file as described
+ above. For example, TERMIDs of T001 through T500 could be mapped to
+ aliases of A001 through A500.

+ This solution allows two TORs using the same set of TERMIDs to access the same
+ AOR. However, even though the aliases created in the AOR are mapped
+ consistently to TERMIDs in the TOR, the solution does not *guarantee* that data
+ mismatch problems cannot occur if terminals are re-shipped. This is because it
+ relies on TERMIDs being allocated consistently *in the TOR*—that is, on specific
+ TERMIDs always being assigned to the same physical devices.

+ **Note:** Your control program could use the correlation identifier contained in each
+ terminal and connection definition to check whether a definition has been
+ re-installed in the TOR—see the description of the
+ INSTALL_SHIPPED_CORRID_PTR parameter on page 463.

+ A better solution might be to map the terminal alias in the AOR to the **netname** of
+ the terminal. This would at least guarantee that a specific alias always relates to
+ the same physical device. But it would still require TERMIDs for which aliases are
+ *not* created to be consistently allocated in the TOR.

+ The autoinstall control program at INSTALL

+ The autoinstall control program is invoked at INSTALL for:

- + • Local VTAM terminals
- + • Local APPC single-session connections initiated by a CINIT
- + • Local APPC parallel-session connections initiated by a BIND
- + • Local APPC single-session connections initiated by a BIND
- + • Remote shipped terminals and connections.

+ On each invocation, CICS passes a parameter list to the control program by means
+ of a communication area addressed by DFHEICAP. The parameter list passed at
+ INSTALL of local terminals and APPC single-session connections initiated by CINIT
+ is described in “The communication area at INSTALL for terminals” on page 434.
+ The parameter list passed at INSTALL of local APPC connections initiated by BIND
+ requests is described in “The communication area at INSTALL for APPC
+ connections” on page 454. This section describes only INSTALL of shipped
+ terminals and connections.

+ **The communications area at INSTALL for shipped terminals**

+ The communications area is mapped by the DSECT for the assembler version of
 + DFHZATDX, which is supplied in CICS410.SDFHMAC.

```

+ *-----*
+ * Remote install parameter list - Shipped definition Functions 7 & 9 *
+ *-----*
+ INSTALL_SHIPPED_COMMAREA      DSECT          Install Parameter List
+ *
+ INSTALL_SHIPPED_STANDARD      DS  F          Standard field
+                                ORG INSTALL_SHIPPED_STANDARD
+ INSTALL_SHIPPED_EXIT_FUNCTION DS  XL1         Install type
+ INSTALL_SHIPPED_TERM          EQU X'F7'       Install terminal
+ INSTALL_SHIPPED_RSE           EQU X'F8'       Install remote system entry
+ INSTALL_SHIPPED_EXIT_COMPONENT DS  CL2         Component ID 'ZC'
+ INSTALL_SHIPPED_CLASH        DS  CL1         Install clash Y/N
+                                ORG ,
+ INSTALL_SHIPPED_NETNAME_PTR   DS  A          Pointer to netname
+ INSTALL_SHIPPED_SELECTED_PTR  DS  A          Pointer to return fields
+ INSTALL_SHIPPED_TERMID_PTR    DS  A          Pointer to incoming TERMID
+ INSTALL_SHIPPED_APPLID_PTR    DS  A          Pointer to applid of TOR
+ INSTALL_SHIPPED_SYSID_PTR     DS  A          Pointer to sysid
+ INSTALL_SHIPPED_CORRID_PTR    DS  A          Pointer to correlation ID
+ INSTALL_SHIPPED_SELECTED_PARMS DSECT ,
+                                DS  CL8         Reserved
+ SELECTED_SHIPPED_TERMID      DS  CL4         Selected TERMID
+ SELECTED_SHIPPED_RETURN_CODE  DS  CL1         Selected return code
+ RETURN_OK                    EQU X'00'       Accept request
+ REJECT                       EQU X'01'       Reject request
+ *
  
```

+ *Figure 35. Autoinstall control program's communications area at INSTALL. For shipped terminals and connections.*

+ **INSTALL_SHIPPED_STANDARD**

+ A fullword input field containing the following information:

+ **INSTALL_SHIPPED_EXIT_FUNCTION**

+ A 1-byte field that indicates the type of resource being installed. For
 + install of remote terminals and connections the equated values are:

- + **INSTALL_SHIPPED_TERM (X'F7')** A shipped terminal
- + **INSTALL_SHIPPED_RSE (X'F8')** A shipped connection (remote
 + system entry).

+ **INSTALL_SHIPPED_EXIT_COMPONENT**

+ A 2-byte component code, which is set to 'ZC'.

+ **INSTALL_SHIPPED_CLASH**

+ A 1-character input field that indicates whether the TERMID of the shipped
 + definition is already in use in the AOR.

- + **Y** The name by which the terminal or connection is known in the TOR
 + (the value of the TERMINAL or CONNECTION attribute on the
 + shipped definition) is already in use in the AOR to identify an installed
 + remote terminal or connection.
- + **N** The name by which the terminal or connection is known in the TOR is
 + not in use in the AOR to identify a remote terminal or connection.

+ **INSTALLED_SHIPPED_NETNAME_PTR**
 + A fullword pointer to an 8-character input field containing the netname of the
 + terminal or connection to be installed.

+ **INSTALL_SHIPPED_SELECTED_PTR**
 + A fullword pointer to the return fields. The output fields, for use by your
 + program, are:

+ **SELECTED_SHIPPED_TERMID**
 + A 4-character field used to specify the name by which the remote
 + terminal or connection is to be known to this system. If the name is
 + less than 4 characters long, it must be padded with trailing blanks.
 + For a list of the characters you can use in terminal names, see the
 + *CICS/ESA Resource Definition Guide*.

+ On invocation, if **INSTALL_SHIPPED_CLASH** is set to 'N' (indicating
 + no conflict of terminal names), **SELECTED_SHIPPED_TERMID**
 + contains the same value as the field pointed to by
 + **INSTALL_SHIPPED_TERMID_PTR** (the value of the **TERMINAL** or
 + **CONNECTION** attribute on the shipped definition). If
 + **INSTALL_SHIPPED_CLASH** is set to 'Y',
 + **SELECTED_SHIPPED_TERMID** contains a CICS-generated alias.

+ Your user program can use this field to override a CICS-generated
 + alias. For advice on choosing terminal and connection names, see
 + "Resetting the terminal identifier" on page 460.

+ **SELECTED_SHIPPED_RETURN_CODE**
 + The 1-character return code field. The equated values are:

#	RETURN_OK (X'00')	Install the remote terminal or connection. Your user program must return this value if the resource is to be autoinstalled.
#		
#	REJECT (X'01')	Do not install the remote terminal or connection. This is the default value.
#		

+ **INSTALL_SHIPPED_TERMID_PTR**
 + A fullword pointer to a 4-character input field containing the name by which the
 + terminal or connection is known in the TOR. (This is the value of the
 + **TERMINAL** or **CONNECTION** attribute on the shipped definition.)

+ **INSTALL_SHIPPED_APPLID_PTR**
 + A fullword pointer to an 8-character input field containing the netname (applid)
 + of the TOR.

+ **INSTALL_SHIPPED_SYSID_PTR**
 + A fullword pointer to a 4-character input field containing the name (sysid) of the
 + connection to the TOR.

+ **INSTALL_SHIPPED_CORRID_PTR**
 + A fullword pointer to an 8-character input field containing the shipped
 + definition's *correlation identifier*. A correlation identifier is a unique "instance
 + token" that is created when a CICS/ESA 4.1 or later terminal or connection
 + definition is installed, and stored within the definition. Thus, if the definition is
 + shipped to another region, the value of the token is shipped too. The
 + correlation ID is used by CICS during attach processing, to check whether
 + existing shipped definitions in an AOR are up-to-date, or whether they need to
 + be deleted and reshipped because the terminal has been re-installed in the

the autoinstall control program for shipped terminals

+ TOR. For further information about instance tokens, see the *CICS/ESA Intercommunication Guide*.
+
+ If your control program maps TOR-allocated TERMIIDs to the aliases that it
+ assigns in the AOR, by recording correlation IDs it could check whether a
+ terminal has been re-installed in the TOR. If the terminal *has* been re-installed,
+ it is possible that the TOR-allocated TERMIID relates to a *different* physical
+ device from that last installed under this TERMIID.

+ The autoinstall control program at DELETE

+ The autoinstall control program is reinvoked when an autoinstalled resource is
+ deleted. (The resources that can be autoinstalled are listed under “The autoinstall
+ control program at INSTALL” on page 461.) Invoking the user program at DELETE
+ enables you to reverse the processes carried out at INSTALL.

+ The parameter list passed to your user program at DELETE of local terminals is
+ described on page 441. This section describes only DELETE of shipped terminals
+ and connections.

+ Shipped terminal and connection definitions are deleted by the CICS/ESA 4.1
+ timeout delete mechanism. For details of the timeout delete mechanism, see the
+ *CICS/ESA Intercommunication Guide*.

+ Figure 36 shows the communications area passed to the autoinstall user program
+ at DELETE.

DELETE_SHIPPED_COMMAREA	DSECT ,	Delete parameter list
DELETE_SHIPPED_STANDARD	DS F	Standard field
DELETE_SHIPPED_EXIT_FUNCTION	DS XL1	Delete type
DELETE_SHIPPED_TERM	EQU X'FA'	Delete terminal
DELETE_SHIPPED_RSE	EQU X'FB'	Delete remote system entry
DELETE_SHIPPED_EXIT_COMPONENT	DS CL2	Component ID 'ZC'
	DS CL1	Reserved
DELETE_SHIPPED_TERMID	DS CL4	TERMIID in TOR
DELETE_SHIPPED_APPLID	DS CL8	Applid of TOR
DELETE_SHIPPED_LTERMID	DS CL4	TERMIID in AOR

+ *Figure 36. Autoinstall control program's communications area at DELETE. For shipped
+ terminals and connections.*

+ At DELETE, all fields in the communications area are input only. Fields not listed
+ below are as described for INSTALL.

DELETE_SHIPPED_EXIT_FUNCTION

+ A 1-byte field that indicates the type of resource being deleted. The equated
+ values are:

DELETE_SHIPPED_TERM (X'FA')	A shipped terminal
DELETE_SHIPPED_RSE (X'FB')	A shipped connection (remote system entry).

+ **Note:** A value of X'F1' represents the deletion of a local terminal, or an
+ APPC single-session device that was autoinstalled via a CINIT request—see
+ page 441.

- + **DELETE_SHIPPED_TERMID**
+ A 4-character field containing the identifier (TERMID) of the terminal or
+ connection in the TOR.
- + **DELETE_SHIPPED_APPLID**
+ An 8-character field containing the netname (applid) of the TOR.
- + **DELETE_SHIPPED_LTERMID**
+ A 4-character field containing the name by which the terminal or connection is
+ known in the AOR. This may or may not be the same as
+ DELETE_SHIPPED_TERMID, depending on whether an alias has been used in
+ the AOR.

+ Default actions of the sample programs

— APAR PN87772 —

Documentation for APAR PN 87772 added 18th December 1996.

+ When DFHZATDX or DFHZATDY is invoked at INSTALL of a shipped terminal or
+ connection, it:

- + 1. Updates, if necessary, the SELECTED_SHIPPED_TERMID field, so that it
+ contains the name by which the terminal or connection is known in the TOR.

+ **Notes:**

- + a. If CICS detected a conflict with a currently-installed remote TERMID, on
+ invocation of the sample programs SELECTED_SHIPPED_TERMID
+ contains a CICS-generated alias. The sample programs overwrite this
+ value.
 - + b. If CICS detected no conflict with a currently-installed remote TERMID, on
+ invocation of the sample programs SELECTED_SHIPPED_TERMID
+ contains the value of the TERMINAL attribute on the shipped definition (the
+ value pointed to by INSTALL_SHIPPED_TERMID_PTR). The sample
+ programs accept this value.
- + 2. Permits the remote definition to be installed by setting the return code field to
+ RETURN_OK, and returning.

+ When DFHZATDX or DFHZATDY is invoked at DELETE of a shipped terminal or
+ connection, it takes no action and returns.

the autoinstall control program for shipped terminals

Chapter 13. Writing a program to control autoinstall of programs

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

This chapter describes the user-replaceable program that controls the automatic installation of programs, mapsets, and partitionsets.

Note: In this chapter, the term “program autoinstall” is used to mean autoinstall of all three program types (program, mapset, and partitionset) unless otherwise specified.

The chapter is divided into the following sections:

1. “**Preliminary considerations**”
2. “**Benefits of autoinstall**” on page 469
3. “**Requirements for autoinstall**” on page 469
4. “**The autoinstall control program at INSTALL**” on page 470
5. “**The sample autoinstall control program for programs, DFHPGADX**” on page 472.

Preliminary considerations

Before the introduction of autoinstall, all resources had to be defined individually to CICS before they could be used. With the introduction of autoinstall for terminals in CICS/OS/VS 1.7, VTAM terminals could be defined dynamically on their first usage, thereby saving on storage for terminal entries, and on time spent creating the definitions.

As well as terminals and APPC connections, you can autoinstall:

- User programs
- Mapsets
- Partitionsets.

If the autoinstall program function is enabled, and an implicit or explicit load request is issued for a previously undefined program, mapset, or partitionset, CICS dynamically creates a definition, and installs and catalogs it, as appropriate. An implicit or explicit load occurs when:

- CICS starts a transaction.
- An application program issues one of the following commands:

```
EXEC CICS LINK
EXEC CICS XCTL
EXEC CICS LOAD
EXEC CICS ENABLE (for a global user exit, or task-related user exit,
program)
EXEC CICS RECEIVE or SEND MAP
EXEC CICS SEND PARTNSET
```

the autoinstall control program for programs

EXEC CICS RECEIVE PARTN.

- A program abend occurs, and CICS transfers control to the program named on an EXEC CICS HANDLE ABEND command.
- CICS calls any user-replaceable program other than the program or terminal autoinstall program.
- A program is named in the PLTPI or PLTSD list.

Autoinstall model definitions

Like autoinstall for terminals, program autoinstall uses model definitions, together with a user-replaceable control program, to create explicit definitions for resources that need to be autoinstalled. The purpose of a model is to provide CICS with a definition that can be used for all programs with the same properties. CICS calls the autoinstall control program with a parameter list that includes the name of a CICS-supplied, default model definition appropriate to the program type (program, mapset, or partitionset). Your autoinstall control program can accept the default model, or specify another (any installed program definition can be used as a model). It can also specify explicitly any properties that are unique to a program, thus overriding those specified on the model definition.

On return from the control program, CICS creates a resource definition from the model and properties returned in the parameter list.

Note that CICS does not call your control program for CICS programs, mapsets, or partitionsets—that is, for any objects that begin with the letters “DFH”.

Autoinstall processing of mapsets

Table 112 shows the differences in mapset processing between CICS regions with program autoinstall active and inactive.

Program autoinstall INACTIVE	Program autoinstall ACTIVE
CSD definition is required. CICS attempts to load a referenced mapset with a suffix. If this fails, CICS tries an un-suffixed version. If that is unsuccessful, abend APCT is issued.	CSD definition is not required. Using autoinstall, CICS attempts to load the referenced suffixed mapset or partitionset, then the un-suffixed one. (In each case, a definition is autoinstalled.) The transaction requesting the resource abends only if no version of the resource exists in the library, either suffixed or un-suffixed. If the suffixed mapset was not found in the library, the definition is marked 'not loadable'.

System autoinstall

Some programs are autoinstalled automatically (if they have not been statically defined) by the CICS **system autoinstall** function, which does not require model definitions or the support of the autoinstall control program. Programs in this category include:

- First phase program list table post initialization (PLTPI) programs (that is, PLTPI programs that are defined *before* the PLT table delimiter DFHDELIM).
- Second phase program list table shutdown (PLTSD) programs (that is, PLTSD programs that are defined *after* the PLT table delimiter DFHDELIM).

Note: PLTPI programs that are defined *after* DFHDELIM, and PLTSD programs that are defined *before* DFHDELIM, are treated like any other user programs—they are eligible for program autoinstall.

Benefits of autoinstall

Program autoinstall reduces system administration, virtual storage usage, and, potentially, restart times.

Reduced system administration costs

Without autoinstall, you have to define all new programs, mapsets, and partitionsets to CICS before they can be used. Autoinstall eliminates this requirement, enabling these resources to be used without prior definition. Furthermore, the need to maintain predefined definitions also disappears, leading to a significant saving in system administration effort.

Saving in virtual storage

There is a saving in virtual storage within the CICS address space, as the definitions of autoinstalled resources do not occupy table space until they are generated.

Faster startup times

Warm and emergency starts: If you are using program autoinstall *with cataloging*, restart times are similar to those of restarting a CICS region that is not using program autoinstall. This is because, in both cases, resource definitions are reinstalled from the catalog during the restart. The definitions after the restart are those that existed before the system was terminated.

If you are using autoinstall *without cataloging*, CICS restart times are improved because CICS does **not** install definitions from the CICS global catalog. Instead, definitions are autoinstalled as required whenever programs, mapsets, and partitionsets are referenced following the restart.

See the *CICS/ESA Recovery and Restart Guide* for information on cataloging.

Cold starts: Startup times are faster than for a region that does not use program autoinstall, because program definitions are installed singly, as required, rather than all together at startup.

Requirements for autoinstall

To use autoinstall with programs, mapsets, and partitionsets, you must:

1. Write a customized version of the autoinstall control program for programs, DFHPGADX (unless the supplied version is entirely suitable for your purposes).

autoinstall control program at INSTALL

2. Specify the name of your control program on the PGAEXIT system initialization parameter (the default name is DFHPGADX), or on a SET SYSTEM PROGAUTOEXIT command.
3. Make program autoinstall active by specifying 'ACTIVE' on the PGAIPGM system initialization parameter (or by issuing a SET SYSTEM PROGAUTOINST(AUTOACTIVE) command).
4. Specify whether you want autoinstalled program definitions to be recorded on the CICS global catalog, on the PGAICTLG system initialization parameter (or on a SET SYSTEM PROGAUTOCTLG command).
5. Include the DFHPGAIP resource definition group in your CICS startup grouplist. DFHPGAIP (which is already included in the CICS-supplied startup list, DFHLIST) contains the default program, mapset, and partitionset model definitions passed to the autoinstall control program, and a definition of DFHPGADX (that you may need to amend).
6. Create any additional program, mapset, and partitionset model definitions that you need, and add this group to your startup grouplist.
7. If you want to log messages associated with program autoinstall, define the CSPL transient data (TD) queue.

For information about coding system initialization parameters, see the *CICS/ESA System Definition Guide*. For information about defining programs, mapsets, partitionsets, and TD queues, see the *CICS/ESA Resource Definition Guide*.

The autoinstall control program at INSTALL

On invocation, CICS passes a parameter list to the autoinstall control program by means of a communication area addressed by DFHEICAP. The communications area is mapped by a copybook that is supplied in each of the languages supported by CICS.

The assembler form of the parameter list is as follows:

PGAC_PROGRAM

passes the 8-byte name of the object to be autoinstalled. This is an input-only field, which your user-replaceable program must not alter.

PGAC_MODULE_TYPE

passes a 1-byte indicator of the type of object to be installed. The equated values are:

PGAC_TYPE_PROGRAM	A program
PGAC_TYPE_MAPSET	A mapset
PGAC_TYPE_PARTITIONSET	A partitionset.

This is an input-only field, which your user-replaceable program must not alter.

PGAC_MODEL_NAME

allows your control program to specify the 8-byte autoinstall model name to be used. If you do not set this field, CICS uses the default model name for the type of object:

DFHPGAPG	For a program
DFHPGAMP	For a mapset

DFHPGAPT For a partitionset.

PGAC_LANGUAGE

allows your control program to specify, in a 1-byte field, the language of the program to be autoinstalled. The equated values are:

PGAC_ASSEMBLER	Assembler
PGAC_COBOL	COBOL
PGAC_C370	C
PGAC_LE370	Language Environment/370
PGAC_PLI	PL/I.

If you do not set this field, the autoinstall routine uses the language defined in the model, if one is specified. However, when control is passed to the program, CICS determines the language from the program itself, and overrides any specification provided.

You should not need to specify the language of executable programs that have been translated using the EXEC CICS translator before compiling.

PGAC_CEDF_STATUS

allows you to specify, in a 1-byte field, the execution diagnostic facility (EDF) status of the program to be autoinstalled. The equated values are:

PGAC_CEDF_YES	EDF can be used with this program.
PGAC_CEDF_NO	EDF cannot be used with this program.

PGAC_DATA_LOCATION

allows you to specify, in a 1-byte field, the data location for task-lifetime storage. The equated values are:

PGAC_LOCATION_BELOW
Task-lifetime storage must be located below 16MB.

PGAC_LOCATION_ANY
Task-lifetime storage can be below or above 16MB.

PGAC_EXECUTION_KEY

allows you to specify, in a 1-byte field, the execution key for the program. The equated values are:

PGAC_CICS_KEY	The program is to execute in CICS key.
PGAC_USER_KEY	The program is to execute in user key.

PGAC_LOAD_ATTRIBUTE

allows you to specify, in a 1-byte field, the load attributes for the object. The equated values are:

PGAC_RELOAD CICS is to load a fresh copy of the object for each request.

PGAC_RESIDENT CICS is to make the object permanently resident.

PGAC_TRANSIENT The storage for this object is to be released whenever the use count reaches zero.

PGAC_REUSABLE CICS can use any copy of the object currently in storage.

PGAC_USE_LPA_COPY

allows you to specify, in a 1-byte field, whether CICS is to use an LPA-resident copy of the program. The equated values are:

PGAC_LPA_YES CICS is to use a copy from the LPA.

PGAC_LPA_NO CICS is to load a private copy from its own DFHRPL library concatenation.

PGAC_EXECUTION_SET

allows you to specify, in a 1-byte field, whether or not the program is restricted to using the distributed program link (DPL) subset of the CICS API. The equated values are:

PGAC_DPLSUBSET The program is to be restricted to the DPL subset of the EXEC CICS API.

PGAC_FULLAPI The program can use the full API.

PGAC_REMOTE_SYSID

allows you to specify, in a 4-byte field, the name of the remote system where the program is to execute. CICS function ships any request for this program to the specified remote CICS.

PGAC_REMOTE_PROGID

allows you to specify, in an 8-byte field, the name by which the program is known in the remote CICS region. For a remote program, the remote name defaults to the local name if you set this field to blank.

PGAC_REMOTE_TRANSID

allows you to specify, in a 4-byte field, the name of the CICS mirror transaction under which the program, if remote, is to run. By default, this is set to the name of the CICS mirror transaction, CSML.

PGAC_RETURN_CODE

allows you to specify, in a 1-byte field, the autoinstall control program's return code to CICS. The equated values are:

PGAC_RETURN_OK

Install the program definition using the values returned in the communications area parameter list.

PGAC_RETURN_DONT_DEFINE_PROGRAM

Do not define the program.

The sample autoinstall control program for programs, DFHPGADX

The CICS-supplied default autoinstall program is an assembler-language command-level program, named DFHPGADX. The source of the default program is provided in COBOL, PL/I, and C, as well as in assembler language. The names of the supplied programs and their associated copy books, and the CICS410 libraries in which they can be found, are summarized in Table 113 on page 473.

Table 113. Sample programs and copy books for program autoinstall

Language	Member name	Library
Executable file:		
Assembler only	DFHPGADX	SDFHLOAD
Program source:		
Assembler	DFHPGADX	SDFHSAMP
COBOL	DFHPGAOX	SDFHSAMP
PL/I	DFHPGALX	SDFHSAMP
C/370	DFHPGAHX	SDFHSAMP
Copy books:		
Assembler	DFHPGACD	SDFHMAC
COBOL	DFHPGACO	SDFHCOB
PL/I	DFHPGACL	SDFHPL1
C/370	DFHPGACH	SDFHC370

Customizing the sample program

You can write your autoinstall control program in any of the languages supported by CICS, with full access to the CICS application and system programming interfaces.

If you customize the supplied control program, or write your own version, you should note the following:

- **Input:**
The first two fields of the parameter list are input-only fields and should not be altered by your program.
- **Output:**
The remaining fields on the parameter list are input/output or output only fields, which you can use to specify attributes that override those of the model definition.
- Some of the output fields in the parameter list are not applicable to mapsets or partitionsets. CICS ignores any parameters you specify that are not applicable to the type of object being installed.
- Any attributes you return to CICS in the parameter list are used to modify the model definition, and CICS installs the modified definition. Once installed, the definition can be modified normally using the EXEC CICS SET PROGRAM or CEMT SET PROGRAM commands.
- If you modify your control program, you can make the new version available by using the EXEC CICS SET PROGRAM NEWCOPY or CEMT SET PROGRAM NEWCOPY command.
- You can discard definitions after they have been installed; they are reinstalled when next referenced.
- You must ensure that the parameters you return to CICS are valid, and consistent with other system attributes in your CICS region. For example:
 - Do not return PGAC_LPA_YES on the PGAC_USE_LPA_COPY parameter if CICS is running with the system initialization parameter LPA=NO.

sample autoinstall programs

- Do not return PGAC_USER_KEY (which is the default) on the PGAC_EXECUTION_KEY parameter if the task for which your control program is called is running with CICS-key task-lifetime storage.

You can determine the storage key for the task by testing the TASKDATAKEY option in its transaction definition by means of the following EXEC CICS commands:

```
EXEC CICS ADDRESS EIB  
EXEC CICS INQUIRE TRANSACTION(eibtrans) TASKDATAKEY(...)
```

#

Important

#

When creating an autoinstalled program definition, CICS ignores the program language specified on the model program definition. CICS determines the language from the load module itself, when the autoinstalled program is invoked.

#

However, CICS does *not* deduce characteristics other than language from the load module. These other program characteristics must be explicitly defined by the autoinstall control program or by RDO. If your programs have varying characteristics (varying AMODE or DATALOCATION requirements, for example), you must be able to distinguish between the various types when using autoinstall. You could do this by keeping a list of exceptions to the default characteristics, and coding your autoinstall control program to refer to this list; or you might decide to install explicit RDO definitions of the exceptions.

Resource definition

The autoinstall control program cannot itself be autoinstalled, nor can any program it references. You must define a program resource definition in the CSD for the control program and for any other programs it references. You must also ensure these definitions are installed in the CICS region during startup by including the group containing the definitions in your startup grouplist. If you specify an invalid name for the control program, CICS disables the program, thus disabling the program autoinstall function.

The following program resource definitions are supplied by CICS for the autoinstall control program; the default is the assembler version, DFHPGADX. If these definitions are not suitable for your use, you can create your own, using RDO or the DFHCSDUP utility.

- Default autoinstall control program definition for DFHPGADX. This defines the assembler version, and its status is set to ENABLED:

GROUP(DFHPGAIP)	PROGRAM(DFHPGADX)	
DESCRIPTION(Assembler definition for program autoinstall exit)		
LANGUAGE(ASSEMBLER)	EXECKEY(CICS)	EXECUTIONSET(FULLAPI)
RELOAD(NO)	RESIDENT(NO)	USAGE(NORMAL)
STATUS(ENABLED)	CEDF(NO)	DATALOCATION(ANY)

- Autoinstall control program definition for DFHPGAOX. This defines the CICS-supplied COBOL version, and its status is set to DISABLED:

```

GROUP(DFHPGAIP)      PROGRAM(DFHPGA0X)
DESCRIPTION(COBOL definition for program autoinstall exit)
LANGUAGE(COBOL)      EXECKEY(CICS)      EXECUTIONSET(FULLAPI)
RELOAD(NO)           RESIDENT(NO)      USAGE(NORMAL)
STATUS(DISABLED)    CEDF(NO)           DATALOCATION(ANY)
    
```

- Autoinstall control program definition for DFHPGAHX. This defines the CICS-supplied C/370 version, and its status is set to DISABLED:

```

GROUP(DFHPGAIP)      PROGRAM(DFHPGAHX)
DESCRIPTION(C definition for program autoinstall exit)
LANGUAGE(C)           EXECKEY(CICS)      EXECUTIONSET(FULLAPI)
RELOAD(NO)           RESIDENT(NO)      USAGE(NORMAL)
STATUS(DISABLED)    CEDF(NO)           DATALOCATION(ANY)
    
```

- Autoinstall control program definition for DFHPGALX. This defines the CICS-supplied PL/I version, and its status is set to DISABLED:

```

GROUP(DFHPGAIP)      PROGRAM(DFHPGALX)
DESCRIPTION(PL/I definition for program autoinstall exit)
LANGUAGE(PLI)         EXECKEY(CICS)      EXECUTIONSET(FULLAPI)
RELOAD(NO)           RESIDENT(NO)      USAGE(NORMAL)
STATUS(DISABLED)    CEDF(NO)           DATALOCATION(ANY)
    
```

Testing and debugging your program

You can use the CICS execution diagnostic facility (EDF) to help you test your autoinstall control program. However, EDF is inhibited for programs with names that begin with the letters DFH; so to use EDF you must name your program something other than one of the default names.

sample autoinstall programs

Chapter 14. Writing a dynamic transaction routing program

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

When you define transactions to CICS, you can describe them as “remote” or “local”. Local transactions are always executed in the terminal-owning region; remote transactions can be routed to other regions connected to the terminal-owning region by MRO links, or to other systems that are connected by APPC (LUTYPE6.2) ISC links.

You can select both the system to which the transaction is to be routed and the transaction’s remote name dynamically, rather than when the transaction is defined to CICS. To do this you must use a **dynamic transaction routing program**. The CICS-supplied default routing program is called DFHDYP. Its source-level code is supplied in assembler-language, COBOL, PL/I, and C versions. You can write your own program in any of these languages, using the default program as a model.

This chapter describes the default program, and tells you how to write your own version. If you need more guidance information about transaction routing, you should first refer to the *CICS/ESA Intercommunication Guide*.

Note: If you use the CICSplex System Manager (CICSplex SM) product to manage your CICSplex, you may not need to write a dynamic transaction routing program. CICSplex SM provides a fully-functioning dynamic transaction routing program that supports workload balancing and workload separation. All you have to do is to tell CICSplex SM, through its user interface, which TORs and AORs in the CICSplex can participate in dynamic transaction routing, and define any transaction affinities that govern the AORs to which particular transactions must be routed. For introductory information about CICSplex SM, see the *CICSplex SM Concepts and Planning* manual.

Dynamic transactions

When you want to route transactions dynamically, you must define them with the value DYNAMIC(YES). (The default value is DYNAMIC(NO).) You must also supply values for both the remote and the local options. This allows CICS to select the appropriate values when the transaction is routed, and to ignore those values that are not needed. For information about defining transactions for dynamic transaction routing, see the *CICS/ESA Intercommunication Guide*.

The dynamic transaction routing program, DFHDYP

CICS invokes the user-replaceable dynamic transaction routing program as follows:

- When a transaction defined with the value DYNAMIC(YES) is initiated.
- When a transaction definition is not found and CICS uses the special transaction defined on the DTRTRAN system initialization parameter. (For more information about DTRTRAN, see the *CICS/ESA System Definition Guide*.)

the dynamic transaction routing program

- Before routing a remote, terminal-oriented, transaction initiated by ATI.
- If an error occurs in route selection.
- At the end of a routed transaction, if the initial invocation requests reinvocation at termination.
- If a routed transaction abends, if the initial invocation requests re-invocation at termination.

Information passed to the dynamic transaction routing program

The CICS relay program, DFHAPRT, passes information to the dynamic transaction routing program by means of a communications area. The communications area contains fields that are mapped by the DSECT DFHDYPDS, and is described in detail in “Parameters passed to the dynamic transaction routing program” on page 484. Some of the data passed to the dynamic transaction routing program in the communications area is:

- The SYSID of the remote CICS region specified when the transaction was installed
- The netname of the remote CICS region
- The name of the remote transaction
- The dispatch priority (MRO only) of the remote transaction
- Whether or not the request is to be queued if no sessions are immediately available to the remote CICS region
- The address of the remote transaction’s communications area
- The address of a copy of the transaction’s terminal input/output area (TIOA)
- A task-local user data area.

The communications area DSECT contains comments to describe the information passed.

The dynamic transaction routing program can accept these values, or change them, or tell CICS not to continue routing the transaction. The values used depend on the function being performed; that is, some values may be ignored.

The information passed to the dynamic transaction routing program indicates whether the transaction is being routed dynamically or statically (for remote ATI requests).

If the transaction is being routed dynamically, the dynamic transaction routing program can change the SYSID or netname to determine where the transaction is to run.

If the transaction was started by ATI, the dynamic transaction routing program is called only to notify it of where the transaction is going to run. In this case, the dynamic transaction routing program cannot change the remote system name, and any changes to the SYSID or NETNAME fields in the communications area are ignored.

For transactions that are run remotely, either because they are defined as remote or because they are dynamically routed to a remote CICS region, CICS monitoring is informed of the SYSID of the remote CICS region. For transactions that the

dynamic transaction routing program routes locally, the monitoring field is set to nulls.

Changing the target CICS region

The communications area passed to the dynamic transaction routing program initially contains the SYSID and netname of the default CICS region to which the transaction is to be routed. These are derived from the value of the REMOTESYSTEM option of the installed transaction definition. If the transaction definition does not specify a REMOTESYSTEM value, the SYSID and netname passed are those of the local CICS region.

The dynamic transaction routing program can change the SYSID and netname, so that the CICS region to which the transaction is routed is determined as follows:

- The NETNAME and the SYSID are not changed.
CICS tries to route to the SYSID as originally specified in the communications area.
- The NETNAME is not changed, but the SYSID is changed.
CICS updates the communications area with the NETNAME corresponding to the new SYSID, and tries to route to the new SYSID.
- The NETNAME is changed, but the SYSID is not changed.
CICS updates the communications area with a SYSID corresponding to the new NETNAME, and tries to route to the new SYSID.
- The NETNAME is changed **and** the SYSID is changed.
CICS overwrites the communications area with a SYSID corresponding to the new NETNAME, and tries to route to that new SYSID.

If the NETNAME specified is invalid, or cannot be found, SYSIDERR is returned to the dynamic transaction routing program.

Using a common transaction definition in the TOR

The recommended method is to use a single, common definition for all remote transactions that are to be dynamically routed. The name of the common definition is specified on the DTRTRAN system initialization parameter. You can use the REMOTESYSTEM option of the common definition to specify a default AOR to which transactions are to be routed. For information about defining remote transactions for dynamic transaction routing, see the *CICS/ESA Intercommunication Guide*.

Important

To route a transaction defined by the DTRTRAN definition, your dynamic transaction routing program must set the DYRDTRRJ field of the communications area to 'N' (the default is 'Y'). If you leave DYTDTRRJ set to 'Y', the transaction is rejected.

You can test the DYRDTRXN field to check if the transaction passed to your routing program is defined by the DTRTRAN definition. Figure 37 on page 480 contains skeleton code for routing transactions defined by DTRTRAN.

the dynamic transaction routing program

```
#          if DYRDTRXN='Y' then          /* Is DYP invoked because of DTRTRAN */
|          do                            /* .. Yes                               */
|          Call Find_AOR(sysid)         /* Select the SYSID of the AOR          */
|          if rc=0 then                 /* Is AOR available?                   */
|          do                            /* .. Yes                               */
|          DYRRETC=RETCOD0              /* Set OK Return Code                  */
|          DYRSYSID=sysid               /* Set the sysid                       */
|          DYRDTRRJ='N'                 /* Don't reject DTRTRAN defns         */
|          ...                          /* Set other commarea fields          */
|          end                          /*                                     */
|          else                         /* .. No                               */
|          ...                          /* AOR unavailable logic               */
|          end                          /*                                     */
```

Figure 37. Example pseudocode to route transactions defined by DTRTRAN

Changing the program name

For transactions defined as DYNAMIC, on invocation of the routing program the DYRLPROG field in the communications area contains the name of the initial program associated with the transaction to be routed. If you decide to route the transaction locally, you can use this field to specify an alternative program to be run. For example, if all remote CICS regions are unavailable and the transaction cannot be routed, you may want to run a program in the local CICS terminal-owning region to send an appropriate message to the user.

Telling CICS whether to route or terminate a transaction

If you want a transaction to be routed, whether you have changed any values or not, you return a zero value to CICS in field DYRRETC of the communications area. If you want to terminate the transaction with a message or an abend, you supply a return code of X'8'. A further option (return code X'4') tells CICS to terminate the transaction without issuing a message or abend.

Warning: Setting a return code of X'4' for APPC transaction routing leads to unpredictable results, and should be avoided.

When you return control to CICS with return code zero, CICS first compares the returned SYSID with its own local SYSID:

- If the SYSIDs are the same (or the returned SYSID is blank) CICS executes the transaction locally.
- If the two SYSIDs are not the same, CICS routes the transaction to the remote CICS region, using the remote transaction name.

The dynamic transaction routing program is invoked again if the routed transaction abends and, optionally, when the routed transaction terminates.

If the system is unavailable or unknown

The dynamic transaction routing program is invoked again if the remote system name that you specify on the route selection call is not known or is unavailable. When this happens, you have a choice of actions:

- You can tell CICS not to continue trying to route the transaction, by issuing a return code of '8' in DYRRETC. If the reason for the error is that the system is

unavailable, CICS issues message 'DFHAC2014' or 'DFHAC2029' to the terminal user. If the reason for the error is that the system is unknown, DFHAPRT abends the transaction.

- You can tell CICS to terminate the transaction without issuing a message or abend by placing a return code of '4' in DYRRETC. However, note the above warning about setting return code '4'.
- If the reason for the error is that no sessions are immediately available to the remote system, you can reset field DYRQUEUE to 'Y' (it must previously have been set to 'N'—the request is **not** to be queued—for this error to occur), issue a return code of '0' in DYRRETC, and try to route the transaction again.

If you try to route the transaction again **without** resetting DYRQUEUE to 'Y' (and without changing the sysid), and the system is still unavailable, DFHDYP is reinvoked. If you then choose to set return code '8', CICS terminates the transaction with message 'DFHAC2030'.

- You can change the sysid, and issue a return code of '0' in DYRRETC to try to route the transaction again. Note that if you change the sysid, you may also need to supply a different remote transaction ID. You need to do this if, for example, the transaction has a different remote transaction name on each system.

A count of the times the routing program has been invoked for routing purposes for this transaction is passed in field DYRCOUNT. Use this count to help you decide when to stop trying to route the transaction.

Invoking the dynamic transaction routing program at end of routed transactions

If you want your dynamic transaction routing program to be invoked again when the routed transaction has completed, you must set the DYROPTER field in the communications area to 'Y' before returning control to CICS. You might want to do this, for example, if you are keeping a count of the number of transactions currently executing on a particular CICS region. However, during this reinvocation, the dynamic transaction routing program should update only its own resources, because at this stage the final command to the terminal from the application program in the AOR may be pending, and the dynamic transaction routing program is about to terminate.

Note: If you have set DYROPTER to 'Y', and the routed transaction abends, the dynamic transaction routing program is invoked again to notify it of the abend. You could use this invocation to initiate a user-defined program in response to the transaction abend.

Invoking the dynamic transaction routing program on abend

If the routed transaction abends, the DFHAPRT program in the TOR:

1. Passes back a response to the CICS transaction manager indicating that a transaction abend has occurred
2. If the dynamic transaction routing program requested to be reinvoked at termination of the transaction (by setting DYROPTER to 'Y' when invoked for routing), reinvokes the dynamic transaction routing program
3. Returns to CICS transaction manager.

Modifying the initial terminal data

The dynamic routing program should **not** perform an EXEC CICS RECEIVE or an EXEC CICS GDS RECEIVE command, because this prevents the routed-to transaction from obtaining the initial terminal data.

The CICS relay program, DFHAPRT, places a *copy* of the user's initial terminal input into a separate buffer. This information includes SNA presentation services headers for APPC mapped and unmapped conversations. A pointer to this buffer (DYRBPNTNTR), and its length (DYRBLGTH), are provided in the communication area passed from DFHAPRT to the dynamic routing program.

Because the transaction profile has not been queried at this point, uppercase translation has not been performed on the input data unless UCTRAN(YES) is specified on the TYPETERM definition.

Sometimes you may want to modify the initial data input by the user. (It may be necessary to do this if, for example, you change the ID of the remote transaction, using field DYRTRAN of the communications area.) To modify the input data, your routing program should, when invoked for route selection:

1. Copy the input data pointed to by DYRBPNTNTR into a named variable, of length DYRBLGTH
2. Modify the data in the named variable
3. Use the INPUTMSG option of the EXEC CICS RETURN command to make the modified data available to the application program.

For guidance information about using INPUTMSG on EXEC CICS RETURN commands, see the *CICS/ESA Application Programming Guide*. For programming information about the INPUTMSG option, see the *CICS/ESA Application Programming Reference* manual.

Note: If, after modifying the input data, the dynamic transaction routing program is reinvoked because an error occurs in routing to the selected transaction, it should “remember” that it has modified the original user-input.

Modifying the application's communications area

Sometimes you may want to modify the routed application's communications area. For example, if your routing program changes the ID of the remote transaction, it may also need to change the input communications area passed to the routed application. Field DYRACMAA of the routing program's communications area enables you to do this; it is a pointer to the application's communications area.

Receiving information from a routed transaction

If your dynamic transaction routing program chooses to be reinvoked at the end of a routed transaction, it can obtain information about the transaction by monitoring its output communications area and output TIOA.

Monitoring the output communications area

A routed transaction can pass information back to the dynamic transaction routing program in its output communications area. When invoked at transaction termination, your routing program can examine the output communications area (pointed to by DYRACMAA). The following is an example of how this facility could be used:

You have a CICSplex consisting of sets of functionally-equivalent TORs and AORs, and need to identify any intertransaction affinities that may affect transaction routing. You could use the CICS Transaction Affinities Utility⁶ to do this, but there are some affinities that the utility cannot detect (for example, those created by non-CICS functions). Also, some transactions may sometimes create affinities, and sometimes not.

However, the routed transactions themselves “know” when an affinity is created, and can communicate this to the dynamic transaction routing program. The routing program is then able to route such transactions accordingly.

Monitoring the output TIOA

When invoked at transaction termination, your routing program can examine the copy of the routed transaction’s output TIOA pointed to by DYRBPNTTR. This can be useful, for example, to guard against the situation where one AOR in a CICSplex develops software problems. These may be reported by means of a message to the end user, rather than by a transaction abend. If this happens, the routing program is unaware of the failure and cannot bypass the AOR that has the problem. By reading the output TIOA, your routing program can check for messages indicating specific kinds of failure, and bypass any AOR that is affected.

Some processing considerations

- Any of the EXEC CICS commands (except EXEC CICS RECEIVE—see “Modifying the initial terminal data” on page 482) can be issued from the routing program. You are likely to find the EXEC CICS INQUIRE CONNECTION and INQUIRE IRC commands particularly useful if you want to confirm that a link is available before routing a transaction. The EXEC CICS INQUIRE and SET commands are described in the *CICS/ESA System Programming Reference* manual.
- Although the routing program can issue any EXEC CICS command, you should consider carefully the effect of commands that alter protected resources, because changes to those resources may be committed or backed out inadvertently as a result of logic in the routed transaction. You should also consider carefully the effect of EXEC CICS SYNCPOINT and ABEND commands on APPC transaction routing.
- If you want to keep information about how transactions are routed, it must be done in the user routing program, perhaps by writing the information to a temporary storage queue associated with this terminal.

⁶ The IBM CICS Transaction Affinities Utility MVS/ESA, program number 5696-582.

parameters passed to DFHDYP

- Several transactions can form a single conversation with the end user. At the start of the conversation, resources are allocated to record the state of the conversation. Because these resources are local to the system to which the first transaction in the conversation was routed, the routing program must be able to continue to route to this system until the end of the conversation.
- It is important to avoid creating “tangled daisychains”: for any transaction that is being dynamically routed, you must avoid routing back to a node that has previously been routed from.
- The dynamic transaction routing program can be RMODE ANY but must be AMODE 31.

Unit of work considerations

Depending on the terminal type, the CICS relay program, the dynamic transaction routing program, and the routed transaction may constitute a single unit of work. Any protected resources owned by the dynamic transaction routing program could therefore be affected by the syncpoint activity of the routed transaction. This means that these resources may be committed or backed out inadvertently by the routed transaction. If you want to avoid this, you have to define the routing program's resources as unprotected rather than protected.

Parameters passed to the dynamic transaction routing program

Figure 38 on page 485 shows all the parameters passed from DFHAPRT, the CICS relay program, to the dynamic transaction routing program by means of a communications area. The communications area is mapped by the copy book DFHDYPDS, which is in the appropriate CICS library for all the supported programming languages.

		DS	OCL4	Standard Header
	DYRFUNC	DS	CL1	Function Code
	DYRCOMP	DS	CL2	Component Code Always 'RT'
	DYRFILL1	DS	CL1	Reserved
	DYRERROR	DS	CL1	Route selection error code
	DYROPTER	DS	CL1	Transaction termination option
	DYRQUEUE	DS	CL1	Queue-the-request indicator
	DYRFILL2	DS	CL1	Reserved
	DYRRETC	DS	F	Return code
	DYRSYSID	DS	CL4	Default/Selected sysid
#	DYRVER	DS	H	Version of the interface
#	DYRFILL3	DS	H	Reserved
	DYRTRAN	DS	CL8	Default/Selected remote tranid
	DYRCOUNT	DS	F	Number of invocations count
	DYRBPNTN	DS	F	Address of input buffer
	DYRBLGTH	DS	F	Length of input buffer
#	DYRTPRI	DS	CL1	Route priority to AOR?
#	DYRFILL4	DS	CL1	Reserved
#	DYRPTY	DS	H	Dispatch priority passed to AOR
	DYRNETNM	DS	CL8	Netname matching sysid
	DYRLPROG	DS	CL8	Run this program if routed locally
	DYRDTRXN	DS	CL1	DTRTRAN indicator
	DYRDTRRJ	DS	CL1	DTRTRAN reject
	DYRSRCK	DS	XL4	MVS WLM source token
	DYRABCDE	DS	CL4	Transaction abend code
	DYRCABP	DS	CL1	Continue abend processing?
	DYRFILL5	DS	CL4	Reserved
	DYRACMAA	DS	F	Address of applications's commarea
	DYRACMAL	DS	F	Length of application's commarea
	DYRUSER	DS	CL128	User area

Figure 38. The communication area passed to a dynamic transaction routing program

DYRABCDE

is the abend code returned when a remote or locally routed transaction abends.

DYRACMAA

is the 31-bit address of the routed application's communications area. If there is no communications area, this field is set to null.

When your dynamic routing program is invoked for routing (DYRFUNC=0), the address is that of the input communications area (if any). Likewise, when your routing program is invoked because of a route-selection error or a remote ATI request (DYRFUNC=1 and 3, respectively), the address is that of the input communications area.

When your routing program is invoked because a previously-routed transaction has terminated normally (DYRFUNC=2), the address is that of the output communications area (if any). Routed applications can use their output communications area to pass information to the dynamic routing program—see “Receiving information from a routed transaction” on page 482.

When your routing program is invoked because the routed transaction has abended (DYRFUNC=4), the information in the communications area is not meaningful.

Your routing program can alter the data in the application's communications area.

DYRACMAL

is the length of the routed-to application's communications area. If there is no communications area, this field is set to zero.

DYRBLGTH

is the length of the copy of the TIOA/LUC buffer.

DYRBPNTR

is the 31-bit address of a copy of the TIOA/LUC buffer.

When your dynamic routing program is invoked for routing, because of a route-selection error, or for a remote ATI request (DYRFUNC=0, 1, and 3, respectively), it is given a copy of the input TIOA. Your routing program can alter the terminal input data passed to the routed transaction—see “Modifying the initial terminal data” on page 482.

When your routing program is invoked because a previously-routed transaction has terminated normally (DYRFUNC=2), it is given a copy of the output TIOA. Your routing program can monitor the output TIOA to detect possible problems in the AOR—see “Receiving information from a routed transaction” on page 482.

DYRCABP

indicates whether or not you want CICS to continue standard abend processing. The possible values are:

Y Continue with CICS abend processing.

N Terminate the transaction, do not continue with CICS abend processing, and give control to the program specified by DYRLPROG.

This option enables you to pass control to a local program that can handle the condition in your own way, and issue appropriate messages to terminal users.

If you enter N, you must ensure that DYRLPROG specifies the name of a valid program on the local system.

There is no default value.

DYRCOUNT

is a count of the times the dynamic transaction routing program has been invoked for routing purposes for this transaction with DYRFUNC set to '0', '1', or '3'. This option is provided to allow you to limit the number of retry requests.

DYRDTRRJ

indicates whether the transaction, which is defined by the common transaction definition specified on the DTRTRAN system initialization parameter, is to be rejected, or accepted for processing. This parameter is only relevant when DYRTRXN is set to Y. The possible values are:

Y The transaction is rejected. This is the default.

N The transaction is not rejected.

This indicator is always set to the reject condition when the dynamic transaction routing program is invoked. To dynamically route a transaction defined by the DTRTRAN definition, you must change this to the accept condition.

If you reject the transaction, message DFHAC2001—“Transaction '*tranid*' is unrecognized”—is sent to the user's terminal.

DYRDTRXN

indicates whether the transaction to be routed is defined by the common transaction definition specified on the DTRTRAN system initialization parameter, or by a specific transaction definition. The possible values are:

Y The transaction is defined by the definition specified by the system initialization parameter DTRTRAN. That is, there is no resource definition for the input transaction identifier (id).

The transaction is initiated in the terminal-owning region using the transaction id specified by the system initialization parameter, DTRTRAN. The input transaction id is passed to the dynamic transaction routing program in the DYRTRAN field.

N The transaction is not defined by the definition specified by the system initialization parameter, DTRTRAN. There is an installed resource definition for the input transaction id.

The transaction is initiated in the terminal-owning region using the input transaction id. The transaction id passed to the dynamic transaction routing program in the DYRTRAN field is the remote transaction id from the transaction resource definition (if this is different from the input transaction id).

For an explanation of the DTRTRAN system initialization parameter, see the *CICS/ESA System Definition Guide*.

DYRERROR

has a value only when DYRFUNC is set to '1'. It indicates the type of error that occurred during the last attempt to route a transaction. The possible values are:

0 The selected sysid is unknown.

1 The selected system is not in service.

2 The selected system is in service, but no sessions are available.

3 An allocate request has been rejected, and SYSIDERR returned to the application program. This error occurs for one of the following reasons:

1. An XZIQUE global user exit program requested that the allocate be rejected, or
2. CICS rejected the allocate request automatically because the QUEUELIMIT value specified on the CONNECTION resource definition has been reached.

4 A queue of allocate requests has been purged, and SYSIDERR returned to all the waiting application programs. This error occurs for one of the following reasons:

1. An XZIQUE global user exit program requested that the queue be purged, or
2. CICS purged the queue automatically because the MAXQTIME limit specified on the CONNECTION resource definition has been reached.

DYRFUNC

tells you the reason for this invocation of the dynamic transaction routing program. The possible values are:

- 0** Invoked for transaction routing
- 1** Invoked because an error occurred in route selection
- 2** Invoked because a previously routed transaction has terminated
- 3** Invoked before automatic initiation
- 4** Invoked because the routed transaction abended.

DYRLPROG

is the name of the initial program associated with the transaction for which the dynamic transaction routing program is invoked, if the transaction is defined for dynamic routing. If the transaction is being statically routed (it is defined with DYNAMIC(NO) and a specific remote system name), this field contains blanks.

You can use this field to specify the name of an alternative program to be run if the transaction is routed locally. For example, if all remote CICS regions are unavailable, and the transaction cannot be routed, you may want to run a program in the local terminal-owning region to send an appropriate message to the user.

Note: DYRLPROG must not be set to blanks when you specify DYRCABP=N. If you specify DYRCABP, ensure you also specify a valid program name on DYRLPROG.

DYRNETNM

is the netname of the CICS region identified in DYRSYSID.

If the DYRNETNM value is changed by the initial invocation of the dynamic transaction routing program, CICS tries to route the transaction to the CICS region with the new netname.

DYROPTER

specifies whether the dynamic transaction routing program is to be reinvoked when the routed transaction terminates. The possible values are:

- N** The dynamic transaction routing program is not to be reinvoked. This is the default.
- Y** The dynamic transaction routing program is to be reinvoked.

You can specify this option for transactions that are routed to remote CICS regions and also for transactions that are executed locally.

DYRQUEUE

identifies whether or not the request is to be queued if no sessions are immediately available to the remote system identified by DYRSYSID. The possible values are:

- Y** The request is to be queued if necessary. This is the default.
- N** The request is not to be queued.

DYRPRTY

can be used to set the dispatch priority of the task in the application-owning region, if the connection between the terminal-owning region and application-owning region is MRO.

CICS sets this value to '0' (zero) before invoking the dynamic transaction routing program. If the DYR RTPRI value is 'Y' on return from the initial invocation, CICS passes the DYPPRTY value to the application-owning region.

DYRRET C

contains a return code that tells CICS how to proceed. The possible values are:

- 0** Continue processing the transaction.
- 4** Terminate the transaction without message or abend.
- 8** Terminate the transaction with either a message or an abend.

Whenever the routing program is invoked, DYRRET C is set to '0'. If you want CICS to continue processing the transaction, you must leave it set to '0'.

To make CICS terminate the transaction (issuing a message or abend), return a value of '8'.

To make CICS terminate the transaction without issuing a message or abend (indicating that DFHDYP has done all the processing that is necessary), return a value of '4'.

Warning: Setting a return code of '4' for APPC transaction routing leads to unpredictable results, and should be avoided.

You do not need to set a return code when the routing program is invoked at transaction termination. (Any code you set is ignored by CICS.)

DYR RTPRI

indicates whether or not the dispatch priority of the transaction should be passed to the application-owning region, if the connection between the terminal-owning region and the application-owning region is MRO. The possible values are:

- N** The dispatch priority is not passed. This is the default.
- Y** The dispatch priority is passed.

DYRSRCTK

is the MVS workload management service and reporting class token for the routed transaction.

DYRSYSID

identifies the SYSID of a CICS region. The exact meaning of this parameter depends on the value for DYRFUNC:

- When DYRFUNC is set to '0', DYRSYSID contains:
 - The remote CICS region name as specified on the REMOTESYSTEM option of the installed transaction definition, or,
 - The system name of the local CICS region if REMOTESYSTEMNAME is not specified.

The dynamic transaction routing program can accept this value or change it before returning to CICS.

If the SYSID you return to CICS is the same as the local sysid, CICS runs the transaction in the local region (the terminal-owning region).

- When DYRFUNC is set to '1', DYRSYSID contains the CICS region name returned to CICS by the dynamic transaction routing program on its previous invocation, and the SYSID is found to be in error.

The action your dynamic transaction routing program can take when DYRFUNC=1 depends on the DYRERROR parameter setting:

- If DYRERROR is set to '0' (unknown sysid) or '1' (CICS region not in service) and you want CICS to retry transaction routing, you must change DYRSYSID before returning to CICS.
- If DYRERROR is set to '2' (no session available) and you want CICS to retry transaction, you must change DYRSYSID or change the value of DYRQUEUE to 'Y' (queue the request until a session is available).
- When DYRFUNC is set to '2', DYRSYSID contains the name of the CICS region on which the completed transaction executed.
- When DYRFUNC is set to '3', DYRSYSID contains the CICS region name specified on the REMOTESYSTEM option of the installed transaction definition, with DYNAMIC(NO) also specified. Any changes to this value, or to DYRNETNAME, are ignored.
- When DYRFUNC is set to '4', DYRSYSID contains the name of the CICS region on which the transaction abended.

DYRTRAN

contains the remote transaction id.

When DYRFUNC is set to '0' or '3', DYRTRAN contains the remote transaction id specified on the REMOTENAME option of the installed transaction definition. Your dynamic transaction routing program can accept this remote transaction id, or supply a different transaction name for forwarding to the remote CICS region. If the supplied name is longer than four characters, it is truncated by CICS.

You can change DYRTRAN on any call to the dynamic transaction routing program, though it is effective only when DYRFUNC is set to '0', '1', or '3'.

DYRUSER

is a 128-byte user area.

CICS initializes this user area to zeroes before invoking the dynamic transaction routing program for a given task. This user area can be modified by the dynamic transaction routing program, returned to the CICS relay program, DFHAPRT, and is passed to subsequent invocations of the dynamic transaction routing program for a given transaction instance.

DYRVER

is the version number of the dynamic transaction routing program interface. The default is "2".

Naming your dynamic routing program

The supplied, user-replaceable dynamic routing program is named DFHDYP. If you write your own version, you can name it differently.

After the system has been loaded, to find the name of the dynamic routing program currently identified to CICS, use the EXEC CICS INQUIRE SYSTEM command. Field DTRPROGRAM contains the name of the current program.

The default is DFHDYP.

To change the current program:

- Use the DTRPGM system initialization parameter. For more guidance information about how to do this, refer to the *CICS/ESA System Definition Guide*.
- Make the change online using the EXEC CICS SET SYSTEM DTRPROGRAM command. For programming information about this command, refer to the *CICS/ESA System Programming Reference* manual.

Note: A sample definition is provided for DFHDYP, but you must install a new resource definition for a customized dynamic routing program.

Testing your dynamic transaction routing program

You can use the CICS execution diagnostic facility (EDF) to test your dynamic transaction routing program. To do so, you must name your program something other than DFHDYP, because you cannot use EDF for programs that begin with "DFH". For details of how to use EDF, see the *CICS/ESA Application Programming Guide*.

You can use EDF in either single- or dual-terminal mode. If you choose single-terminal mode, EDF displays screens for both the dynamic transaction routing program and the application program that is invoked by the routed transaction. The screens relate to:

- The initial invocation of the dynamic transaction routing program for transaction routing (DYRFUNC=0 or DYRFUNC=3)
- The invocation of the dynamic transaction routing program if an error occurs in route selection (DYRFUNC=1)
- The invocation of the application program
- The termination of the task
- The invocation of the dynamic transaction routing program at termination of the routed transaction (DYRFUNC=2), if you have specified DYROPTER=Y
- The invocation of the dynamic transaction routing program if the routed transaction abends (DYRFUNC=4).

If you want EDF to display the execution of your dynamic routing program only, either choose dual-terminal mode, or use one of the other methods described in the *CICS/ESA Application Programming Guide*.

Dynamic transaction routing sample programs

The CICS-supplied sample dynamic transaction routing program is an assembler-language command-level program, named DFHDYP. The corresponding sample copy book that defines the communication area is DFHDYPDS. In addition, there are COBOL, PL/I, and C source-level samples and copy books. The names of the supplied programs and copy books, and the CICS410 libraries in which they can be found, are summarized in Table 114. Note that the COBOL, PL/I, and C versions have aliases of DFHDYP (programs) or DFHDYPDS (copy books).

Table 114. Dynamic transaction routing programs and copy books

Language	Member name	Alias	Library
Programs:			
Assembler	DFHDYP	—	SDFHSAMP
COBOL	DFHDYPO	DFHDYP	SDFHCOB
PL/I	DFHDYPP	DFHDYP	SDFHPL1
C/370	DFHDYPC	DFHDYP	SDFHC370
Copy books:			
Assembler	DFHDYPDS	—	SDFHMAC
COBOL	DFHDYPDO	DFHDYPDS	SDFHCOB
PL/I	DFHDYPDP	DFHDYPDS	SDFHPL1
C/370	DFHDYPCD	DFHDYPDS	SDFHC370

You can write your own dynamic routing program in VS COBOL II, PL/I, C, or assembler language, and you can change the name of the program.

When invoked with DYRFUNC set to '0' or '3', the sample programs accept the sysid and remote transaction name that are passed in fields DYRSYSID and DYRTRAN of the communication area, and set DYRRET to '0' before returning to CICS. When invoked with DYRFUNC set to '1' or '4', they set a return code of '8'.

If you want to route transactions dynamically, you must customize DFHDYP, or replace it completely with your own routing program.

Chapter 15. The user-replaceable journaling programs

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

CICS/ESA 4.1 provides automatic journal archiving, as described in Chapter 18, “CICS journaling” on page 521. However, you can still use the two user-replaceable programs, DFHXJCO and DFHXJCC, in CICS/ESA 4.1 to use your own archiving procedures.

The user-replaceable programs, DFHXJCO and DFHXJCC

There are two user-replaceable journaling programs. These are DFHXJCO and DFHXJCC.

DFHXJCO is invoked just before an open-for-output request to a non-SMF disk journal.

DFHXJCC is invoked just after a request to close an open-for-output, non-SMF disk journal. DFHXJCC is invoked whether the CLOSE request is successful or not. If DBRC is operating, the module receives control before the call to DBRC that indicates the closing of the log data set.

The supplied versions of DFHXJCO and DFHXJCC simply return to their callers. You can write your own programs to replace the CICS-supplied versions if you want to do any additional processing at journal open or close time. Note, if you do write your own programs, that they must be coded in assembler language.

Some possible uses are:

- Control procedures to ensure that journal data sets have been archived before they are made available for reuse.
- Automatic submission of journal archiving jobs through an internal reader, rather than using manual procedures. This can be valuable in an XRF environment, where the need to minimize operator involvement in the recovery procedure may be an important factor.
- Dynamic allocation of journal data sets to avoid the need to archive them immediately before they may be reused.

The communication area

The type of OPEN or CLOSE is passed to the user-replaceable programs, along with additional information, in a communication area. The communication area contains all the information you may need to do any extra processing in the user-replaceable programs. It is listed in Figure 39 on page 494.

user-replaceable journaling programs

DFHXJCOM	DSECT		
XJCOMBA	EQU	*	Commarea begin address
COMINVDS	DS	0CL4	Invocation Descriptor
COMLOCDS	DS	CL1	Local Invocation Descriptor
COMEFO	EQU	C'1'	End of file open
COMEFC	EQU	C'2'	End of file close
COMFOPN	EQU	C'3'	First open
COMMSCL	EQU	C'4'	Midstream close
COMMSOP	EQU	C'5'	Midstream open
COMSDCL	EQU	C'6'	Shutdown close
COMGLBDS	DS	CL2	Global Invocation Descriptor 'JC'
	DS	CL1	Reserved
COMJRNUM	DS	XL2	Hexadecimal Journal Number
COMJRNS	DS	CL1	Journal Data Set Id (A,B,X)
	DS	CL1	Reserved
COMJNDCB	DS	A	DCB Address
COMJNDDN	DS	CL8	Journal DDname
XJCOMEA	EQU	*	Commarea end address
XJCOMLEN	EQU	XJCOMEA-XJCOMBA	Commarea length

Figure 39. DFHXJCOM communication area

APAR PN 81337

Documentation for APAR PN81337 added on 19th May 1996.

The settings of field COMLOCDS have the following meanings:

- COMEFO** Set when the journal is opened during CICS emergency restart to ensure that end-of-file (EOF) is written. This occurs for the system log only when an X-extent is defined.
- COMEFC** Set when the journal is closed during CICS emergency restart to ensure that EOF is written. This occurs for the system log only when an X-extent is defined.
- COMFOPN** Set when the journal is first opened at normal CICS start up.
- COMSDCL** Set when the journal is closed at CICS shutdown.
- COMMSCL** Set when the journal is closed at times other than shutdown and emergency restart.
- COMMSOP** Set, for the system log with an X-extent, when the journal is opened at times other than shutdown and emergency restart.

You should address the communication area as follows:

```
EXEC CICS ADDRESS COMMAREA (XJCOMPTR)
      USING DFHXJCOM,XJCOMPTR
```

Limitations on the use of DFHXJCO and DFHXJCC

Because DFHXJCO and DFHXJCC may be invoked during CICS initialization, before support for other languages is available, they must be coded in assembler language.

You can only use a certain set of CICS commands in programs DFHXJCO and DFHXJCC, for the following reasons:

- You must not invoke journal control and many of its requesters recursively.

- The user-replaceable programs may be invoked during initialization before some of the CICS services are initialized.

You are strongly advised not to use any CICS commands that are not in the list below; if you do, the results are unpredictable. Permitted commands are:

- EXEC CICS HANDLE CONDITION and IGNORE CONDITION
- EXEC CICS PUSH HANDLE and POP HANDLE
- EXEC CICS ABEND and HANDLE ABEND
- EXEC CICS ADDRESS and ASSIGN
- EXEC CICS GETMAIN and FREEMAIN
- EXEC CICS RETURN
- EXEC CICS WAIT EVENT
- EXEC CICS INQ JOURNALNUM and INQ SYSTEM.

If you want to modify the programs so that they can do extra processing, you must take into account the circumstances under which these user-replaceable programs are called. Both may be called during initialization if CICS has determined that the journals were not closed during the previous run of CICS. Under these circumstances, CICS may issue an end-of-file OPEN and CLOSE prior to the first journal OPEN.

The sample programs and copy book

The source-code of the default journaling programs, DFHXJCO and DFHXJCC, is provided, in assembler language only, in the CICS410.SDFHSAMP library. A copy book, DFHXJCOM, which maps the communication area, is in the CICS410.SDFHMAC library.

user-replaceable journaling programs

Chapter 16. Writing a CICS–DBCTL interface status program

Considerations common to all user-replaceable programs

Note that the comments contained in Chapter 5, “General notes about user-replaceable programs” on page 343 apply to this chapter.

The CICS–DBCTL interface status program DFHDBUEX is a user-replaceable program forming part of the support for the CICS–DBCTL interface. It is designed to invoke user-supplied code whenever CICS successfully connects to or disconnects from DBCTL. It runs in a CICS application environment and is driven at specific points to allow you to enable and disable your CICS-DL/I transactions when the CICS–DBCTL interface initializes or terminates.

DFHDBUEX is invoked in the following case for the ENABLE command:

- CICS has connected to DBCTL successfully. This occurs after a connection request has been issued from CICS to DBCTL. The control exit (DFHDBCTX) is invoked by the database resource adapter (DRA)⁷ for ‘initialization complete’. The control exit posts the control transaction (CDBO). The control program (DFHDBCT) then invokes DFHDBUEX.

DFHDBUEX is invoked in the following cases for the DISABLE command:

- A request has been issued to disconnect from DBCTL. The CICS–DBCTL menu program (DFHDBME) starts the disconnection transaction (CDBT) to disconnect from DBCTL. The disconnection program (DFHDBDSC) invokes DFHDBUEX before issuing the interface termination request to the adapter.
- The control transaction (CDBO) has been notified of one of the following events:
 - A checkpoint freeze request to DBCTL
 - DRA abnormal termination
 - DBCTL abnormal termination.

In each of these cases, the control program (DFHDBCT) invokes DFHDBUEX.

Input to DFHDBUEX is by means of a communication area addressed by DFHEICAP. The layout of the communication area is shown in Figure 40.

DBUSHEAD	DS	OCL4	Standard Header	
DBUREQT	DS	CL1	Function Code	
DBUCOMP	DS	CL2	Component Code	Always 'DB'
DBURESV	DS	CL1	Reserved	
DBUREAS	DS	CL1	Reason for disconnection	
DBUSUFF	DS	CL2	DRA startup table suffix	
DBUDBCTL	DS	CL4	DBCTL identifier	

Figure 40. The DFHDBUEX communication area

⁷ The interface that enables DBCTL databases to be accessed from CICS.

the CICS–DBCTL interface status program

The parameter list contains the following information:

DBUREQT Request Type. The function code has one of the following values:

DBUCONN (X'01') Connected

DBUDISC (X'02') Disconnected.

DBUREAS Reason for Disconnection. Contains flags:

DBUMENU (X'01') Disconnected from menu

DBUDBCC (X'02') Checkpoint Freeze input to DBCTL

DBUDRAF (X'03') DRA Failure has taken place

DBUDBCF (X'04') DBCTL Failure has taken place.

DBUSUFF DRA startup table suffix.

DBUDBCTL DBCTL identifier.

The sample program and copy book

The source-code of the supplied CICS–DBCTL interface status program, DFHDBUEX, is provided, in assembler language only, in the CICS410.SDFHSAMP library. A corresponding copy book, DFHDBUCA, that maps the communication area, is in CICS410.SDFHMAC.

The sample program checks for the presence of the input parameters (passed in the communication area). If these do not exist, control returns to the calling program.

The type of request (CONNECTION|DISCONNECTION) is then determined, and a branch is taken to the appropriate function routine (CONPROC|DISPROC).

The sample contains an example, as part of a comment, of how to enable and how to disable a transaction. To use the program, it is necessary for transactions using DBCTL to be defined in the CSD as DISABLED.

You can code your own CICS–DBCTL interface status program in any of the languages supported by CICS. For information about the job control statements necessary to assemble and link-edit user-replaceable programs, refer to “Assembling and link-editing user-replaceable programs” on page 344.

Part 4. Customizing the XRF overseer program

A general note about user-written programs	500	The sample overseer program	501
Chapter 17. The extended recovery facility overseer program	501	The DFHWOSM macros	506
		Customizing the sample overseer program	516

A general note about user-written programs

On return from any user-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

Chapter 17. The extended recovery facility overseer program

The information in this chapter is of interest only to users of the extended recovery facility (XRF). Details of XRF are provided in the *CICS/ESA 3.3 XRF Guide*, and guidance information about running the overseer, including a sample job stream, is provided in the *CICS/ESA Operations and Utilities Guide*.

The XRF overseer program has two major functions:

- To display the current status of active and alternate CICS regions
- To restart failed CICS regions in place without operator intervention.

There is a CICS-supplied sample overseer program that performs these two functions and which you may find adequate for your installation.

The chapter is divided into the following sections:

1. “**The sample overseer program**” contains:
 - “The functions of the sample program”
 - “How the sample overseer program interfaces with CICS” on page 505
 - “How to tell the overseer which actives and alternates to monitor” on page 505.
2. “**The DFHWOSM macros**” on page 506 describes the macros that you use to provide services to the overseer program.
3. “**Customizing the sample overseer program**” on page 516 describes how you can extend the functions of the sample program.

The sample overseer program

The CICS-supplied sample overseer is an assembler-language batch program that runs in its own address space. The source of the sample program is in four members of CICS410.SDFHSAMP:

- DFH\$AXRO
- DFH\$AGCB
- DFH\$ADSP
- DFH\$ARES.

The associated DSECTs are supplied in DFH\$XRDS in the same library. An assembled version of the sample program is supplied in CICS410.SDFHLOAD.

The functions of the sample program

The program acts on five commands entered by the console operator. (Minimum abbreviations are shown like this: D.) The commands are as follows:

Display

to display the current status of all active-alternate pairs being monitored by the overseer program

Restart

to enable or disable the restart-in-place function of the overseer program

Snap

to take a snap dump of the sample program

End

to terminate the sample program

Open

to ask the overseer to try to open CICS availability manager (CAVM) data sets that it has previously failed to open.

The full format of the operator command entered at the MVS console is:

```
MODIFY overseer-jobname,command-identifier
```

where “command-identifier” is Display, Restart, Snap, End, or Open, or an abbreviation of any of these. The Display and Restart commands control the two major functions of the sample overseer program, which are described below. The Open command is described under “Opening CAVM data sets dynamically” on page 505.

The display function

When the operator enters the Display command at the MVS console, the sample overseer program issues a multiline write-to-operator (MLWTO) command showing the last known state of each of the active-alternate pairs that it is monitoring. The overseer retrieves this information from the control and message data sets, in which the CICS availability manager (CAVM) has been recording state and surveillance information. The display includes a title line and one line of status information for each active-alternate pair. The title line is as follows:

```
GEN-APP ACT-JOB ACT-APP ACPU A-ST BKP-JOB BKP-APP BCPU B-ST
```

Each line of status information provides the following:

- The generic applid of the active-alternate pair (GEN-APP)
- The CICS job name of the active (ACT-JOB) and of the alternate (BKP-JOB)
- The specific applid of the active (ACT-APP) and of the alternate (BKP-APP)
- The SMF IDs of the CPUs on which the active and the alternate were last known to be executing (ACPU and BCPU)
- The last known status of the active (A-ST) and of the alternate (B-ST). The status value can be one of the following:

ACT Active signed on normally and running the active CICS workload.

BACK Alternate signed on and running normally.

SOFN Signed off normally.

SOFA Signed off abnormally.

TKOV Taking over (alternate only).

INCA Incipient active, meaning that an alternate CICS is taking over from an active CICS. The active job has signed off abnormally, and the incipient active is waiting for the active job to terminate.

TKIP Takeover in progress. An alternate CICS is attempting a takeover of this active system. When the takeover is complete, the status is changed.

- UNKN** Unknown—the overseer has no current information about the status, which was in an intermediate state when the Display command was processed. Reissuing the Display command causes UNKN to be replaced by another status value.
- OLD** The information displayed for the alternate refers to out-of-date information about the system that was the alternate until a recent takeover. That system is the current active, and the information displayed for the alternate is marked as OLD until a new alternate is signed on and running normally.

An example of the status display is shown, for guidance purposes, in the *C/CS/ESA Operations and Utilities Guide*.

Note: An 'X' following any of these status values indicates that the associated job is currently executing. However, because JES services are used to discover the execution state of a job, only those jobs that are running on the same JES as the overseer program (or on the same JES shared spool) show the correct execution state. Any job that is not on the same JES shared spool appears not to be executing.

There are two additional items that may appear on the status display. These are:

NO ACTIVE DATA
NO BACKUP DATA.

These are displayed instead of status data when no data was extracted from the CAVM data sets. This happens when newly-created data sets are used—CICS has not yet written any data to them—or when the overseer fails to open the data sets.

The restart-in-place function

The overseer program can restart failed CICS regions in place automatically, if they are in the same MVS image as the overseer. The alternatives to automatic restart are operator-initiated restart, automatic takeover to the alternate, and operator-initiated takeover.

Automatic restart in place of failed regions is most useful in the multi-MVS image MRO environment. Because related regions must operate in the same MVS image, a takeover of one region means that all related regions must also be taken over by their alternates. A region may not be important enough for you to want every failure to cause a takeover to the alternate MVS image. This could disrupt users who would not otherwise have been affected by the failure. Automatic restart in place of the failed region is therefore likely to be preferred to takeover in these circumstances.

If your system consists of one or more independent regions, with actives and alternates located in separate MVS images, you can:

- Allow the overseer to restart an active region in place automatically when it fails.
- Choose automatic takeover by the alternate.
- Leave the operator to decide what to do. The operator could decide to restart the failing region in place or to initiate a takeover by the alternate, and this decision is likely to depend on which part of your system has failed.

the XRF overseer program

If you are operating an MRO system in a single MVS image, the failure of an active region can be handled by a takeover by the alternate, without causing all the related regions to be taken over, because the new active region can continue communication with the other active regions. Takeover is therefore likely to be your preferred course of action.

Enabling and disabling restart in place: The restart-in-place function of the overseer program can be enabled and disabled using the Restart command. When you enter this command, restart processing is enabled or disabled for all generic applids that the overseer is monitoring. You can also specify that particular active-alternate pairs are not to be automatically restarted in place, regardless of whether restart processing is enabled or disabled. This is described in “How to tell the overseer which actives and alternates to monitor” on page 505.

The Restart command works like an ON/OFF switch. Restart in place is enabled when the sample program is initialized. When the Restart command is first entered, restart in place is disabled. If you issue the command again, restart is enabled again, and so on. If a region fails while restart in place is disabled, no attempt to restart it is made, even if restart in place is enabled again.

Rules that control restart in place: The sample overseer program concludes that a region has failed if both:

- The region is not executing now, and was known to have been executing during the previous examination of the relevant CAVM data sets by the overseer.
- The region did not sign off normally from the CICS availability manager (CAVM).

The overseer program can restart a failed **active** region in place, if all the following conditions are met:

- Restart in place is enabled for this overseer.
- Restart in place is enabled for this active-alternate pair.
- There is no executing alternate region for this active, or the alternate region is currently defined with TAKEOVER=COMMAND. If the alternate region is defined with TAKEOVER=AUTO or TAKEOVER=MANUAL, the overseer assumes that the alternate will initiate a takeover or that the console operator will decide what action to take.
- The failing region was running in the same MVS image as the overseer.
- An attempt to restart the region in place is not already in progress.
- If the failing region belongs to a group of related regions (an MRO environment, for example), a takeover to another MVS image, perhaps initiated by another region, is not under way.

When a failed active region is restarted in place, whether by the operator or by the overseer, the corresponding alternate region cannot continue to support the new active region, and must be restarted. The overseer program restarts the **alternate** region automatically in these circumstances, if restart processing is enabled for both the failing region and the overseer.

If you want to be able to restart regions in place in both MVS images in a two-image environment, an overseer program must execute in each image.

If the failed region was started originally as a started task, the overseer program restarts it as a started task, and if the failed region was started as a job, the overseer restarts it as a job. For more guidance information about how the sample overseer program restarts failed regions in place, refer to the *CICS/ESA Operations and Utilities Guide*.

Opening CAVM data sets dynamically

When the overseer program is initialized, it is possible that some CAVM data sets have not yet been formatted by a CICS system. The overseer program obtains an 'open error' return code on these data sets, and subsequent attempts to display details about the associated CICS systems receive the response 'NO ACTIVE DATA AVAILABLE'.

This problem arises only if the overseer is initialized before all the CAVM data sets have been formatted. If it occurs, the operator can use the Open command (see page 502) to retry the opening of those CAVM data sets on which the Open previously failed. The overseer retries an Open only if the previous attempt failed with the return code X'C'. (See "DFHWOSM FUNC=OPEN macro" on page 510.)

The use of the Open command is indicated when:

- The overseer displays 'NO ACTIVE DATA AVAILABLE' for a system that the operator knows has successfully signed on to the CAVM.
- In an already established XRF environment, a new CICS/XRF system has just started up and formatted its CAVM data sets, and the operator wants future displays from the overseer to display information for the new job.

How the sample overseer program interfaces with CICS

The overseer service is made up of a CICS overseer module (name DFHWOS), which you cannot customize, and a CICS-supplied sample overseer program (module name DFH\$AXRO), which you can customize or replace with your own overseer program. DFHWOS loads the overseer program. DFHWOS and DFH\$AXRO are supplied in CICS410.SDFHAUTH.

The CICS overseer module DFHWOS provides a stable interface to the CAVM data sets and to those MVS-authorized services that the overseer program requires. The overseer program invokes those services by means of a CICS-supplied group of macros called the DFHWOSM macros, which are described, beginning on page 506.

DFHWOS therefore invokes the sample program, and is subsequently invoked by the sample program whenever the sample issues a DFHWOSM macro. The DFHWOSM macros do not interact directly with either the active or the alternate CICS address spaces.

How to tell the overseer which actives and alternates to monitor

The sample overseer program is written to handle active-alternate pairs and "related system names". A related system name identifies those regions or systems that cannot be considered in isolation by the overseer. The most common example of this is an MRO environment, where the overseer needs to be able to identify related regions when deciding whether to restart a failed region in place. Those regions or systems that are identified with a common related system name must be executed in the same MVS image.

The maximum number of active-alternate pairs that the overseer can monitor is 50.

The sample program discovers which active-alternate pairs it is monitoring from a VSAM key-sequenced data set called DFHOSD, which contains a single entry for each active-alternate pair. You create this data set and initialize it with information about active-alternate pairs before you use the overseer for the first time. You also have to redefine the DFHOSD data set whenever you want to change the information that it holds. CICS provides a sample job stream that you can use to:

- Delete and define the DFHOSD data set
- Initialize the DFHOSD data set with information about sample active-alternate pairs
- Execute the overseer code and the sample overseer program.

The sample job stream is called DFHIVXRO and is in CICS410.INSTLIB. Guidance information about this sample job stream is provided in the *CICS/ESA Operations and Utilities Guide*.

The sample overseer program reads the DFHOSD records in key sequence and builds a table of entries. Each active-alternate pair is known by its generic applid on this data set. Every entry on the data set contains the following information:

- A 12-byte key field, containing the 4-byte value 'GNbb' followed by the 8-byte generic applid of the active-alternate pair.
- The ddnames of the control data set and the message data set associated with this generic applid. Each of these is an 8-byte value.
- An optional 8-byte RELATEID, to identify related systems.
- A restart-in-place indicator to show whether a region can be restarted in place. The only value that prevents an attempt to restart in place is 'N'.

The data structure of the DFHOSD data set entries is provided in member DFH\$XRDS of CICS410.SDFHSAMP.

For a sample of the job log from an overseer job, refer to the *CICS/ESA Operations and Utilities Guide*.

The DFHWOSM macros

The DFHWOSM macros invoke the CICS module DFHWOS to provide services to the overseer program. The macros are the supported interface to the CAVM data sets, and are supplied to perform the following functions:

DFHWOSM FUNC=BUILD

Open communication with DFHWOS

DFHWOSM FUNC=CLOSE

Terminate access to the CAVM data sets for a named generic applid

DFHWOSM FUNC=DSECT

Generate required DSECTS

DFHWOSM FUNC=JJC

Issue a JES cancel for a named job

DFHWOSM FUNC={JJS|QJJS}

Discover current JES JOB status

DFHWOSM FUNC=OPEN

Initialize access to the CAVM data sets for a named generic applid

DFHWOSM FUNC=OSCMD

Issue MVS commands

DFHWOSM FUNC=READ

Retrieve status information for a named generic applid from the CAVM data sets

DFHWOSM FUNC=TERM

Close communication with DFHWOS.

The macros are described in detail in the following sections. For all the DFHWOSM macros, the following rules apply:

- The “label” field is optional.
- If the macro has an input parameter list, the address of that parameter list must be supplied as the value of the PARM operand. The address itself may be specified as a register number or as a label. Register 1 is the default value.
- If the macro has to supply either a BUILD TOKEN or an OPEN TOKEN to DFHWOS (as described in “The DFHWOSM tokens”), the token must be provided in the register specified in the TOKEN operand. Register 14 is the default value.

The DFHWOSM tokens

When DFHWOS first invokes the overseer program, it passes a value in register 1 which is known as the **ENTRY** token. The ENTRY token value is stored by the overseer program on entry and is passed back to DFHWOS as input to the BUILD, OSCMD, JJS, and JJC macros.

The DFHWOSM FUNC=BUILD macro must be the first macro issued by the overseer program and must complete successfully. The register 1 output from this macro is a second token called the **BUILD** token. The BUILD token value is stored by the overseer program and passed back to DFHWOS as input to the OPEN, CLOSE, READ, QJJS, and TERM macros.

DFHWOSM FUNC=BUILD macro

The DFHWOSM FUNC=BUILD macro must be issued by the overseer program to initialize its communication with DFHWOS. No other macro can be issued by the overseer program until DFHWOS FUNC=BUILD has completed successfully.

label	DFHWOSM FUNC=BUILD [,TOKEN={token register 14}]
-------	--

Input

The TOKEN value is the ENTRY token that was passed to the sample overseer program when it was first invoked by DFHWOS.

Output

- Register 1** Contains the BUILD token value, which must be returned as an input value by the overseer program on certain subsequent requests. This value is returned to register 1 only if register 15 has a return code of '0'.
- Register 15** Contains one of the following completion codes:
- 0** Communication successfully initialized between the overseer program and DFHWOS
 - 4** Incorrect TOKEN value supplied
 - 8** Insufficient storage.

DFHWOSM FUNC=CLOSE macro

The DFHWOSM FUNC=CLOSE macro terminates access to the CAVM data sets for a named generic applid.

```
label      DFHWOSM FUNC=CLOSE
           [,PARM={parm address|1}]
           [,TOKEN={token register|14}]
```

Input

The PARM value is a pointer to the address of the generic applid whose associated CAVM data sets are no longer to be accessed by the overseer program.

The TOKEN value is the BUILD token.

Output

- Register 15** Completion codes:
- 0** CLOSE request was successful and the CAVM data sets associated with this generic applid can no longer be accessed by the overseer program.
 - 4** Incorrect TOKEN value supplied.
 - 8** Access to CAVM data sets for the named generic applid had not been initialized.

DFHWOSM FUNC=DSECT macro

The DFHWOSM FUNC=DSECT macro generates a number of DSECTs, including the DSECT of the DBLID definitions.

```
DFHWOSM FUNC=DSECT
```

DFHWOSM FUNC=JJC macro

The DFHWOSM FUNC=JJC macro issues a JES cancel for a named job with a JES job identifier.

```
DFHWOSM FUNC=JJC
      [,PARM={parm address|1}]
      [,TOKEN={token register|14}]
```

Input

The PARM value is a pointer to the addresses of the following:

- An 8-byte job name
- An 8-byte JES job ID
- A 256-byte SSOB return area.

The TOKEN value is the ENTRY token.

Output

Register 15 Contains the following completion codes:

- 0** JES cancel completed; SSOB and status array returned from JES
- Nonzero** Return code from JES.

DFHWOSM FUNC={JJS|QJJS} macro

Supplied with a job name and JES job identifier, both versions of the DFHWOSM FUNC={JJS|QJJS} macro return the current JES job status into a copy of the JES subsystem options block (SSOB).

The FUNC=JJS macro returns control when the JES call has completed successfully or unsuccessfully. The FUNC=QJJS macro returns control immediately and posts an event control block (ECB) once the JES request has completed.

```
label DFHWOSM FUNC={JJS|QJJS}
      [,PARM={parm address|1}]
      [,TOKEN={token register|14}]
```

Input

For FUNC=JJS, the PARM value is a pointer to the addresses of the following:

- An 8-byte job name
- An 8-byte JES job ID
- A 256-byte SSOB return area.

The TOKEN value is the ENTRY token.

DFHWOSM macros

For FUNC=QJJS, the PARM value is a pointer to the addresses of the following:

- An 8-byte job name
- An 8-byte JES job ID
- A 256-byte SSOB return area
- A doubleword area to hold two ECBs.

The FUNC=QJJS macro requires 2 ECBs: the first is posted when the JES call completes; the second is posted if a time-out occurs before JES returns.

The TOKEN value is the BUILD token.

Output

Register 15 Contains the following completion codes:

- | | |
|----------------|--|
| 0 | JES status returned as requested in the SSOB return area |
| Nonzero | Return code from JES. |

DFHWOSM FUNC=OPEN macro

The DFHWOSM FUNC=OPEN macro initializes access to the CAVM data sets for a named generic applid.

```
label      DFHWOSM FUNC=OPEN
           [,PARM={parm address|1}]
           [,TOKEN={token register|14}]
```

Input

The PARM value is a pointer to three further addresses, and these are:

1. The address of the generic applid
2. The address of the ddname of the control data set
3. The address of the ddname of the message data set.

The TOKEN value is the BUILD token.

Output

Register 15 Contains one of the following completion codes:

- | | |
|----------|---|
| 0 | Access initialized, active and alternate signed on |
| 1 | Access initialized, active signed on |
| 2 | Access initialized, alternate signed on |
| 3 | Access initialized, nothing signed on |
| 4 | Same SMF MVS name; IPL time of active earlier than MVS IPL time |
| 5 | Same SMF MVS name; IPL time of alternate earlier than MVS IPL time |
| 6 | Insufficient storage |
| 7 | Generic applid is not associated with the named CAVM data sets |
| 8 | Access already initialized for this generic applid or for this ddname |

9	Version numbers of the named CAVM data sets do not match
C	Data set open failure
10	SHOWCB failure.

A register 15 return code value of '0' through '5' indicates that a DFHWOSM FUNC=READ macro can now be issued. A return code value of '6' or above indicates that the OPEN has failed and that the overseer program will not be able to access the CAVM data sets.

DFHWOSM FUNC=OSCMD macro

The DFHWOSM FUNC=OSCMD macro is used to issue MVS commands. (The overseer program restarts a failed region in place by issuing this macro.) The text of the required MVS command is provided as input to the macro, and the OSCMD service issues an SVC 34 specifying this command text. In addition, the OSCMD service issues an MVS WTO request so that a copy of the command text appears on the MVS console to keep the operator informed of what is about to happen. This copy has the comment '(BY IOP)' appended to show that the command is going to be issued by an overseer program. A second copy of the command text is sent to the console when the MVS command is issued.

```
label      DFHWOSM FUNC=OSCMD
           [,PARM={parm address|1}]
           [,TOKEN={token register|14}]
```

Input

The PARM value is a single address that points to a "command area". The command area is made up of a 4-byte length field followed by the command data. The length field contains the length of the whole command area. The command data must be in write-to-operator (WTO) command format.

The TOKEN value is the ENTRY token.

Output

Register 0 Completion code set by SVC 34 as a response to the MVS command that was issued by the macro.

Register 15 Response to the macro itself. A return code of '16' indicates that the OSCMD has failed.

DFHWOSM FUNC=READ macro

The DFHWOSM FUNC=READ macro returns information about a named generic applid from its associated CAVM data sets.

```
label      DFHWOSM FUNC=READ
           [,PARM={parm address|1}]
           [,TOKEN={token register|14}]
```

Input

The PARM value is a pointer to a parameter list that contains the addresses of the generic applid and the “dbllist”. The dbllist is a list of one or more doublewords.

In the first two bytes of the second word of each of these doublewords you supply the DBLID of the information you require. Each piece of information that you can request is identified by a DBLID, and a list of these is provided in Figure 42 on page 513.

The first word of each doubleword is an output area to contain the address of the requested information, and the last two bytes of the second word of each doubleword contain the length of the information. The end of the dbllist is signaled by setting the high-order bit of the last doubleword to ‘1’. Figure 41 illustrates the input to and output from the READ macro.

The TOKEN value is the BUILD token.

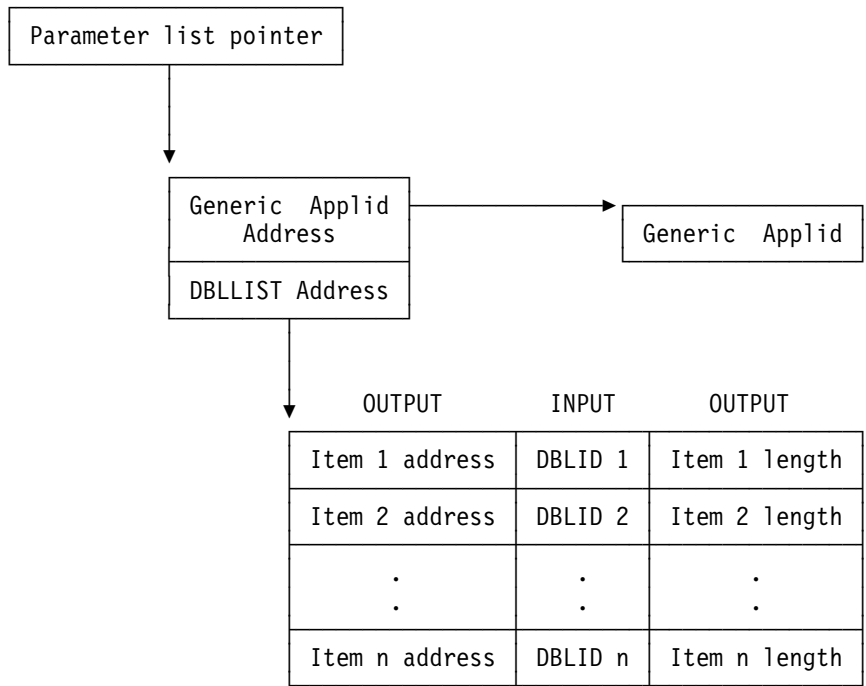


Figure 41. Input to and output from the DFHWOSM FUNC=READ macro

DBLIDs for the active:

DBLID1	EQU	X'0001'	JOB NAME
DBLID2	EQU	X'0002'	JES JOB ID
DBLID3	EQU	X'0003'	JOB SUBMISSION TIME (STIME)
DBLID4	EQU	X'0004'	JOB STEP TASK ATTACH TIME (ATIME)
DBLID5	EQU	X'0005'	CANCEL NAME
DBLID6	EQU	X'0006'	JES SSNAME
DBLID7	EQU	X'0007'	MVS SMF NAME
DBLID8	EQU	X'0008'	MVS IPL TIME
DBLID9	EQU	X'0009'	SPECIFIC APPL NAME
DBLID10	EQU	X'000A'	ADDRESS SPACE IDENTIFIER (ASID)
DBLID11	EQU	X'000B'	RESTART TYPE
DBLID12	EQU	X'000C' - X'001F'	SPARE FOR STATE CTL ITEMS
DBLID32	EQU	X'0020'	HEARTBEAT INTERVAL
DBLID33	EQU	X'0021'	HEARTBEAT COUNTER
DBLID34	EQU	X'0022'	MSG FILE CURSOR
DBLID35	EQU	X'0023'	STATUS VALUE (STATE)
DBLID36	EQU	X'0024'	INQUIRE HEALTH DATA
DBLID37	EQU	X'0025'	INQUIRE GLOBAL DATA
DBLID38	EQU	X'0026'	SYSPLEX NAME
DBLID39	EQU	X'0027'	MVS SYSTEM NAME
DBLID40	EQU	X'0028'	MVS SYSTEM TOKEN

DBLIDs for the alternate:

DBLID257	EQU	X'0101'	JOB NAME
DBLID258	EQU	X'0102'	JES JOB ID
DBLID259	EQU	X'0103'	JOB SUBMISSION TIME (STIME)
DBLID260	EQU	X'0104'	JOB STEP TASK ATTACH TIME (ATIME)
DBLID261	EQU	X'0105'	CANCEL NAME
DBLID262	EQU	X'0106'	JES SSNAME
DBLID263	EQU	X'0107'	MVS SMF NAME
DBLID264	EQU	X'0108'	MVS IPL TIME
DBLID265	EQU	X'0109'	SPECIFIC APPL NAME
DBLID266	EQU	X'010A'	ADDRESS SPACE IDENTIFIER (ASID)
DBLID267	EQU	X'010B'	RESTART TYPE
DBLID268	EQU	X'010C' - X'011F'	SPARE FOR STATE CTL ITEMS
DBLID288	EQU	X'0120'	HEARTBEAT INTERVAL
DBLID289	EQU	X'0121'	HEARTBEAT COUNTER
DBLID290	EQU	X'0122'	MSG FILE CURSOR
DBLID291	EQU	X'0123'	STATUS VALUE (STATE)
DBLID292	EQU	X'0124'	INQUIRE HEALTH DATA
DBLID293	EQU	X'0125'	INQUIRE GLOBAL DATA
DBLID294	EQU	X'0126'	SYSPLEX NAME
DBLID295	EQU	X'0127'	MVS SYSTEM NAME
DBLID296	EQU	X'0128'	MVS SYSTEM TOKEN

Figure 42. DBLIDs for the DFHWOSM FUNC=READ macro

Notes:

1. The data structures of the status information pointed to by items X'0024' and X'0124' are provided in DSECT DFHXRHDS of CICS410.SDFHMAC.
2. The data structures of the status information pointed to by items X'0025' and X'0125' are provided in DSECT DFHXRGDS of CICS410.SDFHMAC.
3. The data structures of the status information pointed to by items X'000B' and

X'010B' are mirrors of the field WSJRST in DSECT WSJDESC within DFHWSMDS.

Output

Register 15	Contains one of the following completion codes:
0	Read successful, active and alternate signed on
1	Read successful, active signed on
2	Read successful, alternate signed on.
3	Read successful, nothing signed on
4	Same SMF MVS name; IPL time of active earlier than MVS IPL time
5	Same SMF MVS name; IPL time of alternate earlier than MVS IPL time
8	CAVM data set access not initialized
10C	DBLID not known
1xx	Read subtask problem.

If a completion code of '0' through '5' is returned to register 15, each doubleword of the DBLLIST contains the address (4 bytes) and the length (2 bytes) of the output from this read. A completion code of '8', '10C', or '1xx' indicates a READ failure.

Reading DBCTL status information from the CAVM data sets: If you are using DBCTL and have active and alternate DBCTL subsystems, status information about the subsystem connected to the active CICS is written to the CAVM data sets. However, the supplied sample overseer program does not read the DBCTL information from the CAVM data set. If you want the overseer to retrieve this information and to display or use it, you must write your own overseer program.

The information in the CAVM data set about the connected DBCTL subsystem is updated when the active CICS:

- Establishes a connection to a DBCTL subsystem
- Disconnects normally from a DBCTL subsystem
- Loses a connection to a DBCTL subsystem.

If more than one active CICS is connected to a single DBCTL subsystem, the CAVM data set for each CICS contains information about the same DBCTL subsystem. The overseer can recognize this situation because in every case the DBCTL startup time stamp is the same.

Copy book DFHDXGHD contains the information shown in Figure 43 on page 515.

DFHDXGHD	DSECT		
GHDDXADB	DS	CL4	DBCTL SSID
GHDDXRSE	DS	CL8	IMS RSE name
GHDDXCTM	DS	CL4	IMS connect time
GHDDXDTM	DS	CL4	IMS disconnect time
GHDDXJNM	DS	CL8	JES Job name of old active IMS
GHDDXJID	DS	CL8	JES Job ID of old active IMS
GHDDXASD	DS	H	ASID of old active IMS
GHDDXIRT	DS	X	IMS region type
DXRHTSBY	EQU	X'01'	region type is hot standby
DXRDBDC	EQU	X'02'	region type is IMS DB/DC
DXRDBCTL	EQU	X'04'	region type is DBCTL
GHDDXTYP	DS	X	GHD message type
DXMCNCT	EQU	X'01'	Message type = DBCTL connection
DXMDISC	EQU	X'02'	Message type = DBCTL disconnect
DXMDRAF	EQU	X'04'	Message type = DRA failure
DXMABEND	EQU	X'08'	Message type = DBCTL abend
DXMERROR	EQU	X'80'	Indicates severe error in DFHDBCT
DXGHDLEN	EQU	*-DFHDXGHD	length of CICS XRF/DBCTL GHD
ORG		,	

Figure 43. DFHDXGHD copy book

DFHWOSM FUNC=TERM macro

The DFHWOSM FUNC=TERM macro terminates communication between the overseer program and DFHWOS, and releases any associated storage. It must be issued before the overseer program completes to ensure an orderly termination.

```
label      DFHWOSM FUNC=TERM
           [,TOKEN={token register|14}]
```

Input

The TOKEN value is the BUILD token.

Output

Register 15 Contains the following completion codes:

- 0** Communication terminated successfully
- Nonzero** Request failed.

Customizing the sample overseer program

The sample overseer program consists of four modules link-edited together. The main module is **DFH\$AXRO**; the other three modules are subroutines, described below:

DFH\$AGCB

sets up request parameter lists (RPLs), access method control blocks (ACBs), and parameter lists.

DFH\$ADSP

displays status information.

DFH\$ARES

performs restart in place.

The associated DSECTs are provided in member DFH\$XRDS of CICS410.SDFHSAMP. There are several ways in which you can change the supplied code to make the overseer program more suitable for your installation.

Here are some customization suggestions:

- If the supplied display of status information (DSECT DSPDS) is not suitable, you can change the layout for your installation. You may also want to change the content of the status display if you are using DBCTL. The supplied sample overseer program does not read the DBCTL-related information from the CAVM data sets, and so cannot display DBCTL information on the MVS console. If you want this to be displayed, you have to issue a DFHWOSM FUNC=READ macro to retrieve the global data, and you have to change the status display to accommodate the extra information. DFHWOSM FUNC=READ is described on page 512.
- The CSECT DFH\$ADSP can be customized so that, for example, status information is displayed automatically at regular intervals, or whenever a region is in trouble, as well as when the console operator enters the Display command. This would require interpretation of the status information by the overseer.
- Any of the messages to the system console, which are listed in the prolog of the source module DFH\$AXRO, can be changed.
- You can change the format or the content of the DFHOSD data set (DSECT OSDDS) if, for example, you want it to contain more information.
- You can change the restart function so that, for example, a failed region is restarted only during periods of heavy use, while at other times a takeover to the alternate is initiated by the operator.
- When an active region fails and is taken over by the alternate, the old active region must be restarted as the new alternate. In those cases where the cause of the takeover was not an MVS failure, restart of the old active as an alternate region could be automated in the overseer program.
- To extend the function of the overseer program, you can incorporate the CEBT command, which is normally issued by the console operator to control the alternate. The CEBT command is described in the *CICS/ESA CICS-Supplied Transactions* manual.

All of the CEBT functions are available for use in the overseer program, though it is unlikely that you will find it helpful to automate all of them, and there would, in some cases, be difficulties in handling the responses from the INQUIRE

commands. However, it might be helpful for you to be able to automate the takeover process in some circumstances. Here are two examples of situations in which you could use the CEBT command to influence or to initiate takeover from the overseer program:

- The active CICS may place error information in the CAVM data sets when a VTAM failure occurs, depending on whether you have coded an exit program at the global user exit point XXRSTAT, and, if so, how you have coded it. (An exit program at this point can be used to decide whether or not VTAM failure data is recorded in the CAVM data sets.) If such data is placed in the CAVM data sets, information about the last eight failures detected by the active CICS region is available to the overseer. The overseer can evaluate this information and, if necessary, initiate a takeover by issuing the following CEBT command:

```
MODIFY jobname,CEBT PERFORM TAKEOVER
```

In this case, you should ensure that the actions taken by the global user exit program at exit point XXRSTAT do not conflict with or duplicate those taken by the overseer program. For example, it would be possible for the global user exit program to request a CICS abend, and thereby initiate a takeover, and for the overseer program to issue the PERFORM TAKEOVER command while acting on the same information.

- At particular times of the day, perhaps when fewer operational staff are available than at other times, you may find it convenient to change the TAKEOVER setting for some, or all, of your regions. For example, you can change the TAKEOVER value for a region from COMMAND or MANUAL to AUTO, without shutting down the alternate, so that takeover is automatic until the setting is next changed. The CEBT command is as follows:

```
MODIFY jobname,CEBT SET TAKEOVER AUTO
```

In both of these examples, you would include takeover commands in the command list tables (CLTs) of these regions to ensure that their related regions are also switched when appropriate.

There is one optional section of code in the overseer program, which is described below.

Loop or wait detection

The sample overseer program includes some code that you can use to detect possible loops or waits in the active CICS region.

The sample program monitors the CICS task control block (TCB) time stamp. If this remains the same for a period defined by the variable LOOPTM, a message is sent to the console warning of a possible loop or wait. The value of LOOPTM is the number of seconds (wait time) before a loop is suspected, and may need to be changed to suit your requirements and to avoid the detection of “false” loops. It should be set to a value greater than the largest runaway task time interval (as specified on the ICVR system initialization parameter) to avoid detection of user transaction loops.

To include this LOOP WARNING code, set the variable &LOOPWARN to ‘1’ and reassemble the sample.

Assembling and link-editing the overseer program

The non-specific job control statements required to assemble and link-edit the overseer program are the same as those required for user-replaceable programs, and are described in “Assembling and link-editing user-replaceable programs” on page 344.

The specific link-edit statements that you require are:

```
ORDER DFH$AXRO    this CSECT is in USERTEXT(DFH$AXRO)
ORDER DFH$AGCB    this CSECT is in DLOADLIB(DFH$AGCB)
ORDER DFH$ARES    this CSECT is in DLOADLIB(DFH$ARES)
ORDER DFH$ADSP    this CSECT is in DLOADLIB(DFH$ADSP)
INCLUDE USERTEXT(DFH$AXRO)
INCLUDE DLOADLIB(DFH$ADSP)
INCLUDE DLOADLIB(DFH$AGCB)
INCLUDE DLOADLIB(DFH$ARES)
ENTRY DFHXRONA
NAME DFH$AXRO(R)
```

If you change the overseer code in any way, note that the libraries SYS1.SDFHMAC and SYS1.AMODGEN are required for the assembly, and that the link-edit job step requires the entry name DFHXRONA. If you change any of the DSECTs used by the sample overseer program, you should reassemble the four modules.

Part 5. CICS journaling, monitoring, and statistics

A general note about user-written programs	520	Introduction to CICS monitoring	539
Chapter 18. CICS journaling	521	CICS monitoring record formats	547
Formatting a journal before output	521	Data produced by CICS monitoring	557
Opening, closing, and reading journals	522	Chapter 20. CICS statistics	579
The structure and contents of journal records	523	Introduction to CICS statistics	579
Notes on CICS journaling	537	CICS statistics record format	583
Chapter 19. CICS monitoring	539	Global user exit in the CICS statistics domain	588
		Processing the output from CICS statistics	588

A general note about user-written programs

The following comment applies to all user-written programs mentioned in Part 5 of this book:

- Upon return from any user-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

Chapter 18. CICS journaling

CICS journaling provides facilities for the creation, control, and retrieval of a special-purpose series of data sets called **journals** during real-time CICS execution. Journals are intended for recording, in chronological order, any information that you may later need to reconstruct data or events. For example, you could create journals to act as audit trails; to record database updates, additions, and deletions for backup purposes; or to track transaction activity in the system.

This chapter describes the following aspects of CICS journaling:

1. **“Formatting a journal before output”**
2. **“Opening, closing, and reading journals”** on page 522
3. **“The structure and contents of journal records”** on page 523
4. **“Notes on CICS journaling”** on page 537. This section contains:
 - “Automatic journal archiving”
 - “The journaling user-replaceable programs”
 - “The global user exits in DFHJCP”.

For introductory information about journaling, refer to the *CICS/ESA 3.3 Facilities and Planning Guide*, and for information about the uses of journaling, refer to the *CICS/ESA Recovery and Restart Guide*.

Formatting a journal before output

Before using a journal data set for the first time, you must format it. Use DFHJCJFP for formatting disk journals, and DFHFTAP for formatting tape journals.

The method by which CICS repositions at startup normally rules out the need for the journal set to be reformatted before subsequent CICS runs. Thus, *reformatting should only be undertaken in exceptional circumstances* (for example, after emergency restart has used the DFHJ01X data set). Refer to the *CICS/ESA Recovery and Restart Guide* for guidance information about journal positioning at system initialization. If you do decide to reformat, ensure that the journaled data that will be erased is no longer needed.

As an alternative to reformatting, with the data set offline, you can force an EOF into the block number 1 position. You should *not* apply this procedure (or reformatting) to the system log, nor for DL/I, nor for the automatic archiving facility, because they may be required for emergency restart.

Data collected during the current run of CICS is likely to be written after the last record from the previous run. Therefore, when reading the journals, you must be aware that data may be present for more than one CICS execution.

Opening, closing, and reading journals

Journal records are written to the journal data sets either directly from a user application program, or from a CICS management program on behalf of a user application. Journal records can be written from a user application using the EXEC CICS WRITE JOURNALNUM command. For programming information about the EXEC CICS WRITE JOURNALNUM command, see the *CICS/ESA Application Programming Reference* manual.

This section describes the commands that you use for opening and closing journals, and for reading journals offline.

Opening and closing a journal

To open or close a journal data set from an application program, use the EXEC CICS SET JOURNALNUM command. For programming information about the EXEC CICS SET JOURNALNUM command, see the *CICS/ESA System Programming Reference* manual.

Reading journals offline

Journal data sets can be read offline by user-written programs. Although written as operating-system undefined (U-format) records by CICS journal management, the blocks are compatible with records of the variable-length blocked (VB) format. Each block begins with a 4-byte block-length field, and each logical record within a block begins with a 4-byte record-length field. These 4-byte fields are in the form 'LLbb' (where "LL" represents 2 bytes containing the length of the block or record). The data set label information indicates U-format, but this can be overridden to VB (by a DD statement), so that data management deblocks records, and offers them to the offline program.

When using standard labeled tapes, a user standard header and trailer record are written at the start and end respectively of each volume. These are bypassed, depending upon the JCL options, when the journal is opened or closed.

Offline user-written programs can map journal records by issuing the DFHJCR CICSYST=YES statement, which results in the DFHJCRDS DSECT being included in the program. The DSECT thus generated is identical to that obtained for CICS programs by the COPY DFHJCRDS statement, except that the fields are not preceded by a CICS storage accounting area. The DSECT is intended to map journal records directly in the block, rather than in a CICS storage area.

Using the offline program: The offline program can be used with a DISK or TAPE journal device. The following points should be considered when reading journals offline while CICS is still active.

- For a DISK journal, two data sets should be allocated. The appropriate JCT option (JOUROPT=PAUSE or JOUROPT=AUTOARCH) and JCL statement (DISP=SHR) must be specified. You are responsible for ensuring that journal volumes are read in the required sequence, by concatenating DD statements in the correct order in your JCL.
- For a TAPE journal, the journal volume can be removed and read whenever you want. Another tape volume can be mounted to record data while the first volume is being processed. The advantages of a tape journal over a journal on a disk device are that the job to read the tape journal can run for a relatively

long time and is usually easier to process clerically because there is no need to alternate between the information on the two separate data sets.

Printing journal files: You can use the CICS-supplied utility, DFHJUP, to print journal files. For more guidance information about the CICS-supplied utilities, refer to the *CICS/ESA Operations and Utilities Guide*.

The structure and contents of journal records

DL/I and SMF records

The following description does not apply to journal records written on behalf of DL/I, nor to those written to an SMF data set. These are described on page 535.

Journal data sets are usually specified as of undefined record type. They are then constructed by the journal control program (JCP) to correspond to the layout of variable-length blocked records, where each block and every record within the block begins with a 4-byte 'LLbb' length field. Each resultant block on a journal data set contains the following logical records:

1. One journal control label record.

The journal control label record provides global CICS information, such as the block number, the CICS run start time, and the journal identification.

2. One or more journal records.

These records follow the journal control label record, and contain all the information that has been written for the different CICS tasks. These records are variable length. The number in a particular block depends on the length of data to be logged, on the size of the two journal buffers specified for the journal in the journal control table, and on the frequency of writes. Writes occur because they have been explicitly forced, or because the current buffer is full.

Figure 44 shows a typical layout of a block on a journal data set.

LL bb	LL bb	Journal Control Label Record ('8045')	
LL bb	Journal Record 1		LL bb
Journal Record 2			
	LL bb	Journal Record 3	

Figure 44. CICS journal block layout

The DSECT DFHJCRDS describes all of the journal records. DFHJCRDS includes the 8-byte storage accounting field, which precedes the journal records if they are read by CICS, but which is not present if the journal is read by an offline program. To include DFHJCRDS in an offline program, code DFHJCR CICSYST=YES. This causes the DSECT to be included without the CICS storage accounting area.

The system header

The first 10 bytes of every journal record, including the journal control label record (but excluding DL/I and SMF records) is the system header.⁸ The system header consists of the fields shown in Table 115.

Table 115. Format of the system header for journal records

Field name in DFHJCRDS DSECT	Field size in bytes	Format	Contents
JCRBA	EQU	*	Label for start of journal records
JCRLL	2	Halfword binary	Length of record
JCRBB	2	Binary zeros	Not used
JCRSTRID	2	Hexadecimal	System-type ID
JCRUTRID	2	Hexadecimal	User-type ID
JCRLRN	2	Packed decimal	Record number within block.

The field JCRSTRID (the system-type ID) and the field JCRUTRID (the user-type ID) allow you to distinguish those journal records output by CICS (by such features as automatic journaling), from those issued by direct user requests.

For CICS journal requests, JCRUTRID contains binary zeros, and JCRSTRID contains a 1-byte function code followed by a 1-byte module code. The function code tells you which function was being journaled, and the module code shows which module caused the record to be written. Valid settings of these codes are contained in the member DFHFMIDS of the CICS assembler-language macro library. Figure 45 on page 525 shows the valid function identifiers, and Figure 46 on page 529 shows the valid module identifiers.

For user journal requests, JCRSTRID always contains binary zeros, and JCRUTRID contains the 2-byte hexadecimal code specified by the JTYPEID keyword of the JOURNAL request in the application program.

⁸ The formats of journal records written on behalf of DL/I, and those written to an SMF data set, are described on page 535.

```

*****
* * *
* * * FUNCTION AND MODULE IDENTIFIERS * * *
* * * (SEE FOLLOWING DSECTS: DFHDWEDS,DFHJCADS,DFHJCR) * * *
* * *
*****
* * FUNCTION IDENTIFIERS * *
*****
*
* X'01' THRU X'7F' ARE RESERVED FOR DL/I *
* X'20' PLUS X'8-' ...USE FOR AUTOMATIC JOURNALING *
* X'40' PLUS X'8-' ...USE FOR AUTOMATIC LOGGING *
* X'E0' THROUGH X'FF' ARE RESERVED FOR SYNCPOINT LOGGING *
* (MUST BE PRESENT IN 'LOGGABLE' DWE'S) *
*
*****
* * JOURNAL CONTROL * *
*****
FIDJCLAB EQU X'80' ...JOURNAL CONTROL LABEL
* RECORD (DFHJCR) *
*****
* * DYNAMIC BACKOUT FUNCTION IDENTIFIERS:- * *
*****
FIDDBOFL EQU X'80' OVERFLOW DYNAMIC LOG RECORD
FIDDBCHN EQU X'81' CHAIN DYNAMIC LOG RECORD
*****
* * FILE CONTROL * *
*****
FIDALOG EQU X'40' ...AUTOMATICALLY LOGGED
FIDAJRN EQU X'20' ...AUTOMATICALLY JOURNALED
FIDMASS EQU X'10' ...MASSINSERT REQ. (FIDFCWA ONLY)
* PLUS ONE OF... *
FIDFCRO EQU X'80' ...FILE CONTROL READ-ONLY
FIDFCRU EQU X'81' ...FILE CONTROL READ-UPDATE
FIDFCWU EQU X'82' ...FILE CONTROL WRITE-UPDATE
FIDFCWA EQU X'83' ...FILE CONTROL WRITE-ADD
FIDFCWAC EQU X'84' ...FILE CONTROL WRITE-ADD-COMPLETE
FIDFCWD EQU X'86' ...FILE CONTROL WRITE DELETE
FIDFCBOF EQU X'88' ...BACKOUT FAILED LOG RECORD
FIDFCDSN EQU X'8F' ...DSNAME RECORD
*
* NOTE THAT FID* VALUES (AS ABOVE) ARE OFTEN USED BOTH TO *
* IDENTIFY THE FUNCTION OF THE DWE AND THE FUNCTION OF THE *
* LOG RECORD. IN THE CASE OF THE FIDFC* EQU'S ABOVE, THEY *
* ARE USED FOR LOG RECORDS ONLY. THOSE BELOW APPLY ONLY *
* TO DWE'S *
*
FIDFCVWA EQU X'80' THIS DWE ADDRESSES A VSWA.
FIDFCRVY EQU X'40' THIS DWE IS ASSOCIATED WITH A
RECOVERABLE CHANGE.

```

Figure 45 (Part 1 of 4). Journal function identifiers

structure of journal records

```

*****
*          TRANSIENT DATA FUNCTION IDENTIFIERS:-          *
FIDTDIT EQU X'F1'          TD DESTINATION'S INPUT TASK
FIDTDOT EQU X'F2'          TD DESTINATION'S OUTPUT
*          TASK          *
FIDTDPT EQU X'F3'          TD DESTINATION'S PURGE TASK
*          EQU X'08'          ...DYNAMIC BACKOUT MASK          *
          RESERVED
FIDTDPLP EQU X'81'          TD PHYSICAL 'FIRST PUT' LOG
FIDTDPGT EQU X'82'          TD PHYSICAL 'GET' LOG
FIDTDPRL EQU X'83'          TD PHYSICAL QUEUE ZERO LOG
*          - REUSE=YES          *
FIDTDPLG EQU X'84'          TD PHYSICAL 'PURGE' LOG
FIDTDPQZ EQU X'85'          TD PHYSICAL QUEUE ZERO LOG
*          - REUSE=NO          *
*****
*          TEMPORARY STORAGE FUNCTION IDENTIFIERS          *
FIDTSAL EQU X'40'          AUTOMATIC LOGGING MASK
*          EQU X'08'          ...DYNAMIC BACKOUT MASK          *
          RESERVED
FIDTSUPD EQU X'80'          ..TEMPORARY STORAGE UPDATE
FIDTSPRI EQU X'F2'          ..TEMPORARY STORAGE
*          PURGE/RELEASE          *
FIDTSPUT EQU X'F4'          ..TEMPORARY STORAGE
*          PUT/PUTQ          *
*****
*          SPECIAL FEATURES FUNCTION IDENTIFIERS          *
FIDPSOPC EQU X'80'          CONTINOUS LOGICAL SPOOLOPEN
FIDPSWRC EQU X'81'          CONTINOUS LOGICAL SPOOLWRITE
FIDPSCLC EQU X'82'          CONTINOUS LOGICAL SPOOLCLOSE
FIDPSOPS EQU X'83'          STANDARD SPOOLOPEN
*****
*          INTERVAL CONTROL FUNCTION IDENTIFIERS          *
FIDICPDF EQU X'50'          INTERVAL CONTROL PUT,DEFER
FIDICRGT EQU X'80'          RESTART GET.
*          EQU X'08'          BACKOUT MASK, RESERVED          *
*****
*          PROGRAM CONTROL FUNCTION IDENTIFIERS          *
FIDPCPPT EQU X'80'          PC REPLACE PPT DEFINITION
*****
*          TASK CONTROL FUNCTION IDENTIFIERS          *
FIDKCPCT EQU X'80'          KC REPLACE PCT DEFINITION
FIDKCPFT EQU X'81'          KC REPLACE PFT DEFINITION
FIDKCPCA EQU X'82'          KC ADD PCT DEFINITION
FIDKCPFA EQU X'83'          KC ADD PFT DEFINITION
FIDKPCPD EQU X'84'          KC DELETE PCT DEFINITION
FIDKCPFD EQU X'85'          KC DELETE PFT DEFINITION

```

Figure 45 (Part 2 of 4). Journal function identifiers

```

*****
*          ACTIVITY KEYPOINT FUNCTION IDENTIFIERS:-          *
FIDAKS  EQU  X'80'          ACTIVITY KEYPOINT: START.
FIDAKE  EQU  X'81'          ACTIVITY KEYPOINT: END.
*       EQU  X'82'          RESERVED - WAS FIDKPTCA          *
*       EQU  X'83'          RESERVED - WAS FIDKPDCT          *
*       EQU  X'84'          RESERVED - WAS FIDKPTCR          *
*       EQU  X'85'          RESERVED - WAS FIDKPTST          *
FIDAKM  EQU  X'86'          ACTIVITY KEYPOINT: MIDDLE.
*****
          SPACE 1
*****
*          SYNCPOINT FUNCTION IDENTIFIERS:-          *
FIDSPUOW EQU  X'EF'          START OF UNIT OF WORK
FIDSPLUC EQU  X'F0'          LUC RELATED FUNCTION
FIDLSOSP EQU  X'F1'          LOGICAL START OF SYNCPOINT
FIDLEOTK EQU  X'F2'          LOGICAL END OF TASK
FIDPEOTK EQU  X'F3'          PHYSICAL END OF TASK
FIDBEOTK EQU  X'F4'          BAD END OF TASK LOG RECORD
FIDSPR  EQU  X'F5'          SPR LOG RECORD
FIDBEOSP EQU  X'F6'          BAD END OF SYNCPOINT LOG
*                               RECORD          *
FIDRSQ  EQU  X'F7'          REMOTE SESSION QUALIFIER
FIDSPRMI EQU  X'F8'          RMI DWE. (ALSO USED AS A TEST MASK..*
                               ..FOR THE FOLLOWING RMI CODES.   *
                               ...
FIDSPPTC EQU  X'F9'          RMI PREPARE
FIDSPPTB EQU  X'FA'          RMI ABOUT TO BACKOUT
FIDSPRSC EQU  X'FB'          RMI RESYNC COMMITTED
FIDSPRSB EQU  X'FC'          RMI RESYNC BACKED OUT
FIDSPFGT EQU  X'FD'          RMI FORGET
FIDSPLTC EQU  X'FE'          RMI 'LOST TO COLD START'
FIDSPNID EQU  X'FF'          RMI 'NOT IN DOUBT'
*****
          SPACE 1
*****
*          RECOVERY CONTROL FUNCTION IDENTIFIERS:-          *
FIDRCFWD EQU  X'F0'          FORWARD RECOVERY RECORD
FIDRCBWD EQU  X'80'          BACKOUT RECORD
*****
          SPACE 1
*****
*          BMS FUNCTION IDENTIFIERS:-          *
FIDBMPM  EQU  X'81'          ...BMS - PARTIAL MESSAGE ON
*                               TEMPORARY STORAGE          *
FIDBMODS EQU  X'82'          ...BMS - OPEN DATA SET ON
*                               BATCH LU          *

```

Figure 45 (Part 3 of 4). Journal function identifiers

structure of journal records

```

*****
*                TERMINAL CONTROL FUNCTION IDENTIFIERS                *
*                                                                 *
FIDTCML EQU X'F0'          SYNCPOINT - LOG SEQUENCE
*                                                                 *
*                            NUMBERS                                *
*                            CAN BE OR'ED WITH ANY OF              *
*                            THE FOLLOWING THREE FIELDS:           *
FIDTCDWL EQU X'01'        ...DEFERRED WRITE DATA
FIDTCFMH EQU X'02'        ...+ FUNCTION MANAGEMENT
*                            HEADER                                 *
FIDTCDIP EQU X'04'        ...+ DIP REQUEST
*                                                                 *
*                            ...DYNAMIC BACKOUT MASK              *
*                            RESERVED                               *
FIDTCAL EQU X'40'        AUTOMATIC LOGGING MASK...
FIDTCAJ EQU X'20'        AUTOMATIC JOURNALING MASK..
*                            ...THE ABOVE 2 PLUS 1 OF FOLLOWING SET *
FIDTCTL EQU X'80'        ...SEQUENCE NUMBER ONLY
*                            (LOG ONLY)                            *
FIDTCIM EQU X'81'        ...INPUT MESSAGE (LOG AND
*                            JOURNAL)                              *
FIDTCOM EQU X'82'        ...OUTPUT MESSAGE (JOURNAL
*                            ONLY)                                 *
FIDTCWP EQU X'83'        ...WRITE WAS PURGED (LOG
*                            ONLY)                                *
FIDTCPRR EQU X'84'        ...POSITIVE RESPONSE
*                            RECEIVED (LOG ONLY)                  *
FIDTCIMF EQU X'85'        ...INPUT MESSAGE (W/FMH,
*                            LOG AND JOURNAL)                      *
FIDTCOMN EQU X'86'        ...OUTPUT MESSAGE, (W/O
*                            FMH, JOURNAL ONLY)                   *
FIDTCO  EQU X'87'        ...OUTPUT MESSAGE, FMH,
*                            CCOMPL=NO                           *
FIDTCO  EQU X'88'        ...OUTPUT MESSAGE, W/O FMH,
*                            ...CCOMPL=NO                         *
FIDTCUA EQU X'89'        ...INITIAL TCT USER AREA
FIDTCEIB EQU X'8A'        ...INITIAL EXEC COMM AREA
FIDTCIMN EQU X'8B'        INPUT MSG, NO FMH, COMPLETE
FIDTCINN EQU X'8C'        INPUT MSG, NO FMH, INCOMPLETE
*****
*                TABLE BUILDER SERVICES FUNCTION IDENTIFIERS        *
*                                                                 *
FIDBSDOP EQU X'80'        TBS DWE IS 'OPEN'
FIDBSDCL EQU X'81'        TBS DWE IS 'CLOSED'
*****
*                FRONT END PROGRAMMING INTERFACE IDENTIFIERS        *
*                                                                 *
FIDFEPIN EQU X'F0'        FEPI INBOUND DATA   API <--- FEPI
FIDFEPOU EQU X'F1'        FEPI OUTBOUND DATA  API ---> FEPI
*****
*                GENERAL PURPOSE SUBTASK IDENTIFIERS                *
*                                                                 *
FIDSKDF EQU X'80'        ...SK - DEFAULT
*****

```

Figure 45 (Part 4 of 4). Journal function identifiers


```

*****
* *           M O D U L E   I D E N T I F I E R S           * *
*****
*
*           (MAY BE X'01'-->X'FF'.)
*
MODIDKC EQU X'03'           ...TASK CONTROL
MODIDPC EQU X'04'           ...PROGRAM CONTROL
MODIDSC EQU X'05'           ...STORAGE CONTROL
MODIDDC EQU X'07'           ...DUMP CONTROL
MODIDIC EQU X'08'           ...INTERVAL CONTROL
MODIDTC EQU X'10'           ...TERMINAL CONTROL
MODIDFC EQU X'11'           ...FILE CONTROL
MODIDTD EQU X'12'           ...TRANSIENT DATA
MODIDTS EQU X'13'           ...TEMPORARY STORAGE
MODIDFCJ EQU X'14'          ...FILE CONTROL JOURNALING
MODIDIRC EQU X'37'           ...IRC INTERFACE
MODIDDL EQU X'39'           ...DL/I INTERFACE
MODIDBM EQU X'40'           ...BASIC MAPPING
MODIDJC EQU X'45'           ...JOURNAL CONTROL
MODIDDB EQU X'46'           ...DYNAMIC BACKOUT PROGRAM
MODIDVC EQU X'47'           ...VOLUME CONTROL PROGRAM
MODIDPS EQU X'53'           ...SPECIAL FEATURES
MODIDKPP EQU X'54'          ...KEYPOINT PROGRAM
MODIDBI EQU X'55'           ...BUILT-IN FUNCTIONS
MODIDAKP EQU X'58'          ...ACTIVITY KEYPOINT PROG
MODIDSPP EQU X'59'          ...SYNCPOINT PROGRAM
MODIDEIP EQU X'5A'          ...EXEC INTERFACE PROGRAM
MODIDTMP EQU X'5B'          ...TABLE MANAGER
MODIDSKP EQU X'5C'          ...SUBTASK MANAGER
MODIDRCP EQU X'CE'          ...RECOVERY CONTROL PROGRAM
MODIDTBS EQU X'E3'          ...TABLE BUILDER SERVICES
MODIDTOR EQU X'EF'          ...TERMINAL OBJECT RESOLUTION
MODIDUSR EQU X'FF'          RESERVED FOR USER SYNCPOINT
*                               SUPPORT
*****

```

Figure 46. Journal module identifiers

The journal control label record

The first record in each block of journal records is a label record. The label record, including the system header, is 42 bytes long. The format of the journal control label record (excluding the system header) is shown in Table 116.

Table 116. Format of the journal control label record

Field name in DFHJCRDS DSECT	Field size in bytes	Format	Contents
JCLRJFID	1	Binary	Journal ID (1–99)
JCLRBLKN	3	Packed decimal	Block number (1–n) in this data set
JCLRVCDD	4	Packed decimal	Volume creation date (0cyydd+)
JCLRVSNN	2	Packed decimal	Volume sequence number within run (nnn+) (only 1 or 2 for disk journals)
JCLRLBW	4	Binary (disk)	Relative TTR0 of previous block
JCLRTBAL	2	Binary (disk)	Track-balance from previous block (disk journals)
JCLRTIME	4	Packed decimal	Time block written (hhmmsss+)
JCLRRST	4	Packed decimal	Run start time (hhmmsss+)
JCLRDATE	4	Packed decimal	Date block written (0cyydd+)
JCLRSEQ	4	Binary	Journal block sequence number

Other journal records

All journal records other than the journal control label record—that is, all those written in response to EXEC CICS WRITE JOURNALNUM commands—have one through three variable-length segments in the following order after the system header.

Note: This does not apply to DL/I or SMF journal records, which have their own format, described on page 535.

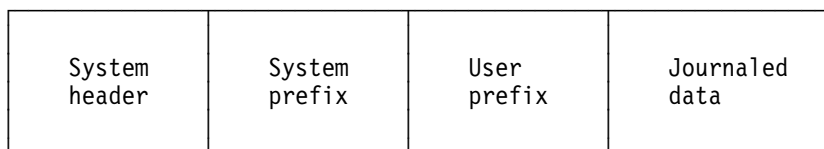


Figure 47. CICS journal record format

Each of these three segments is described in the sections that follow.

The system prefix

The system prefix is variable in length, and it identifies the origin of the record. The system prefix contains at least the data shown in Table 117.

<i>Table 117. Data contained in the system prefix</i>			
Field name in DFHJCRDS DSECT	Field size in bytes	Format	Contents
JCSPBA	0	-	Label for system prefix begin address
JCSPLL	2	Halfword binary	Length of system prefix
JCSPFS	3	Binary	Flags
<p>Note: The first two bytes of JCSPFS are reserved. The third byte is field JCSPF1, and the settings are:</p>			
JCSPUP		EQU X'01'	User prefix present in record
JCSPSOTK		EQU X'02'	Physical start-of-task ⁹
JCSPLSTK		EQU X'04'	Logical start-of-task ⁹
JCSPRRIF		EQU X'08'	DFHRUP record-in-flight flag
JCSPMIDT		EQU X'10'	Output message in doubt
JCSPEMER		EQU X'20'	Written by emergency restart
JCSPDSP		EQU X'80'	DL/I checkpoint
JCSPTASK	3	Packed decimal	Task number
JCSPTIME	4	Packed decimal	Time of request (hhmmsss+)
JCSPTRAN	4	Characters	Transaction identification
JCSPTERM	4	Characters	Terminal identification (or binary zeros)
JCSPREA		EQU *	Label for end of system prefix common root.

Additional system prefix data: For some CICS journal requests, additional data is included in the system prefix to identify more specifically the originator of the request. This extra data follows the common fields of the system prefix, and is usually variable in length; hence the need for the length field JCSPLL at the start of the system prefix. All the following have their own prefix layout, and these are described, for the purposes of diagnosis and recovery, in the *CICS/ESA Data Areas* manual.

- Activity keypointing.
- File control.
- Front End Programming Interface (FEPI). (For details of FEPI, see the *CICS/ESA Front End Programming Interface User's Guide*.)
- Recovery control.
- Remote session qualifier.
- Resource manager interface.
- SPR (intersystem communication).
- Terminal control.

⁹ The setting of this bit is meaningful for system journal records only. You should ignore its setting for user journal records.

The user prefix

The user prefix is optional, and is placed in a journal output record next to the system prefix, in response to the PREFIX option of the EXEC CICS WRITE JOURNALNUM command. As with the system prefix, the user prefix always begins with a halfword binary length field; the user prefix data follows the halfword. For journal records that include a user prefix, the flag byte JCSPF1 of the system prefix has the indicator bit JCSPUP set to '1'.

Journalled data

The final segment of a journal record is the journalled data. CICS does not generate a length field for the data portion of the record. If you want a length field for the data, you must include it in the data. Alternatively, you can compute the length of the data portion of a journal record by taking the length of the system header (10 bytes), plus the length of the system prefix (JCSPLL), plus the length of the user prefix (in the field, if any, defined by yourself), and subtracting the total from the length of the journal record (JCRL).

Not all journal records contain journalled data. For example, syncpoint records contain no journalled data.

Identifying records for the start and end of tasks or LUWs

You can identify records written by the syncpoint program to mark the start and end of tasks, and the start and end of LUWs, by examining the value of JCRSTRID, together with the flags set in the JCSPFS field of the system prefix. See Table 118 and Figure 48 on page 534.

Table 118. Identifying syncpoint events

Syncpoint event	JCRSTRID value	System prefix flag
Physical start-of-task	X'EF59'	JCSPSOTK
Logical start-of-task (start-of-LUW)	X'EF59'	JCSPLSTK
Logical end-of-task (end-of-LUW)	X'F259'	N/A
Physical end-of-task	X'F359'	N/A

Note: If the first protected resource accessed by a task or LUW is a local DL/I program specification block (PSB), CICS writes a special record to the system log. This record follows the X'EF59' start-of-task or start-of-LUW record, and precedes the DL/I logged record for the database access. You can identify the special record as follows:

- JCRSTRID has the value X'0000'.
- The 8-byte character string 'DLI SOTK' is at the end of the system prefix common root, addressed by JCSPREA.

The special record is needed only for shared database support. (Shared database support allows a DL/I application program running in a batch region to access a database owned by a CICS region running with local DL/I. It allows the database to be updated concurrently from online and from batch, without loss of integrity.) If the special record is written because the database is to be accessed by a batch program, it is followed by the 35-byte identifier of the appropriate batch job.

Identifying a record for an unfinished task or LUW

You can tell if a record is for an unfinished task by the presence or absence of a physical end-of-task record (JCRSTRID=X'F359') for the same task. The end of each LUW is indicated by a logical end-of-task record (JCRSTRID=X'F259'). In Figure 48 on page 534, if a system failure occurred before the completion of task 33, no physical end-of-task record would be written to the journal. In the restart data set, written by the recovery utility program (DFHRUP) during emergency restart, records for in-flight tasks are identified by the flag JCSPRRIF in the system prefix.

Identifying in-doubt messages at system termination

You can identify in-doubt messages at system termination only if the task is invoked from a VTAM terminal, and the installed transaction definition includes the MSGPOPT|MSGPREQ=PROTECT option. This causes logging (for each LUW) of the first input, last output, and positive-response-received messages. In this case, the deferred write record (JCRSTRID=X'F110') should be followed by a positive-response-received record (JCRSTRID=X'C410'). If not, the message was in doubt when the system terminated. On the restart data set (and in the message cache) the deferred write record, if in doubt, has the JCSPMIDT flag set on. (For information about message caches, see the *CICS/ESA Recovery and Restart Guide*.)

structure of journal records

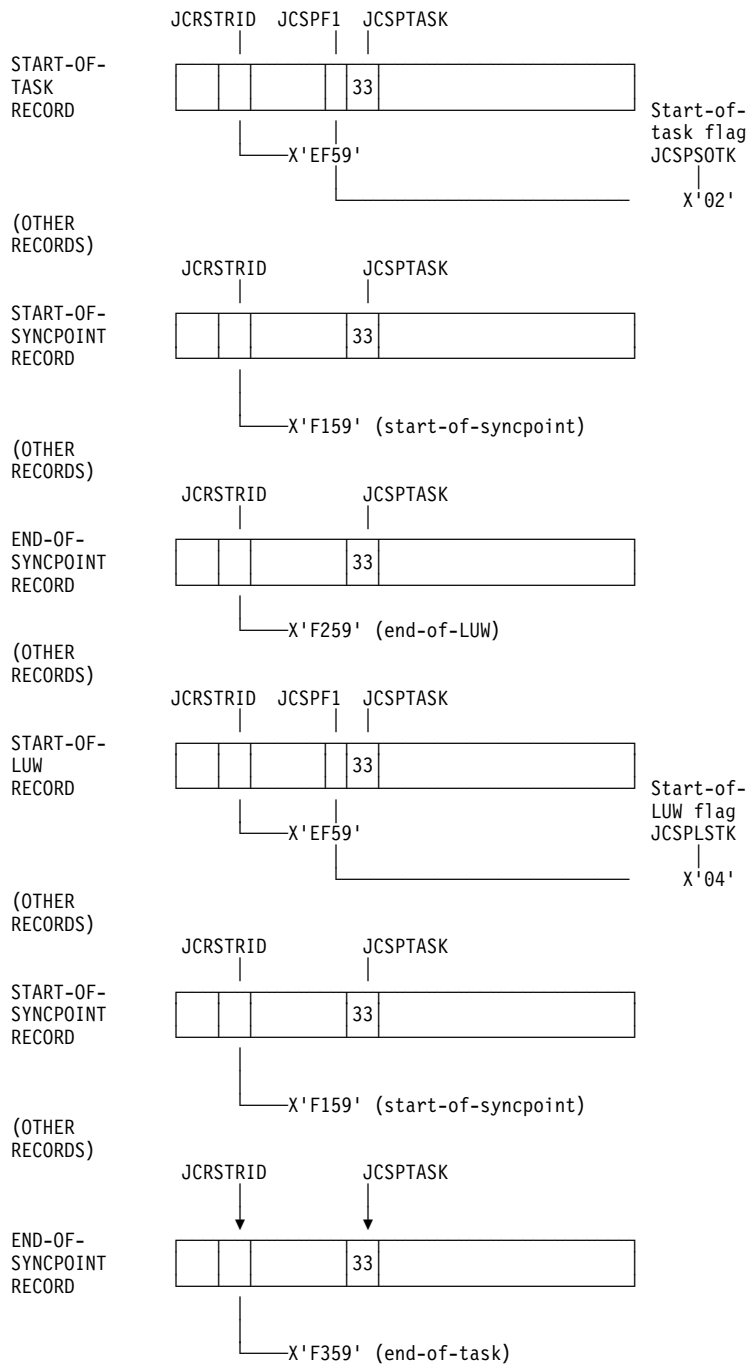


Figure 48. Journal records, showing task and LUW delimiters. This figure shows physical and logical start- and end-of-task records, with the setting of JCSPSOTK and JCSPSTK flags to indicate start-of-task and start-of-LUW in a task with two LUWs.

Journal records for DL/I

Records written to the CICS journal on behalf of DL/I have the format shown in Table 119.

Table 119. The format of journal records for DL/I

Field name in DFHJCRDS DSECT	Field size in bytes	Format	Contents
JCRLL	2	Halfword binary	Length of record
JCRBB	2	Binary zeros	Not used
None	V	Data	Variable-length DL/I data

Note: If the first byte of the data following field JCRBB has a value in the range X'01' through X'7F', the journal records are written to DL/I. To analyze a DL/I log record, refer to the IMS/ESA documentation.

Format of journal records written to SMF

This section describes the format of journaling records that are written to an SMF data set. You need this information if you write your own program to analyze the data. The three components of the journaling record are an SMF header, an SMF product section, and a CICS data section.

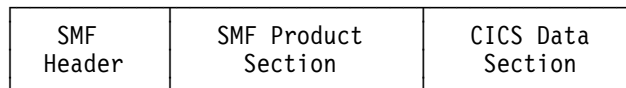


Figure 49. Format of an SMF type 110 journaling record

The SMF header describes the system creating the output.

The SMF product section identifies the subsystem to which the journaling data relates. Both the SMF header and the SMF product section can be mapped by the DSECT JCSMFDS, which you can generate using the DFHJCSMF macro as follows:

```
JCSMFDS DFHJCSMF PREFIX=SMF
```

The label 'JCSMFDS' is the default DSECT name, and SMF is the default PREFIX value, so you could also generate the DSECT simply by coding DFHJCSMF. The format of the JCSMFDS DSECT is shown in Figure 50 on page 536.

Note: The copy book DFHSMFDS is also provided and can be used to map the SMF header and the SMF product sections of all three subtypes of SMF 110 records written by CICS journaling, CICS monitoring, and CICS statistics.

The CICS data section contains a variable-length blocked journal record. Figure 44 on page 523 shows a typical layout of a blocked journal record.

structure of journal records

```

*          START OF THE SMF HEADER
*
JCSMFDS  DSECT
SMFFLG  DS   X          OPERATING SYSTEM INDICATOR (see note)
SMFRTY  DC   X'6E'     RECORD 110 FOR CICS
SMFTME  DS   XL4       TIME RECORD MOVED TO SMF
SMFDTE  DS   XL4       DATE RECORD MOVED TO SMF
SMFSID  DC   CL4'CICS' SYSTEM IDENTIFICATION
SMFSSI  DC   CL4'CICS' SUBSYSTEM IDENTIFICATION
SMFSTY  DS   XL2       RECORD SUBTYPE - X'0000' FOR JOURNALING
SMFTRN  DS   XL2       NUMBER OF TRIPLETS
          DS   XL2       RESERVED
SMFAPS  DS   XL4       OFFSET TO PRODUCT SECTION
SMFLPS  DS   XL2       LENGTH OF PRODUCT SECTION
SMFNPS  DS   XL2       NUMBER OF PRODUCT SECTIONS
SMFASS  DS   XL4       OFFSET TO DATA SECTION
SMFASL  DS   XL2       LENGTH OF DATA SECTION
SMFASN  DS   XL2       NUMBER OF DATA SECTIONS
*
*          THIS CONCLUDES THE SMF HEADER
*
*          START OF THE SMF PRODUCT SECTION
*
SMFPSRVN DS   XL2       RECORD VERSION (CICS)
SMFPSPRN DS   CL8       PRODUCT NAME (GENERIC APPLID)
SMFSPSPN DS   CL8       PRODUCT NAME (SPECIFIC APPLID)
SMFPSMFL DS   XL2       RECORD MAINTENANCE INDICATOR
          DS   XL2       RESERVED
SMFPSRSN DS   PL4       RECORD-NUMBER WITHIN JOURNAL
SMFPSJID DS   X         JOURNAL IDENTIFIER
SMFPSBKN DS   PL3       RECORD-NUMBER WITHIN DATA SET
SMFPSLBW DS   XL4       LAST-RECORD ADDRESS
          *             (FORMAT IS TTR0)
SMFPSBAL DS   AL2       TRACK BALANCE IN BYTES
*
          DS   XL46      RESERVED
SMFPSJBN DS   CL8       JOBNAME
SMFPSRSD DS   XL4       JOB DATE
SMFPSRST DS   XL4       JOB TIME
SMFPSUIF DS   CL8       USER IDENTIFICATION
SMFSPDN  DS   CL8       OPERATING SYSTEM PRODUCT LEVEL
SMFSEND  EQU   *        END OF SMF PRODUCT SECTION
*
*          THIS CONCLUDES THE SMF PRODUCT SECTION

```

Figure 50. Format of the SMF header and product section for journaling records

Note: CICS sets only the subsystem-related bits of the operating system indicator flag byte in the SMF header (SMFFLG). SMF sets the remainder of the byte according to the operating system level and other factors. For an explanation of the setting of the other bits, refer to the *MVS/ESA System Management Facilities* manual.

Notes on CICS journaling

This section contains brief notes on aspects of CICS journaling that are covered fully in other books, or in other parts of this book.

Automatic journal archiving

CICS/ESA 4.1 provides automatic journal archiving: when a journal data set fills up, CICS can automatically archive it while the second disk data set is being used. You can find more information about automatic journal archiving in the *CICS/ESA Recovery and Restart Guide*.

The journaling user-replaceable programs

For guidance information about the journaling user-replaceable programs, refer to Chapter 15, “The user-replaceable journaling programs” on page 493.

The global user exits in DFHJCP

Global user exit XJCWR in the journaling control program is invoked after a journal record is written to the journal buffer, but before it is written to the journal data set. At this exit, the addresses of the journal record and of the JCT entry are available to your exit program.

Exit XJCWB is invoked before each physical WRITE to each non-SMF journal. It could be used (for example) to pass copies of CICS log records to another system, to aid recovery of the prime-site system after a disaster.

For more information about global user exits in general, and about the journaling exits in particular, refer to Chapter 1, “Global user exit programs” on page 3.

notes on journaling

Chapter 19. CICS monitoring

This chapter describes the monitoring facilities of CICS/ESA 4.1. It is divided into the following sections:

1. “**Introduction to CICS monitoring**” describes the classes of monitoring data, event-monitoring points, and the use of the monitoring control table.
2. “**CICS monitoring record formats**” on page 547 describes the formats of CICS monitoring SMF type 110 records.
3. “**Data produced by CICS monitoring**” on page 557 lists all the system-defined data that can be produced by CICS monitoring.

Introduction to CICS monitoring

CICS monitoring collects data about the performance of all user- and CICS-supplied transactions during online processing for later offline analysis. The records produced by CICS monitoring are of the MVS System Management Facility (SMF) type 110, and are written to an SMF data set.

Note: Statistics records and some journaling records are also written to the SMF data set as type 110 records. You might find it particularly useful to process the statistics records and the monitoring records together, because statistics provide resource and system information that is complementary to the transaction data produced by CICS monitoring. The contents of the statistics fields, and the procedure for processing them, are described in Chapter 20, “CICS statistics” on page 579.

Monitoring data is useful both for performance tuning and for charging your users for the resources they use.

The classes of monitoring data

Three types, or “classes”, of monitoring data can be collected. These are **performance class** data, **exception class** data, and **SYSEVENT** data.

Performance class data is detailed transaction-level information, such as the processor and elapsed time for a transaction, or the time spent waiting for I/O. At least one performance record is written for each transaction that is being monitored. See “Performance class monitoring data” on page 540 for further information.

Exception class data is information about exceptional conditions suffered by a transaction, such as queuing for file strings, or waiting for temporary storage. This data highlights possible problems in system operation. There is one exception record for each exception condition.

SYSEVENT class data is a special kind of transaction timing information. More information about SYSEVENT data is on page 545.

You can choose which classes of monitoring data you want to be collected. How to do this is described in “Controlling CICS monitoring” on page 546.

Performance class monitoring data

CICS performance class monitoring data is collected at system-defined event-monitoring points (EMPs) in the CICS code. You cannot relocate these monitoring points, but you can create additional ones, at which user-defined performance data can be gathered.

Coding additional event-monitoring points

If you want to gather more performance class data than is provided at the system-defined EMPs, you can code additional EMPs in your application programs. You could use these additional EMPs to count the number of times a certain event occurs, or to time the interval between two events, for example. If the performance class was active when a transaction was started, but was not active when a user EMP was issued, the operations defined in that user EMP would still be executed on that transaction's monitoring area. The DELIVER option would result in a loss of data at this point, because the generated performance record cannot be output while the performance class is not active. If the performance class was not active when a transaction was started, the user EMP would have no effect.

To code user EMPs in your application programs, you use the EXEC CICS MONITOR command. For programming information about this command, see the *CICS/ESA Application Programming Reference* manual.

Additional EMPs are provided in some IBM program products, such as DL/I. From a CICS point of view, these are like any other user-defined EMP. EMPs in user applications and in IBM program products are identified by a decimal number. The numbers 1 through 199 are available for EMPs in user applications, and the numbers 200 through 255 are for use in IBM program products. The numbers can be qualified with an entry name, so that you can use each number more than once. For example, 'ENTRYA.4', 'ENTRYB.4', and '4' identify three different EMPs. Furthermore, any counts, clocks, or byte-strings updated at one of them are different objects from those updated at any of the others. If you do not specify an entry name, CICS assumes the default of 'USER'.

For each EMP that you code in an application program, there must be a corresponding monitoring control table (MCT) definition, with the same entry name and identification number as the EMP that it describes. (The following sections refer to the combination of entry name and identification number as an "empid".)

If you want to record the same type of data for different transactions, you can code the same empids in several application programs. This causes similar fields in the corresponding transaction performance records to be updated.

You do not have to assign empids to system-defined EMPs, and you do not have to code MCT entries for them.

The monitoring control table (MCT)

You use the monitoring control table (MCT):

- To tell CICS about the EMPs that you have coded in your application programs and about the data that is to be collected at these points. See "DFHMCT TYPE=EMP" on page 541.
- To tell CICS that you want specific system-defined performance data not to be recorded during a particular CICS run. See "DFHMCT TYPE=RECORD" on page 541.

Full details of the DFHMCT macros are provided in the *CICS/ESA Resource Definition Guide*, and you should refer to that book when reading the following sections.

DFHMCT TYPE=EMP

There must be a DFHMCT TYPE=EMP macro definition for every user-coded EMP. This macro has an ID operand, whose value must be made up of the ENTRYNAME and POINT values specified on the EXEC CICS MONITOR command. The PERFORM operand of the DFHMCT TYPE=EMP macro tells CICS which user count fields, user clocks, and character values to expect at the identified user EMP, and what operations to perform on them.

Note that, in a single run of CICS, the format of all performance records is identical, and that the length of records increases relative to the number of data fields in the user EMPs defined in the MCT.

The maximum amount of user data that can be added to performance records is 16384 bytes. The user data is divided into user areas. Each user area is defined by coding an entry name qualifier on the ID operand of the DFHMCT TYPE=EMP macro. If you code the same entry name when defining multiple EMPs, all the EMPs operate on fields in the same user area. Correspondingly, by coding different entry names you can append multiple user areas to the monitoring records. Provided that the overall maximum of 16384 bytes is not exceeded, each user area can contain:

- 0 through 256 counters
- 0 through 256 clocks
- A single 8192-byte character string.

Each user area is uniquely referenced by its entry name. For example:

```
DFHMCT TYPE=EMP, ID=ENTRYA.1, PERFORM=...
DFHMCT TYPE=EMP, ID=ENTRYA.2, PERFORM=...
DFHMCT TYPE=EMP, ID=ENTRYB.1, PERFORM=...
DFHMCT TYPE=EMP, ID=ENTRYB.1, PERFORM=...
DFHMCT TYPE=EMP, ID=1, PERFORM=...
```

In the above examples, in addition to the system-defined performance fields, three user areas, 'ENTRYA', 'ENTRYB', and 'USER', are defined (if no entry name is specified, the default is 'USER'). If the application codes an EMP invocation with ENTRYNAME(ENTRYA), only the ENTRYA user area is operated on. The only operation that spans all user areas is DELIVER, which operates across the whole monitoring area.

DFHMCT TYPE=RECORD

The DFHMCT TYPE=RECORD macro allows you to *exclude* specific system-defined performance data from a CICS run. (Each performance monitoring record is approximately 572 bytes long, without taking into account any user data that may be added, or any excluded fields.)

Each field of the performance data that is gathered at the system-defined EMPs belongs to a group of fields that has a group identifier. Each performance data field also has its own numeric identifier that is unique within the group identifier. For example, the transaction sequence number field in a performance record belongs to the group DFHTASK, and has the numeric identifier '031'. Using these

identifiers, you can exclude specific fields or groups of fields, and reduce the size of the performance records.

Examples of MCT coding

The examples below show some EXEC CICS MONITOR commands with the MCT entries that must be coded for them.

Example 1

EXEC CICS MONITOR command	MCT entry
<pre>EXEC CICS MONITOR POINT(11) ENTRYNAME(PROG3)</pre>	<pre>DFHMCT TYPE=EMP, * CLASS=PERFORM, * ID=(PROG3.11), * CLOCK=(1,CLOCKA), * PERFORM=SCLOCK(1)</pre>

Example 1 shows a user clock being started by an application that is identified as PROG3. This is the eleventh EMP in this application. To prevent confusion with the eleventh EMP in another application, this EMP is uniquely identified by the empid PROG3.11. The clock that is being started is the first clock in a string, and has the identifier CLOCKA.

Example 2

EXEC CICS MONITOR command	MCT entry
<pre>EXEC CICS MONITOR POINT(12) ENTRYNAME(PROG3)</pre>	<pre>DFHMCT TYPE=EMP, * CLASS=PERFORM, * ID=(PROG3.12), * PERFORM=PCLOCK(1)</pre>

Example 2 shows the same user clock (CLOCKA) being stopped. Although this is the same clock being stopped by the same application as in example 1, it is being stopped from a different EMP. The EMP is uniquely identified by the empid PROG3.12.

Example 3

EXEC CICS MONITOR command	MCT entry
EXEC CICS MONITOR POINT(13) DATA1(address of data) DATA2(length of data) ENTRYNAME(PROG3)	DFHMCT TYPE=EMP, * CLASS=PERFORM, * ID=(PROG3.13), * PERFORM=MOVE(0,32)

Example 3 shows 32 bytes of user data being updated in the character string reserved for that purpose. The updated data starts at offset 0, and the data is not more than 32 bytes in length.

DFHMCT TYPE=EMP entries for DL/I

The following MCT entries make use of the nine event-monitoring points in the performance class used by the CICS-DL/I interface modules:

```
DFHMCT TYPE=EMP, ID=(PP,1), CLASS=PERFORM, PERFORM=SCLOCK(3)
DFHMCT TYPE=EMP, ID=(PP,2), CLASS=PERFORM, PERFORM=PCLOCK(3)
DFHMCT TYPE=EMP, ID=(PP,3), CLASS=PERFORM, PERFORM=SCLOCK(1)
DFHMCT TYPE=EMP, ID=(PP,4), CLASS=PERFORM, PERFORM=(PCLOCK(1), *
SCLOCK(2))
DFHMCT TYPE=EMP, ID=(PP,5), CLASS=PERFORM, PERFORM=(PCLOCK(2), *
MLTCNT(1,9)), *
COUNT=(1,GU,GN,GNP,GHU,GHN,GHNP,ISRT,DLET,REPL), *
CLOCK=(1,'PSB WAIT','PSB SCHED','DB CALL','DB IO')
DFHMCT TYPE=EMP, ID=(PP,6), CLASS=PERFORM, PERFORM=SCLOCK(4)
DFHMCT TYPE=EMP, ID=(PP,7), CLASS=PERFORM, PERFORM=PCLOCK(4)
DFHMCT TYPE=EMP, ID=(PP,8), CLASS=PERFORM, PERFORM=(PCLOCK(1), *
PCLOCK(2),PCLOCK(3), *
PCLOCK(4))
DFHMCT TYPE=EMP, ID=(PP,9), CLASS=PERFORM, PERFORM=PCLOCK(1)
```

These MCT entries are coded in the sample copy book DFH\$MCTL.

In the ID parameter, the following specifications are used:

- (PP,1)** Start of database call
- (PP,2)** End of database call
- (PP,3)** Start of PSB schedule
- (PP,4)** End of schedule call
- (PP,5)** Terminate PSB
- (PP,6)** Start of I/O wait
- (PP,7)** End of I/O wait
- (PP,8)** Stop all clocks after task abend in DFHDLQ
- (PP,9)** Stop clock 1 after schedule failure.

DFHMCT TYPE=EMP entries for DBCTL

The following MCT entries make use of the two event-monitoring points in the performance class used by the CICS-DBCTL interface modules:

```
DFHMCT TYPE=EMP, ID=(DBCTL.1), CLASS=PERFORM, PERFORM=(MOVE(0,256)), *
                                FIELD=(1,RMIDATA)
DFHMCT TYPE=EMP, ID=(DBCTL.2), CLASS=PERFORM, PERFORM=(DELIVER)
```

These MCT entries are coded in the sample copy book DFH\$MCTD.

The DBCTL data recorded by these event monitoring points can be mapped using the IMS macro DFSDSTA DSECT=YES, which is available in the IMS genlibs.

For more information about monitoring in DBCTL, refer to the *CICS/ESA CICS-IMS Database Control Guide*.

Exception class data

Exception class data is information on exceptional conditions suffered by a transaction. This data highlights possible problems in system operation. There is one exception record for each exception condition. Exception records are produced after each of the following conditions encountered by a transaction has been resolved:

- Wait for storage in the CDSA
- Wait for storage in the UDSA
- Wait for storage in the SDSA
- Wait for storage in the RDSA
- Wait for storage in the ECDSA
- Wait for storage in the EUDSA
- Wait for storage in the ESDSA
- Wait for storage in the ERDSA
- Wait for auxiliary temporary storage
- Wait for auxiliary temporary storage string
- Wait for auxiliary temporary storage buffer
- Wait for file string
- Wait for file buffer
- Wait for LSRPOOL string.

An exception record is created each time any of the resources covered by exception class monitoring becomes constrained by system bottlenecks. If performance data is also being recorded, it keeps a count of the number of exception records generated for each task. The exception records can be linked to the performance data by the transaction identifier in both records.

This data is intended to help you identify constraints that affect the performance of your transaction. The information is written to a SMF data set as soon as the task that was originally constrained has been released.

You can enable exception-class monitoring by coding MNEXC=ON (together with MN=ON) in the SIT. Alternatively you can use, either the CEMT command (CEMT SET MONITOR ON EXCEPT) or EXEC CICS SET MONITOR STATUS(ON) EXCEPTCLASS(EXCEPT).

The SYSEVENT class of monitoring data

CICS invokes the MVS System Resource Manager (SRM) macro SYSEVENT at the end of every transaction, to record the elapsed time of the transaction. This information is collected by SRM and can be displayed and reported by the MVS Resource Measurement Facility (RMF). It can be directed to the SMF data set, or RMF may simply produce a report of transaction activity. The collection of CICS monitoring SYSEVENT data can be switched on or off in exactly the same way as the collection of performance and exception data.

If an attempt to write to the MVS SRM results in a return code of '08', CICS monitoring retries the write up to five times (for that transaction) before signaling a failure.

MVS IEAICS member

An IEAICS member needs to be coded and placed in SYS1.PARMLIB on the MVS system. For further information about this, see the *MVS/ESA Initialization and Tuning Guide*. Reporting groups are assigned to the CICS system as a whole and to individual transactions.

How CMF SYSEVENT data is mapped to the IEAICSxx member of SYS1.PARMLIB

<i>Table 120. How CMF SYSEVENT data is mapped to IEAICSxx</i>		
SYSEVENT macro	IEAICSxx member	CICS monitoring facility data
Transaction start time	N/A	Time at which user-task attached
Subsystem name	SUBSYS=	First 4 characters of the Generic APPLID
Transaction name	TRXNAME=	Transaction ID of task
User identification	USERID=	User ID
Transaction class	TRXCLASS=	The specific APPLID of the CICS region

For more information about how to use RMF, refer to the *MVS/ESA Resource Measurement Facility (RMF), Version 4.1.1 – Monitor I & II Reference and Users Guide*. If records are directed to SMF, refer to the *MVS/ESA System Management Facilities* manual. The following example shows the additional parameters that you need to add to your IEAICS member for two MRO CICS systems:

```
SUBSYS=ACIC,RPGN=100      /* CICS SYSTEM ACIC HAS REPORTING */
  TRXNAME=CEMT,RPGN=101  /* GROUP OF 100 AND THERE ARE */
  TRXNAME=USER,RPGN=102  /* THREE INDIVIDUAL GROUPS FOR */
  TRXNAME=CSMI,RPGN=103  /* SEPARATE TRANSACTIONS */
SUBSYS=BCIC,RPGN=200     /* CICS SYSTEM BCIC HAS REPORTING */
  TRXNAME=CEMT,RPGN=201  /* GROUP OF 200 AND THERE ARE */
  TRXNAME=USER,RPGN=202  /* THREE INDIVIDUAL GROUPS FOR */
  TRXNAME=CSMI,RPGN=203  /* SEPARATE TRANSACTIONS */
```

Notes:

1. The reporting group (number 100) assigned to the ACIC subsystem reports on all transactions in that system.
2. RMF reports on an individual transaction by name only if it is assigned a unique reporting group. If multiple transactions are defined with one reporting group, the name field is left blank in the RMF reports.

How performance and exception class data is passed to SMF

Performance class records and exception class records are not written to SMF in the same way by CICS monitoring.

Performance data records are written to a performance record buffer, which is defined and controlled by CICS, as they are produced. The performance records are passed to SMF for processing when the buffer is full, when the performance class of monitoring is switched off, and when CICS itself quiesces. When Monitoring itself is deactivated or when there is an immediate shutdown of CICS, the performance records are not written to SMF and the data is lost.

Exception records are passed directly to SMF when the exception condition completes. Each exception record describes one exception condition. You can link performance records with their associated exception records by matching the value of the TRANNUM field in each type of record; each contains the same transaction number.

Controlling CICS monitoring

When CICS is initialized, you switch the monitoring facility on by specifying the system initialization parameter MN=ON. MN=OFF is the default setting. You can select the classes of monitoring data you want to be collected using the MNPER, MNEXC, and MNEVE system initialization parameters. You can request the collection of any combination of performance class data, exception class data, and SYSEVENT data. The class settings can be changed whether monitoring itself is ON or OFF. For guidance information about system initialization parameters, refer to the *CICS/ESA System Definition Guide*.

When CICS is running, you can control the monitoring facility dynamically. Just as at CICS initialization, you can switch monitoring on or off, and you can change the classes of monitoring data that are being collected. There are two ways of doing this:

1. You can use the master terminal CEMT INQ|SET MONITOR command, which is described in the *CICS/ESA CICS-Supplied Transactions* manual.
2. You can use the EXEC CICS INQUIRE and SET MONITOR commands, which are described in the *CICS/ESA System Programming Reference*.

If you activate a class of monitoring data in the middle of a run, the data for that class becomes available only for transactions that are started thereafter. You cannot change the classes of monitoring data collected for a transaction after it has started. It is often preferable, particularly for long-running transactions, to start all classes of monitoring data at CICS initialization.

CICS monitoring record formats

This section describes the formats of CICS monitoring SMF type 110 records in detail. You need this information if you write your own program to analyze the monitoring data. CICS writes three types of SMF 110 record. Each type, or subtype as it is known, can be identified using the record subtype field in the SMF header. The subtype values are as follows:

X'0000' – Journaling
 X'0001' – Monitoring
 X'0002' – Statistics.

For more information about SMF journaling records, refer to Chapter 18, “CICS journaling” on page 521. For more information about SMF statistics records, refer to Chapter 20, “CICS statistics” on page 579.

The three components of a CICS monitoring record are an SMF header, an SMF product section, and a CICS data section. Each of these is described in the sections that follow.

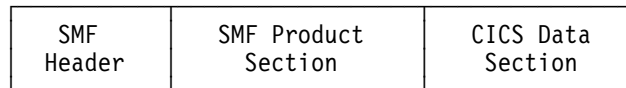


Figure 51. Format of an SMF type 110 monitoring record

SMF header and SMF product section

The SMF header describes the system creating the output. The SMF product section identifies the subsystem to which the monitoring data relates, which, in the case of CICS monitoring (and also of CICS statistics), is the CICS region. Both the SMF header and the SMF product section can be mapped by the DSECT MNSMFDS, which you can generate using the DFHMNSMF macro as follows:

```
MNSMFDS DFHMNSMF PREFIX=SMF
```

The label 'MNSMFDS' is the default DSECT name, and SMF is the default PREFIX value, so you could also generate the DSECT simply by coding:

```
DFHMNSMF
```

The MNSMFDS DSECT has the format shown in Figure 52 on page 548.

monitoring record formats

```

*          START OF THE SMF HEADER
*
MNSMFDS  DSECT
SMFMNLEN DS  XL2          RECORD LENGTH
SMFMNSEG DS  XL2          SEGMENT DESCRIPTOR
SMFMNFLG DS  X           OPERATING SYSTEM INDICATOR (see note 1)
SMFMNRTY DC  X'6E'       RECORD 110 FOR CICS
SMFMNTME DS  XL4          TIME RECORD MOVED TO SMF
SMFMNDTE DS  XL4          DATE RECORD MOVED TO SMF
SMFMNSID DS  CL4          SYSTEM IDENTIFICATION
SMFMNSSI DS  CL4          SUBSYSTEM IDENTIFICATION
SMFMNSTY DS  XL2          RECORD SUBTYPE - MONITORING USES TYPE 1
SMFMNTRN DS  XL2          NUMBER OF TRIPLETS
          DS  XL2          RESERVED
SMFMNAPS DS  XL4          OFFSET TO PRODUCT SECTION
SMFMNLPS DS  XL2          LENGTH OF PRODUCT SECTION
SMFMNPPS DS  XL2          NUMBER OF PRODUCT SECTIONS
SMFMNASS DS  XL4          OFFSET TO DATA SECTION
SMFMNASL DS  XL2          LENGTH OF DATA SECTION
SMFMNASN DS  XL2          NUMBER OF DATA SECTIONS
*
*          THIS CONCLUDES THE SMF HEADER
*
*
*          START OF THE SMF PRODUCT SECTION
*
SMFMNRVN DS  XL2          RECORD VERSION (CICS)
SMFMNPRN DS  CL8          PRODUCT NAME (GENERIC APPLID)
SMFMNSPN DS  CL8          PRODUCT NAME (SPECIFIC APPLID)
SMFMNMFL DS  XL2          RECORD MAINTENANCE INDICATOR
          DS  XL2          RESERVED
SMFMNCL  DS  XL2          CLASS OF DATA
*
*                          1 = DICTIONARY
*                          3 = PERFORMANCE
*                          4 = EXCEPTION
SMFMNDCA DS  XL4          OFFSET TO CICS FIELD CONNECTORS
SMFMNDCL DS  XL2          LENGTH OF EACH CICS FIELD CONNECTOR
SMFMNDCN DS  XL2          NUMBER OF CICS FIELD CONNECTORS
SMFMNDRA DS  XL4          OFFSET TO FIRST CICS DATA RECORD
SMFMNDRL DS  XL2          LENGTH OF EACH CICS DATA RECORD
SMFMNDRN DS  XL2          NUMBER OF CICS DATA RECORDS
*
          DS  XL20         RESERVED
SMFMNTAD DS  XL4          LOCAL TOD CLOCK ADJUSTMENT VALUE
SMFMNLSO DS  XL8          LEAP SECOND OFFSET TOD FORMAT
SMFMNDTO DS  XL8          LOCAL TIME/DATE OFFSET
          DS  XL2          RESERVED
SMFMNJBN DS  CL8          JOBNAME
SMFMNRSO DS  XL4          JOB DATE
SMFMSRST DS  XL4          JOB TIME
SMFMNUIF DS  CL8          USER IDENTIFICATION
SMFMNPDN DS  CL8          OPERATING SYSTEM PRODUCT LEVEL
*
*          THIS CONCLUDES THE SMF PRODUCT SECTION

```

Figure 52. Format of the SMF header and product section for monitoring records

Notes:

1. CICS sets only the subsystem-related bits of the operating system indicator flag byte in the SMF header (SMFMNFLG). SMF sets the remainder of the byte according to the operating system level and other factors. For an explanation of the setting of the other bits, refer to the *MVS/ESA System Management Facilities* manual.
2. For dictionary class monitoring records (described in “Dictionary data sections”), the fields SMFMNDRA, SMFMNDRL, and SMFMNDRN in the SMF product section have the following meaning:
 - SMFMNDRA** Offset to the first dictionary entry
 - SMFMNDRL** Length of a single dictionary entry
 - SMFMNDRN** Number of dictionary entries within the CICS data section.
3. The copy book DFHSMFDS is also provided and can be used to map the SMF header and the SMF product sections of all three subtypes of SMF 110 records written by CICS journaling, CICS monitoring, and CICS statistics.

CICS data section

The CICS data section can be made up of a dictionary data section, a performance data section, or an exception data section. You can identify which of these you are dealing with by looking at the value of field SMFMNCL in the SMF product section. Each of the data section types is described in the sections that follow.

Dictionary data sections

Dictionary data sections describe all the fields in the performance data records that are gathered during this CICS run. They describe **all** the system-provided data fields (whether you have excluded any or not), plus any user-provided data fields, which CICS takes at initialization time from the MCT entries you have coded. This means that the descriptions of the system-provided data fields never change, though the user data fields can be changed each time CICS is initialized. The contents of the dictionary data sections cannot be changed while CICS is running.

Dictionary data sections contain a variable number of 26-byte dictionary entries. Each dictionary entry provides the following information about a single performance record data field:

CMODNAME	The identifier of the group to which the field belongs
CMODTYPE	The field type
CMODIDNT	The field identifier
CMODLENG	The length of the field
CMODCONN	The connector value assigned to the field
CMODOFST	The offset of the field
CMODHEAD	The informal name of the field.

You can map the dictionary entries by generating a DSECT with the DFHMCTDR macro as shown in Figure 53 on page 550.

monitoring record formats

DFHMCTDR TYPE=(PREFIX,CMO)

CMO is the default label prefix. The DSECT is as follows:

```

CMODNAME DS    CL8    + 0    NAME OF OWNER (entry name)
CMODTYPE DS    C      + 8    OBJECT TYPE
*
*                'S' = STOPWATCH (CLOCK)
*                'A' = ACCUMULATOR (COUNT)
*                'C' = BYTE-STRING FIELD
*                'T' = TIMESTAMP (STCK FORMAT)
*                'P' = PACKED-DECIMAL FIELD
CMODIDNT DS    CL3    +9     ID WITHIN TYPE
*                CLOCK-, COUNT-, OR FIELD-NO.
CMODLENG DS    H      +12   LENGTH OF OBJECT
CMODCONN DS    XL2    +14   ASSIGNED CONNECTOR
CMODOFST DS    XL2    +16   ASSIGNED OFFSET
CMODHEAD DS    CL8    +18   INFORMAL NAME
CMODNEXT EQU   *
  
```

Figure 53. CICS monitoring dictionary entry DSECT

Whenever the monitoring of performance class data is switched on, whether at CICS initialization or while CICS is running, a dictionary data section is written. So, if the monitoring of performance class data is switched on and off three times during a single CICS run, there are three separate, but identical, dictionary data sections for that run. The dictionary data section is passed to SMF, together with any performance data sections, when the first buffer of performance data sections for a performance class data monitoring session is output to SMF. Any offline utility should use the most recent dictionary record encountered when processing CICS monitoring records.

The format of dictionary data sections is shown in Figure 54. A list of the default CICS dictionary entries is given in Figure 55 on page 551.

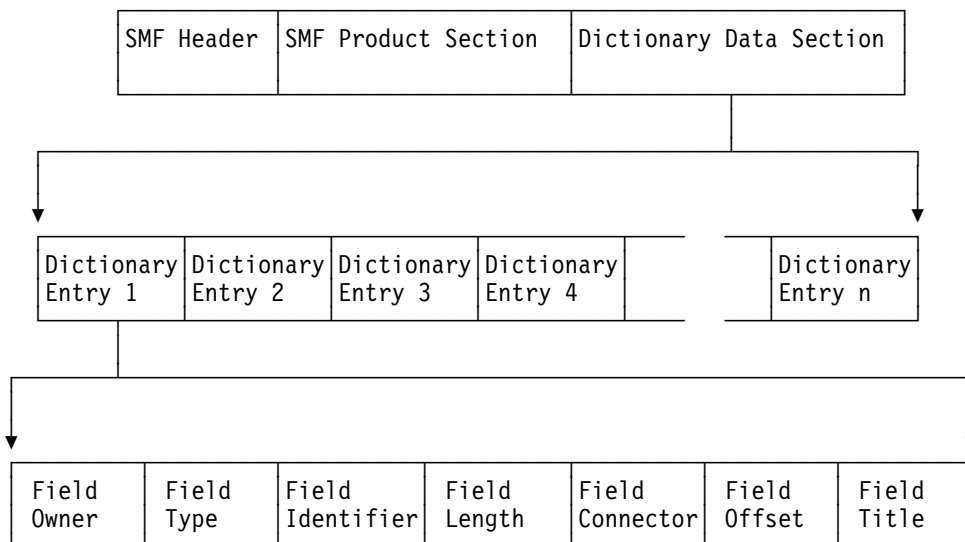


Figure 54. Format of the CICS monitoring dictionary data section

FIELD OWNER	FIELD TYPE	FIELD IDENTIFIER	FIELD LENGTH	FIELD CONNECTOR	FIELD OFFSET	FIELD TITLE
DFHTASK	C	001	4	X'0001'	X'0000'	TRAN
DFHTERM	C	002	4	X'0002'	X'0004'	TERM
DFHCICS	C	089	8	X'0003'	X'0008'	USERID
DFHTASK	C	004	4	X'0004'	X'0010'	TTYTYPE
DFHCICS	T	005	8	X'0005'	X'0014'	START
DFHCICS	T	006	8	X'0006'	X'001C'	STOP
DFHTASK	P	031	4	X'0007'	X'0024'	TRANNUM
DFHTASK	A	109	4	X'0008'	X'0028'	TRANPRI
DFHTASK	C	166	8	X'0009'	X'002C'	TCLSNAME
DFHTERM	C	111	8	X'000A'	X'0034'	LUNAME
DFHPROG	C	071	8	X'000B'	X'003C'	PGMNAME
DFHTASK	C	097	20	X'000C'	X'0044'	NETNAME
DFHTASK	C	098	8	X'000D'	X'0058'	UOWID
DFHCICS	C	130	4	X'000E'	X'0060'	RSYSID
DFHTASK	C	064	4	X'000F'	X'0064'	TASKFLAG
DFHPROG	C	113	4	X'0010'	X'0068'	ABCODE0
DFHPROG	C	114	4	X'0011'	X'006C'	ABCODEC
DFHCICS	C	112	4	X'0012'	X'0070'	RTYPE
DFHTERM	A	034	4	X'0013'	X'0074'	TCMSGIN1
DFHTERM	A	083	4	X'0014'	X'0078'	TCCHRIN1
DFHTERM	A	035	4	X'0015'	X'007C'	TCMSGOU1
DFHTERM	A	084	4	X'0016'	X'0080'	TCCHROU1
DFHTERM	A	067	4	X'0017'	X'0084'	TCMSGIN2
DFHTERM	A	085	4	X'0018'	X'0088'	TCCHRIN2
DFHTERM	A	068	4	X'0019'	X'008C'	TCMSGOU2
DFHTERM	A	086	4	X'001A'	X'0090'	TCCHROU2
DFHTERM	A	135	4	X'001B'	X'0094'	TCM62IN2
DFHTERM	A	137	4	X'001C'	X'0098'	TCC62IN2
DFHTERM	A	136	4	X'001D'	X'009C'	TCM62OU2
DFHTERM	A	138	4	X'001E'	X'00A0'	TCC62OU2
DFHTERM	A	069	4	X'001F'	X'00A4'	TCALLOCT
DFHSTOR	A	054	4	X'0020'	X'00A8'	SCUGETCT
DFHSTOR	A	105	4	X'0021'	X'00AC'	SCUGETCT
DFHSTOR	A	117	4	X'0022'	X'00B0'	SCCGETCT
DFHSTOR	A	120	4	X'0023'	X'00B4'	SCCGETCT
DFHSTOR	A	033	4	X'0024'	X'00B8'	SCUSRHWM
DFHSTOR	A	106	4	X'0025'	X'00BC'	SCUSRHWM
DFHSTOR	A	116	4	X'0026'	X'00C0'	SC24CHWM
DFHSTOR	A	119	4	X'0027'	X'00C4'	SC31CHWM
DFHSTOR	A	095	8	X'0028'	X'00C8'	SCUSRSTG
DFHSTOR	A	107	8	X'0029'	X'00D0'	SCUSRSTG
DFHSTOR	A	118	8	X'002A'	X'00D8'	SC24COCC
DFHSTOR	A	121	8	X'002B'	X'00E0'	SC31COCC
DFHSTOR	A	087	4	X'002C'	X'00E8'	PCSTGHWM
DFHSTOR	A	139	4	X'002D'	X'00EC'	PC31AHWM
DFHSTOR	A	108	4	X'002E'	X'00F0'	PC24BHWM
DFHSTOR	A	142	4	X'002F'	X'00F4'	PC31CHWM
DFHSTOR	A	143	4	X'0030'	X'00F8'	PC24CHWM
DFHSTOR	A	122	4	X'0031'	X'00FC'	PC31RHWM

Figure 55 (Part 1 of 3). Default CICS dictionary entries

monitoring record formats

DFHSTOR	A	162	4	X'0032'	X'0100'	PC24RHWM
DFHSTOR	A	161	4	X'0033'	X'0104'	PC31SHWM
DFHSTOR	A	160	4	X'0034'	X'0108'	PC24SHWM
DFHFILE	A	036	4	X'0035'	X'010C'	FCGETCT
DFHFILE	A	037	4	X'0036'	X'0110'	FCPUTCT
DFHFILE	A	038	4	X'0037'	X'0114'	FCBRWCT
DFHFILE	A	039	4	X'0038'	X'0118'	FCADDCT
DFHFILE	A	040	4	X'0039'	X'011C'	FCDELCT
DFHFILE	A	093	4	X'003A'	X'0120'	FCTOTCT
DFHFILE	A	070	4	X'003B'	X'0124'	FCAMCT
DFHDEST	A	041	4	X'003C'	X'0128'	TDGETCT
DFHDEST	A	042	4	X'003D'	X'012C'	TDPUTCT
DFHDEST	A	043	4	X'003E'	X'0130'	TDPURCT
DFHDEST	A	091	4	X'003F'	X'0134'	TDTOTCT
DFHTEMP	A	044	4	X'0040'	X'0138'	TSGETCT
DFHTEMP	A	046	4	X'0041'	X'013C'	TSPUTACT
DFHTEMP	A	047	4	X'0042'	X'0140'	TSPUTMCT
DFHTEMP	A	092	4	X'0043'	X'0144'	TSTOTCT
DFHMAPP	A	050	4	X'0044'	X'0148'	BMSMAPCT
DFHMAPP	A	051	4	X'0045'	X'014C'	BMSINCT
DFHMAPP	A	052	4	X'0046'	X'0150'	BMSOUTCT
DFHMAPP	A	090	4	X'0047'	X'0154'	BMSTOTCT
DFHPROG	A	055	4	X'0048'	X'0158'	PCLINKCT
DFHPROG	A	056	4	X'0049'	X'015C'	PCXCTLCT
DFHPROG	A	057	4	X'004A'	X'0160'	PCLOADCT
DFHJOUR	A	058	4	X'004B'	X'0164'	JCPUWRCT
DFHTASK	A	059	4	X'004C'	X'0168'	ICPUINCT
DFHSYNC	A	060	4	X'004D'	X'016C'	SPSYNCT
DFHFPEI	A	150	4	X'004E'	X'0170'	SZALLOCT
DFHFPEI	A	151	4	X'004F'	X'0174'	SZRCVCT
DFHFPEI	A	152	4	X'0050'	X'0178'	SZSENDCT
DFHFPEI	A	153	4	X'0051'	X'017C'	SZSTRCT
DFHFPEI	A	154	4	X'0052'	X'0180'	SZCHROUT
DFHFPEI	A	155	4	X'0053'	X'0184'	SZCHRIN
DFHFPEI	A	157	4	X'0054'	X'0188'	SZALLCTO
DFHFPEI	A	158	4	X'0055'	X'018C'	SZRCVTO
DFHFPEI	A	159	4	X'0056'	X'0190'	SZTOTCT
DFHTASK	S	007	8	X'0057'	X'0194'	USRDISPT
DFHTASK	S	008	8	X'0058'	X'019C'	USRCPUT
DFHTASK	S	014	8	X'0059'	X'01A4'	SUSPTIME
DFHTASK	S	102	8	X'005A'	X'01AC'	DISPWT
DFHCICS	S	103	8	X'005B'	X'01B4'	EXWTTIME
DFHTERM	S	009	8	X'005C'	X'01BC'	TCIOWTT
DFHFILE	S	063	8	X'005D'	X'01C4'	FCIOWTT
DFHJOUR	S	010	8	X'005E'	X'01CC'	JCIOWTT
DFHTEMP	S	011	8	X'005F'	X'01D4'	TSIOWTT
DFHTERM	S	100	8	X'0060'	X'01DC'	IRIOWTT
DFHDEST	S	101	8	X'0061'	X'01E4'	TDIOWTT

Figure 55 (Part 2 of 3). Default CICS dictionary entries

DFHPROG	S	115	8	X'0062'	X'01EC'	PCLOADTM
DFHTASK	S	125	8	X'0063'	X'01F4'	DSPDELAY
DFHTASK	S	126	8	X'0064'	X'01FC'	TCLDELAY
DFHTASK	S	127	8	X'0065'	X'0204'	MXTDELAY
DFHTASK	S	129	8	X'0066'	X'020C'	ENQDELAY
DFHTERM	S	133	8	X'0067'	X'0214'	LU61WTT
DFHTERM	S	134	8	X'0068'	X'021C'	LU62WTT
DFHFPEI	S	156	8	X'0069'	X'0224'	SZWAIT
DFHTASK	S	170	8	X'006A'	X'022C'	RMITIME
DFHTASK	S	171	8	X'006B'	X'0234'	RMISUSP

Figure 55 (Part 3 of 3). Default CICS dictionary entries

Note: The “field types” in Figure 55 are:

- A** Count
- C** Byte-string
- P** Packed decimal number
- S** Clock
- T** Time stamp.

Performance data sections

Each performance data section is made up of a string of field connectors, followed by one or more performance data records. All of the performance records produced by a single CICS run have the same format, and each record is, by default, 380 bytes long. However, the length of the performance records changes if you add user data at user EMPs, or if you exclude any system-defined data from the monitoring process. All of the system-defined data fields in the performance records are described in “Performance class data” on page 558. The format of the performance data section is shown in Figure 56.

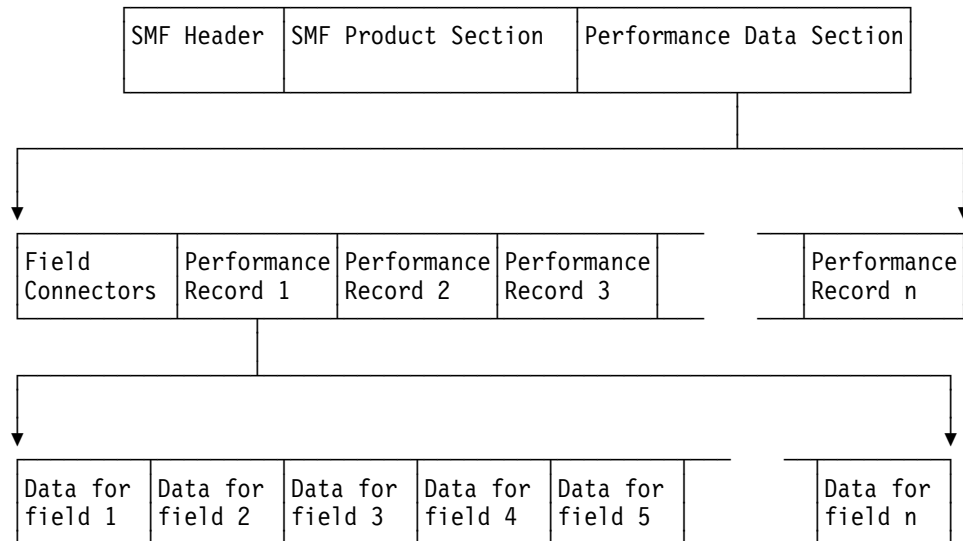


Figure 56. Format of the performance data section

Relationship of the dictionary record to the performance records

Following the performance records' SMF product section, and before the performance records themselves, is a string of **field connectors**. The purpose of the field connectors is to tell you which fields are going to occur in the performance records produced by this CICS run. Each field connector corresponds to one field in each of the succeeding performance records. The first field connector corresponds to the first field, the second to the second field, and so on. Each field connector also corresponds to a single dictionary entry in the associated dictionary record: the connector value is equal to the value of CMODCONN in the corresponding dictionary entry. In this way, each performance record field is connected to the dictionary entry that describes it. A useful technique for calculating the offset of a particular dictionary entry is to take the connector, subtract one, and multiply the result by the length of a single dictionary entry.

Thus, the string of field connectors is the key to the dictionary. And without the dictionary, reporting and analysis programs cannot interpret the performance data.

The successive performance records can be regarded as rows in a table, with each column corresponding to one type of field within the records. Each field connector then describes the contents of one column. This view of the data is helpful when designing tabular reports, which are often arranged in this way.

Figure 57 on page 555 illustrates the relationship between the dictionary record, the field connectors, and the performance records.

Dictionary Record

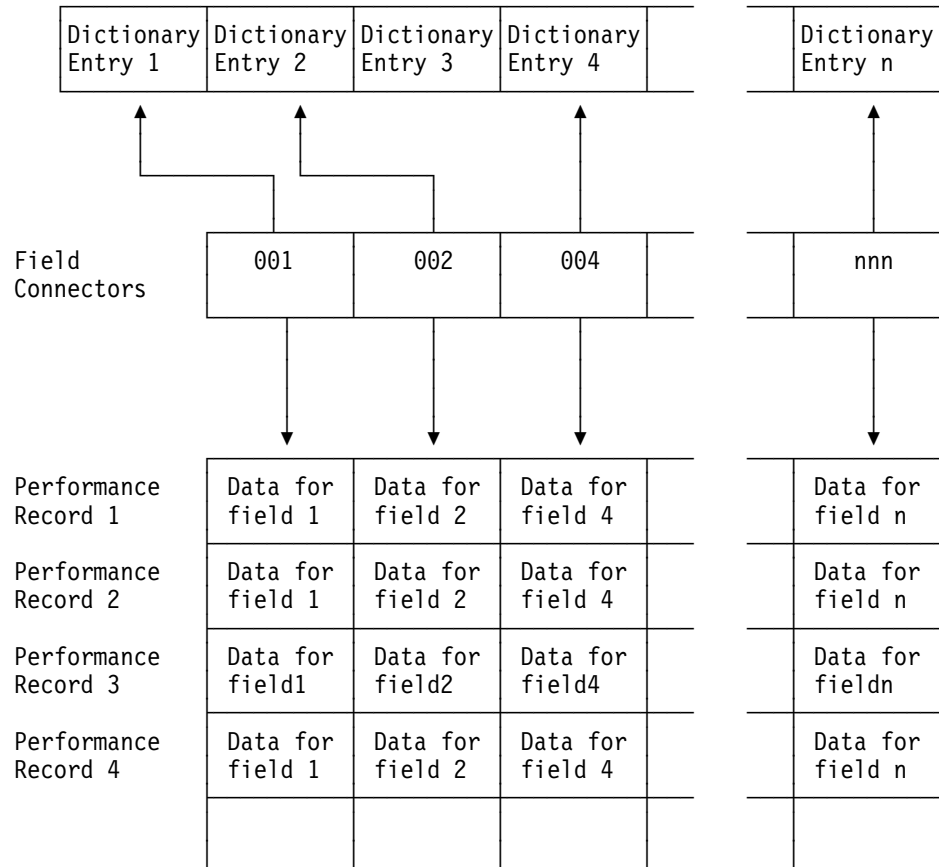


Figure 57. Relationship between the dictionary record and the performance records. In this example, the data that is defined by Dictionary Entry 3 has been excluded, so there is no field connector value for it and it does not appear in the performance records.

How the string of field connectors is constructed: When CICS is initialized, a unique connector value is assigned to every dictionary entry. CICS then examines the MCT entries for this run to see if you have excluded any system-defined performance data. If you have, the offset values for their corresponding dictionary entries are set to X'FFFF'. CICS then constructs a sequence of field connectors that excludes those with offsets of X'FFFF'. In this way, the connectors tell you which system- and user-data fields are going to occur in your performance records for this run. If you have not excluded any system-defined performance data, there is one field connector for every dictionary entry.

Please note the following:

Field connectors link the fields in a performance record with their dictionary entries. They are unique values that are assigned at initialization time. They may, therefore, change from one run of CICS to the next.

Field identifiers allow you to exclude specific system-defined performance data from being collected during a CICS run. They are unique within a group name and record type, and they do **not** change between CICS runs. There is more information about field identifiers in the *CICS/ESA Resource Definition Guide*.

Field offsets in the performance record allow you to build a table for fast selection of required fields in your monitoring data processing programs.

Exception data sections

The format of an exception data record (including the SMF header and SMF product section) is shown in Figure 58. The exception data section contains a single exception record representing one exception condition.

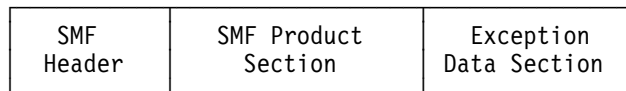


Figure 58. Format of an SMF exception data record

The format of the exception data section can be mapped by the DSECT MNEXCDS, which you can generate using the DFHMNEXC macro as follows:

```
MNEXCDS DFHMNEXC PREFIX=EXC
```

The label 'MNEXCDS' is the default DSECT name, and EXC is the default PREFIX value, so you could also generate the DSECT simply by coding

```
DFHMNEXC
```

The MNEXCDS DSECT has the format shown in Figure 59.

MNEXCDS	DSECT		
EXCMNTRN	DS	CL4	TRANSACTION IDENTIFICATION
EXCMNTER	DS	XL4	TERMINAL IDENTIFICATION
EXCMNUSR	DS	CL8	USER IDENTIFICATION
EXCMNTST	DS	CL4	TRANSACTION START TYPE
EXCMNSTA	DS	XL8	EXCEPTION START TIME
EXCMNSTO	DS	XL8	EXCEPTION STOP TIME
EXCMNTNO	DS	CL4	TRANSACTION NUMBER
EXCMNTPR	DS	CL4	TRANSACTION PRIORITY
	DS	CL4	RESERVED
EXCMNLUN	DS	CL8	LUNAME
	DS	CL4	RESERVED
EXCMNEXN	DS	XL4	EXCEPTION NUMBER
EXCMNRTY	DS	CL8	EXCEPTION RESOURCE TYPE
EXCMNRID	DS	CL8	EXCEPTION RESOURCE ID
EXCMNTYP	DS	XL2	EXCEPTION TYPE
EXCMNWT	EQU	X'0001'	WAIT
EXCMNBWT	EQU	X'0002'	BUFFER WAIT
EXCMNSWT	EQU	X'0003'	STRING WAIT
	DS	CL2	RESERVED
EXCMNTCN	DS	CL8	TRANSACTION CLASS NAME
*			END OF EXCEPTION RECORD

Figure 59. CICS monitoring exception record DSECT

For further information about exception class data, refer to page 575.

monitoring data

The 8 reserved bits have the following significance:

Bits 0, 1, 2 and 3	Used for online control of the clock when it is running, and should always be zeros on output.
Bits 4 and 7	Not used.
Bits 5 and 6	Used to indicate, when set to 1, that the clock has suffered at least one out-of-phase start (bit 5) or stop (bit 6).

A **time stamp** is an 8-byte copy of the output of an STCK instruction.

Note: All times produced in the offline reports are in GMT (Greenwich Mean Time) not local time. Times produced by online reporting can be expressed in either GMT or local time.

Performance class data

The performance class data is described below in order of group name. The group name is always in field CMODNAME of the dictionary entry.

A user task can be represented by one or more performance class monitoring records, depending on whether the MCT event monitoring option DELIVER or the system initialization parameters MNCONV=YES or MNSYNC=YES have been selected. In the descriptions that follow, the term “user task” means “that part or whole of a transaction that is represented by a performance class record”, unless the description states otherwise.

A note about response time

You can calculate the internal CICS response time by subtracting performance data field 005 (start time) from performance data field 006 (finish time).

Figure 61 shows the relationship of dispatch time, suspend time, and CPU time with the response time.

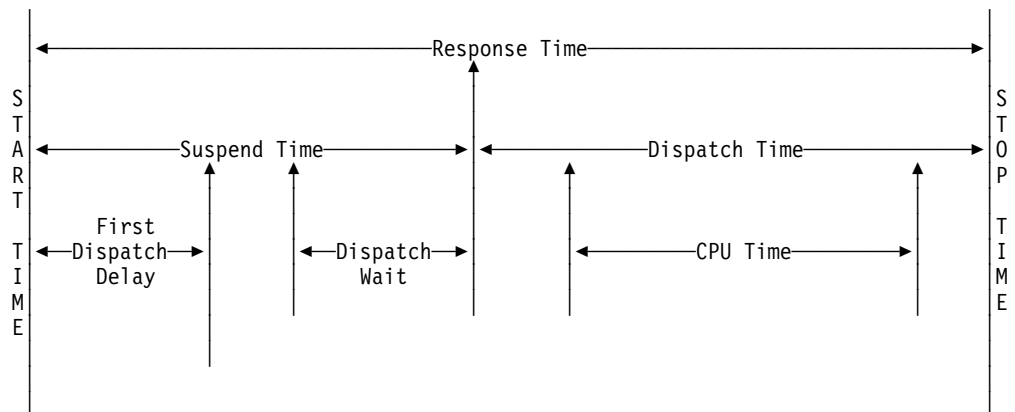


Figure 61. Response time relationships

A note about wait times

The performance data fields 009, 010, 011, 063, 100, 101, 129, 133, 134, 156, and 171. all record the elapsed time spent waiting for a particular type of I/O operation. For example, field 009 records the elapsed time waiting for terminal I/O. The elapsed time includes not only that time during which the I/O operation is actually taking place, but also the time during which the access method is completing the outstanding event control block, and the time subsequent to that until the waiting CICS transaction is redispached.

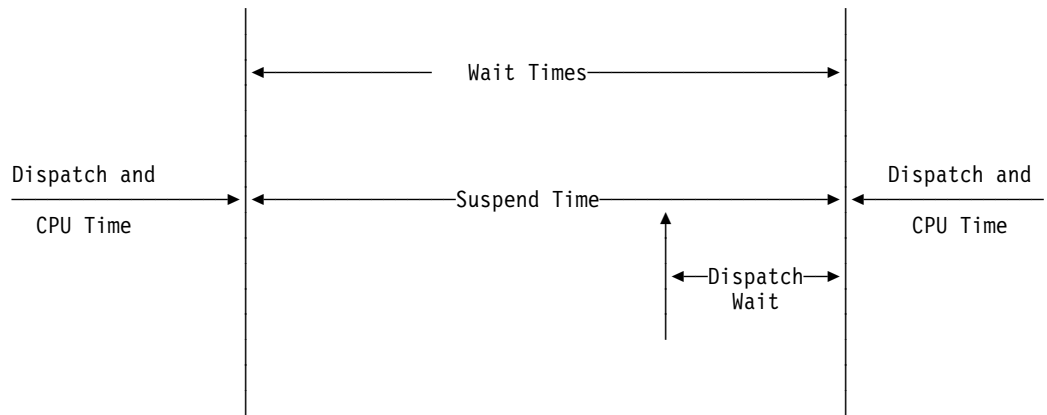


Figure 62. Wait time relationships

A note about program load time

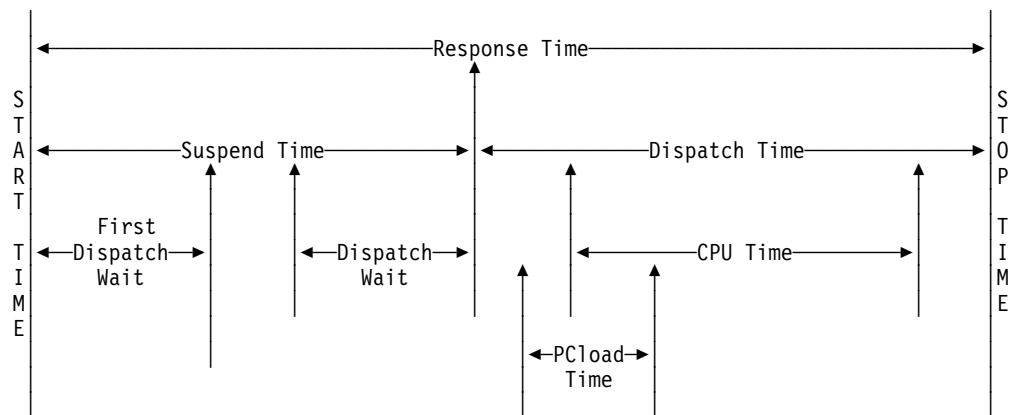


Figure 63. Program load time

Figure 63 shows the relationship between the program load time (field 115) and the dispatch time and the suspend time (fields 7 and 14).

A note about exception wait time

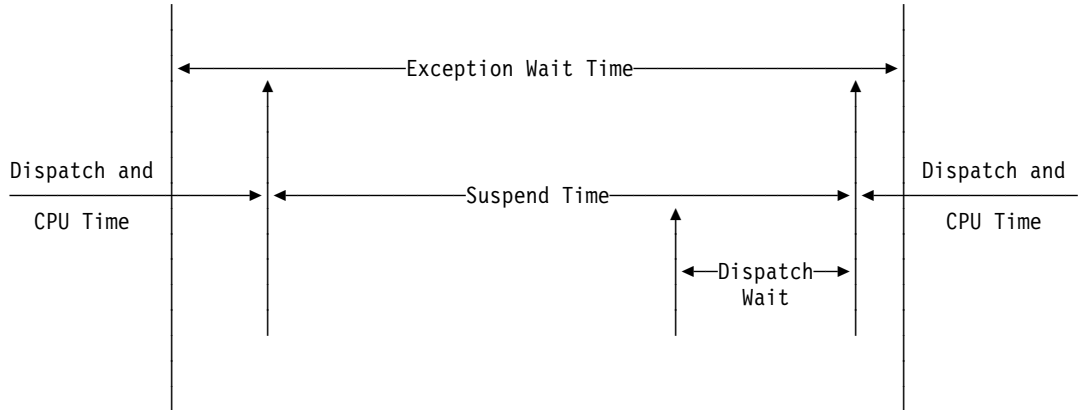


Figure 64. Exception wait time

Figure 64 shows the relationship between the exception wait time (field 103), and the dispatch time and suspend time (fields 7 and 14).

A note about RMI elapsed and suspend time

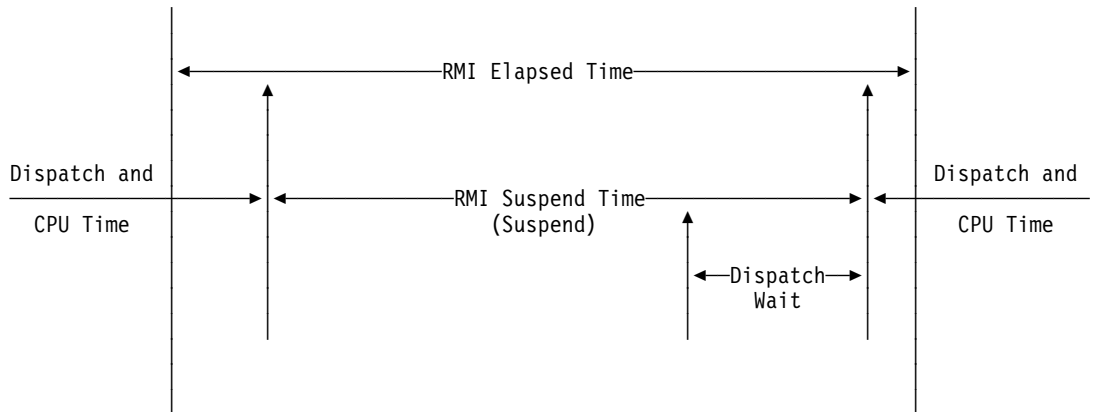


Figure 65. RMI elapsed and suspend time

Figure 65 shows the relationship between the RMI elapsed time and the suspend time (fields 170 and 171).

A note about storage occupancy counts

An occupancy count measures the area under the curve of user-task storage in use against elapsed time. The unit of measure is the “byte-unit”, where the “unit” is equal to 1024 microseconds, or 1.024 milliseconds. A user task occupying, for example, 256 bytes for 125 milliseconds, is measured as follows:

$$125 / 1.024 \text{ ms} = 122 \text{ units} * 256 = 31\,232 \text{ byte-units.}$$

where *ms* is milliseconds.

Note: All references to start time and stop time in the calculations below refer to the middle 4 bytes of each 8-byte start/stop time field. Bit 51 of Start time or Stop time represents a unit of 16 microseconds.

To calculate the response time in microsecond units:

$$\text{Response} = ((\text{Stop time} - \text{Start time}) * 16)$$

To calculate the number of 1024-microsecond units:

$$\text{Units} = (\text{Response} / 1024)$$

or

$$\text{Units} = ((\text{Stop time} - \text{Start time}) / 64)$$

To calculate the average user-task storage used from the storage occupancy count:

$$\text{Average user-task} = \text{storage used} (\text{Storage Occupancy} / \text{Units})$$

To calculate units per second:

$$\text{Units Per Second} = (1\,000\,000 / 1024) = 976.5625$$

To calculate the response time in seconds:

$$\text{Response time} = (((\text{Stop time} - \text{Start time}) * 16) / 1\,000\,000)$$

During the life of a user task CICS measures, calculates, and accumulates the storage occupancy at the following points:

- Before GETMAIN increases current user-storage values
- Before FREEMAIN reduces current user-storage values
- Just before the performance record is moved to the buffer.

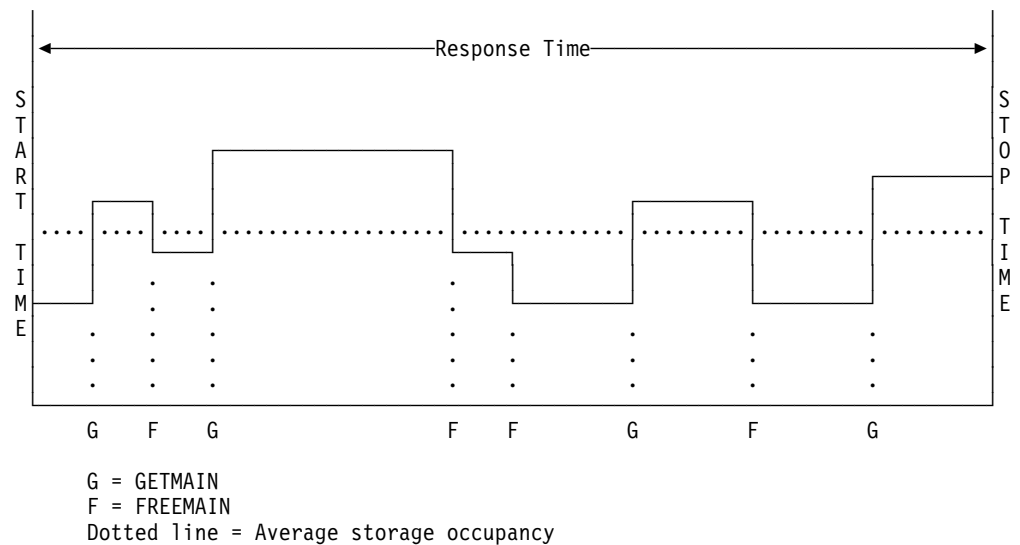


Figure 66. Storage occupancy

A note about program storage

The level of program storage currently in use is incremented at LOAD, LINK, and XCTL events by the size (in bytes) of the referenced program, and is decremented at RELEASE or RETURN events.

Notes:

1. On an XCTL event, the program storage currently in use is also decremented by the size of the program issuing the XCTL, because the program is no longer required.

- 2.

APAR PN63021

Documentation for PN63021 added on 17 October 1994.

On a RELEASE event, if the program being released was loaded via a LOAD HOLD command, program storage currently in use is not decremented. This is because it cannot be determined whether the program being released by the current task was also loaded by the current task.

Figure 67 on page 563 shows the relationships between the “high-water mark” data fields that contain the maximum amounts of program storage in use by the user task. Field PCSTGHWM (field ID 087) contains the maximum amount of program storage in use by the task both above **and** below the 16MB line. Fields PC31AHWM (139) and PC24BHWM (108) are subsets of PCSTGHWM, containing the maximum amounts in use above and below the 16MB line, respectively. Further subset-fields contain the maximum amounts of storage in use by the task in each of the CICS dynamic storage areas (DSAs).

Note: The totaled values of all the subsets in a superset may not necessarily equate to the value of the superset; for example, the value of PC31AHWM plus the value of PC24BHWM may not equal the value of PCSTGHWM. This is because the peaks in the different types of program storage acquired by the user task do not necessarily occur simultaneously.

The “high-water mark” fields are described in detail in “Performance data in group DFHSTOR” on page 568.

PCSTGHWM - high-water mark of program storage in all CICS DSAs

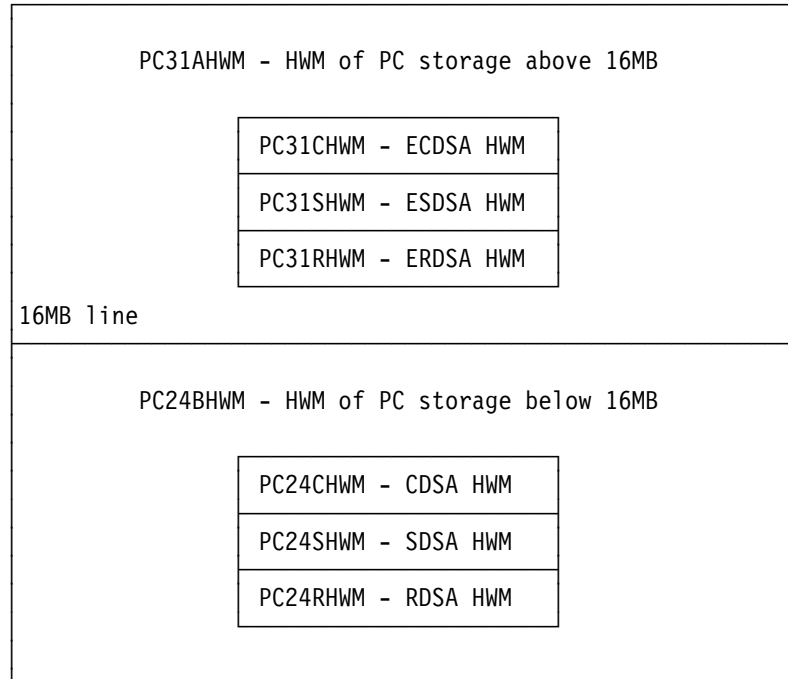


Figure 67. Relationships between the “high-water mark” program storage data fields

Performance data in group DFHCICS

005 (TYPE-T, ‘START’, 8 BYTES)

Start time of measurement interval. This is one of the following:

- The time at which the user task was attached
- The time at which data recording was most recently reset in support of the MCT user event monitoring point DELIVER option or the monitoring options MNSYNC=YES or FREQUENCY.

For more information see “Clocks and time stamps” on page 557.

006 (TYPE-T, ‘STOP’, 8 BYTES)

Finish time of measurement interval. This is either the time at which the user task was detached, or the time at which data recording was completed in support of the MCT user event monitoring point DELIVER option or the monitoring options MNSYNC=YES or FREQUENCY.

Notes:

1. The measurement interval includes transaction dispatch time and suspend time values, if appropriate, and any first dispatch delay due to the CMXT (if this is defined in a TCLASS) or MXT conditions.
2. Response Time = STOP – START. For more information see “A note about response time” on page 558.

089 (TYPE-C, ‘USERID’, 8 BYTES)

User identification at task creation. This can also be the remote user identifier for a task created as the result of receiving an ATTACH request across an MRO or APPC link with attach-time security enabled.

103 (TYPE-S, 'EXWTTIME', 8 BYTES)

Accumulated data for exception conditions. The 32-bit clock contains the total elapsed time for which the user waited on exception conditions. The 24-bit period count equals the number of exception conditions that have occurred for this task. For more information see "Clocks and time stamps" on page 557.

Note: The performance class data field 'exception wait time' will be updated when exception conditions are encountered even when the exception class is inactive. For more information see Figure 64 on page 560.

112 (TYPE-C, 'RTYPE', 4 BYTES)

Performance record type (low-order byte-3):

- C** Record output for a terminal converse
- D** Record output for a user EMP DELIVER request
- F** Record output for a long-running transaction
- S** Record output for a syncpoint
- T** Record output for a task termination.

130 (TYPE-C, 'RSYSID', 4 bytes)

is the name (sysid) of the remote system to which this transaction was routed either statically or dynamically.

Note: If the transaction was not routed or was routed locally, this field is set to null. Also see the program name (field 71).

Performance data in group DFHDEST

041 (TYPE-A, 'TDGETCT', 4 BYTES)

Number of transient data GET requests issued by the user task.

042 (TYPE-A, 'TDPUTCT', 4 BYTES)

Number of transient data PUT requests issued by the user task.

043 (TYPE-A, 'TDPURCT', 4 BYTES)

Number of transient data PURGE requests issued by the user task.

091 (TYPE-A, 'TDTOTCT', 4 BYTES)

Total number of transient data requests issued by the user task. This field is the sum of TDGETCT, TDPUTCT, and TDPURCT.

101 (TYPE-S, 'TDIOWTT', 8 BYTES)

Elapsed time in which the user waited for VSAM transient data I/O. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

Performance data in group DFHFPEI

150 (TYPE-A, 'SZALLOCT', 4 bytes)

Number of conversations allocated by the user task. This number is incremented for each FEPI ALLOCATE POOL or FEPI CONVERSE POOL.

151 (TYPE-A, 'SZRCVCT', 4 bytes)

Number of FEPI RECEIVE requests made by the user task. This number is also incremented for each FEPI CONVERSE request.

152 (TYPE-A, 'SZSENDCT', 4 bytes)

Number of FEPI SEND requests made by the user task. This number is also incremented for each FEPI CONVERSE request.

153 (TYPE-A,'SZSTRCT',4 bytes)

Number of FEPI START requests made by the user task.

154 (TYPE-A,'SZCHROUT',4 bytes)

Number of characters sent through FEPI by the user task.

155 (TYPE-A,'SZCHRIN',4 bytes)

Number of characters received through FEPI by the user task.

156 (TYPE-S,'SZWAIT',8 bytes)

Elapsed time in which the user task waited for all FEPI services. For more information see “Clocks and time stamps” on page 557, and “A note about wait times” on page 559.

157 (TYPE-A,'SZALLCTO',4 bytes)

Number of times the user task timed out while waiting to allocate a conversation.

158 (TYPE-A,'SZRCVTO',4 bytes)

Number of times the user task timed out while waiting to receive data.

159 (TYPE-A,'SZTOTCT',4 bytes)

Total number of all FEPI API and SPI requests made by the user task.

Performance data in group DFHFILE

036 (TYPE-A, 'FCGETCT', 4 BYTES)

Number of file GET requests issued by the user task.

037 (TYPE-A, 'FCPUTCT', 4 BYTES)

Number of file PUT requests issued by the user task.

038 (TYPE-A, 'FCBRWCT', 4 BYTES)

Number of file browse requests issued by the user task. This number excludes the START and END browse requests.

039 (TYPE-A, 'FCADDCT', 4 BYTES)

Number of file ADD requests issued by the user task.

040 (TYPE-A, 'FCDELCT', 4 BYTES)

Number of file DELETE requests issued by the user task.

063 (TYPE-S, 'FCIOWTT', 8 BYTES)

Elapsed time in which the user task waited for file I/O. For more information see “Clocks and time stamps” on page 557, and “A note about wait times” on page 559.

070 (TYPE-A, 'FCAMCT', 4 BYTES)

Number of times the user task invoked file access-method interfaces. This number excludes requests for OPEN and CLOSE.

093 (TYPE-A, 'FCTOTCT', 4 BYTES)

Total number of file control requests issued by the user task. This number excludes any request for OPEN, CLOSE, ENABLE, or DISABLE of a file.

How EXEC CICS file commands correspond to file control monitoring fields:

<i>Table 121. EXEC CICS file commands related to file control monitoring fields</i>	
EXEC CICS command	Monitoring fields
READ	FCGETCT and FCTOTCT
READ UPDATE	FCGETCT and FCTOTCT
DELETE (after READ UPDATE)	FCDELCT and FCTOTCT
DELETE (with RIDFLD)	FCDELCT and FCTOTCT
REWRITE	FCPUTCT and FCTOTCT
WRITE	FCADDCT and FCTOTCT
STARTBR	FCTOTCT
READNEXT	FCBRWCT and FCTOTCT
READPREV	FCBRWCT and FCTOTCT
ENDBR	FCTOTCT
RESETBR	FCTOTCT
UNLOCK	FCTOTCT
Note: The number of STARTBR, ENDBR, RESETBR and UNLOCK file control requests can be calculated by subtracting the file request counts, FCGETCT, FCPUTCT, FCBRWCT, FCADDCT, and FCDELCT from the total file request count, FCTOTCT.	

Performance data in group DFHJOUR

010 (TYPE-S, 'JCIOWTT', 8 BYTES)

Elapsed time for which the user task waited for journal I/O. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

058 (TYPE-A, 'JCPUWRCT', 4 BYTES)

Number of journal output requests during the user task.

Performance data in group DFHMAPP

050 (TYPE-A, 'BMSMAPCT', 4 BYTES)

Number of BMS MAP requests issued by the user task. This field corresponds to the number of RECEIVE MAP requests that did not incur a terminal I/O, and the number of RECEIVE MAP FROM requests.

051 (TYPE-A, 'BMSINCT', 4 BYTES)

Number of BMS IN requests issued by the user task. This field corresponds to the number of RECEIVE MAP requests that incurred a terminal I/O.

052 (TYPE-A, 'BMSOUTCT', 4 BYTES)

Number of BMS OUT requests issued by the user task. This field corresponds to the number of SEND MAP requests.

090 (TYPE-A, 'BMSTOTCT', 4 BYTES)

Total number of BMS requests issued by the user task. This field is the sum of BMSMAPCT, BMSINCT, and BMSOUTCT counts.

Performance data in group DFHPROG

	055 (TYPE-A, 'PCLINKCT', 4 BYTES)
	Number of program LINK requests issued by the user task, including the link to
	the first program of the user task.
	056 (TYPE-A, 'PCXCTLCT', 4 BYTES)
	Number of program XCTL requests issued by the user task.
	057 (TYPE-A, 'PCLOADCT', 4 BYTES)
	Number of program LOAD requests issued by the user task.
	071 (TYPE-C, 'PGMNAME', 8 BYTES)
	The name of the first program invoked at attach-time.
#	For a remote transaction:
#	• If this CICS' definition of the remote transaction does not specify a program
#	name, this field contains blanks.
#	• If this CICS' definition of the remote transaction specifies a program name,
#	this field contains the name of the specified program. (Note that this is not
#	necessarily the program that is run on the remote system.)
#	For a dynamically-routed transaction, if the dynamic transaction routing program
#	routes the transaction locally and specifies an alternate program name, this
#	field contains the name of the alternate program.
	113 (TYPE-C, 'ABCODEO', 4 BYTES)
	Original abend code.
	114 (TYPE-C, 'ABCODEC', 4 BYTES)
	Current abend code.
	115 (TYPE-S, 'PCLOADTM', 8 BYTES)
	Elapsed time in which the user task waited for program library (DFHRPL)
	fetches. Only fetches for programs with installed program definitions or
	autoinstalled as a result of application requests are included in this figure.
	However, installed programs residing in the LPA are not included (because
	they do not incur a physical fetch from a library). For more information about
	program load time see "Clocks and time stamps" on page 557, and "A note
	about wait times" and Figure 63 on page 559.

Performance data in group DFHSTOR

User storage fields in group DFHSTOR:

033 (TYPE-A, 'SCUSRHWM', 4 BYTES)

Maximum amount (high-water mark) of user storage allocated to the user task below the 16MB line, in the user dynamic storage area (UDSA).

054 (TYPE-A, 'SCUGETCT', 4 BYTES)

Number of user-storage GETMAIN requests issued by the user task below the 16MB line, in the UDSA.

095 (TYPE-A, 'SCUSRSTG', 8 BYTES)

Storage occupancy of the user task below the 16MB line, in the UDSA. This measures the area under the curve of storage in use against elapsed time. For more information about storage occupancy, see "A note about storage occupancy counts" on page 560.

105 (TYPE-A, 'SCUGETCT', 4 BYTES)

Number of user-storage GETMAIN requests issued by the user task for storage above the 16MB line, in the extended user dynamic storage area (EUDSA).

106 (TYPE-A, 'SCUSRHWM', 4 BYTES)

Maximum amount (high-water mark) of user-storage allocated to the user task above the 16MB line, in the EUDSA.

107 (TYPE-A, 'SCUCRSTG', 8 BYTES)

Storage occupancy of the user task above the 16MB line, in the EUDSA. This measures the area under the curve of storage in use against elapsed time. For more information see "A note about storage occupancy counts" on page 560.

116 (TYPE-A, 'SC24CHWM', 4 BYTES)

Maximum amount (high-water mark) of user-storage allocated to the user task below the 16MB line, in the CICS dynamic storage area (CDSA).

117 (TYPE-A, 'SCCGETCT', 4 BYTES)

Number of user-storage GETMAIN requests issued by the user task for storage below the 16MB line, in the CDSA.

118 (TYPE-A, 'SC24COCC', 8 BYTES)

Storage occupancy of the user task below the 16MB line, in the CDSA. This measures the area under the curve of storage in use against elapsed time. For more information see "A note about storage occupancy counts" on page 560.

119 (TYPE-A, 'SC31CHWM', 4 BYTES)

Maximum amount (high-water mark) of user-storage allocated to the user task above the 16MB line, in the extended CICS dynamic storage area (ECDSA).

120 (TYPE-A, 'SCCGETCT', 4 BYTES)

Number of user-storage GETMAIN requests issued by the user task for storage above the 16MB line, in the ECDSA.

121 (TYPE-A, 'SC31COCC', 8 BYTES)

Storage occupancy of the user task above the 16MB line, in the ECDSA. This measures the area under the curve of storage in use against elapsed time. For more information see "A note about storage occupancy counts" on page 560.

<i>Table 122. User storage field id cross reference</i>				
	UDSA	EUDSA	CDSA	ECDSA
Getmain count	054	105	117	120
High-water-mark	033	106	116	119
Occupancy	095	107	118	121

Program storage fields in group DFHSTOR:

087 (TYPE-A, 'PCSTGHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task both above **and** below the 16MB line.

108 (TYPE-A, 'PC24BHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task below the 16MB line. This field is a subset of PCSTGHWM (field ID 087) that resides below the 16MB line.

122 (TYPE-A, 'PC31RHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task above the 16MB line, in the extended read-only dynamic storage area (ERDSA). This field is a subset of PC31AHWM (field ID 139) that resides in the ERDSA.

139 (TYPE-A, 'PC31AHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task above the 16MB line. This field is a subset of PCSTGHWM (field ID 087) that resides above the 16MB line.

142 (TYPE-A, 'PC31CHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task above the 16MB line, in the extended CICS dynamic storage area (ECDSA). This field is a subset of PC31AHWM (139) that resides in the ECDSA.

143 (TYPE-A, 'PC24CHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task below the 16MB line, in the CICS dynamic storage area (CDSA). This field is a subset of PC24BHWM (108) that resides in the CDSA.

160 (TYPE-A, 'PC24SHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task below the 16MB line, in the shared dynamic storage area (SDSA). This field is a subset of PC24BHWM (108) that resides in the SDSA.

161 (TYPE-A, 'PC31SHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task above the 16MB line, in the extended shared dynamic storage area (ESDSA). This field is a subset of PC31AHWM (139) that resides in the ESDSA.

162 (TYPE-A, 'PC24RHWM', 4 BYTES)

Maximum amount (high-water mark) of program storage in use by the user task below the 16MB line, in the read-only dynamic storage area (RDSA). This field is a subset of PC24BHWM (108) that resides in the RDSA.

Performance data in group DFHSYNC

060 (TYPE-A, 'SPSYNCCT', 4 BYTES)

Number of SYNCPOINT requests issued during the user task.

Notes:

1. A SYNCPOINT is implicitly issued as part of the task-detach processing.
2. A SYNCPOINT is issued at PSB termination for DL/I and DBCTL.

Performance data in group DFHTASK

001 (TYPE-C, 'TRAN', 4 BYTES)

Transaction identification.

004 (TYPE-C, 'T', 4 BYTES)

Transaction start type. The high-order bytes (0 and 1) are set to:

- 'TO ' Attached from terminal input
- 'S ' Attached by automatic transaction initiation (ATI) without data
- 'SD' Attached by automatic transaction initiation (ATI) with data
- 'QD' Attached by transient data trigger level
- 'U ' Attached by user request
- 'TP' Attached from terminal TCTTE transaction ID
- 'SZ' Attached by Front End Programming Interface (FEPI).

007 (TYPE-S, 'USRDISPT', 8 BYTES)

Elapsed time for which the user task was dispatched. For more information see "Clocks and time stamps" on page 557.

008 (TYPE-S, 'USRCPUT', 8 BYTES)

Processor time for which the user task was dispatched under the CICS TCBs (QR, CO, or RO; also RP if the ONC RP Feature is installed and active). For more information see "Clocks and time stamps" on page 557.

014 (TYPE-S, 'SUSPTIME', 8 BYTES)

Total elapsed wait time for which the user task was suspended by the dispatcher. This includes:

- The elapsed time waiting for the first dispatch. This also includes any delay incurred because of the limits set for this transaction's transaction class (if any) or by the system parameter MXT being reached.
- The task suspend (wait) time.
- The elapsed time waiting for redispach after a suspended task has been resumed.

For more information, see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

031 (TYPE-P, 'TRANNUM', 4 BYTES)

Transaction identification number.

The transaction number field is normally a 4-byte packed decimal number. However, some CICS system tasks are identified by special character "transaction numbers", as follows:

- ' III' for system initialization task

- ' JBS' or ' Jnn' for journal control (where nn = the journal number from 01 - 99)
- ' TCP' for terminal control.

These special identifiers are placed in bytes 2 through 4. Byte 1 is a blank (X'40') before the terminal control TCP identifier, and a null value (X'00') before the others.

059 (TYPE-A, 'ICPUINCT', 4 BYTES)

Number of interval control START or INITIATE requests during the user task.

064 (TYPE-A, 'TASKFLAG', 4 BYTES)

Task error flags, a string of 31 bits used for signaling unusual conditions occurring during the user task:

- Bit 0** Reserved
- Bit 1** Detected an attempt either to start a user clock that was already running, or to stop one that was not running
- Bits 2–31** Reserved.

097 (TYPE-C, 'NETNAME', 20 BYTES)

Fully qualified name by which the terminal-owning CICS region is known to its VTAM. This name is assigned at attach time using either the NETNAME derived from the TCT (when the task is attached to a local terminal), or the NETNAME passed as part of an ISC APPC or IRC attach header. At least three padding bytes (X'00') are present at the right end of the name.

If the originating terminal is VTAM across an ISC APPC or IRC link, then the NETNAME is the *networkid.LUname*. If the terminal is non-VTAM, then the NETNAME is *networkid.generic_applid*.

All originating information passed as part of an ISC LUTYPE6.1 attach header has the same format as the non-VTAM terminal originators above.

When the originator is communicating over a DL/I batch session, the name is a concatenation of 'jobname.stepname.procname' derived from the originating system. Characters in excess of 17 cause truncation **at the left**.

When the originator is communicating over an external CICS interface (EXCI) session, the name is a concatenation of:

'DFHEXCIU'	.	MVS Id	Address Space Id (ASID)'
8 bytes	1 byte	4 bytes	4 bytes

derived from the originating system. That is, the name is a 17-byte LU name consisting of:

- An 8-byte eye-catcher set to 'DFHEXCIU'.
- A 1-byte field containing a period (.).
- A 4-byte field containing the MVSID, in characters, under which the MVS client program is running.
- A 4-byte field containing the address space id (ASID) in which the client program is running. (This is the 4-character EBCDIC representation of the 2-byte hex address space id.)

098 (TYPE-C, 'UOWID', 8 BYTES)

Name by which the unit of work is known within the originating system. This name is assigned at attach time using either an STCK-derived token (when the

task is attached to a local terminal), or the unit of work ID passed as part of an ISC APPC or IRC attach header.

The first six bytes of this field are one of the following:

- A binary value derived from the clock of the originating system and wrapping round at intervals of several months
- A character value of the form “hhmmss”, which wraps round daily. This case applies when the originating system is communicating through a DL/I batch session.

The last two bytes of this field are for the period count. These may change during the life of the task as a result of syncpoint activity.

Note: When using MRO or ISC, the UOWID field must be combined with the NETNAME field (097) to uniquely identify a task, because UOWID is unique only to the originating CICS system.

102 (TYPE-S, ‘DISPWTT’, 8 BYTES)

Elapsed time for which the user task waited for redispatch. This is the aggregate of the wait times between each event completion and user-task redispatch.

Note: This field does not include the elapsed time spent waiting for first dispatch. This field is a subset of the task suspend time, SUSPTIME (014), field.

109 (TYPE-C, ‘TRANPRI’, 4 BYTES)

Transaction priority when monitoring of the task was initialized (low-order byte-3).

125 (TYPE-S, ‘DSPDELAY’, 8 BYTES)

The elapsed time waiting for first dispatch.

Note: This field is a subset of the task suspend time, SUSPTIME (014), field. For more information see “Clocks and time stamps” on page 557.

126 (TYPE-S, ‘TCLDELAY’, 8 BYTES)

The elapsed time waiting for first dispatch which was delayed because of the limits set for this transaction’s transaction class, TCLSNAME (166), being reached. For more information see “Clocks and time stamps” on page 557.

Note: This field is a subset of the first dispatch delay, DSPDELAY (125), field.

127 (TYPE-S, ‘MXTDELAY’, 8 BYTES)

The elapsed time waiting for first dispatch which was delayed because of the limits set by the system parameter, MXT, being reached.

Note: The field is a subset of the first dispatch delay, DSPDELAY (125), field.

129 (TYPE-S, ‘ENQDELAY’, 8 BYTES)

The elapsed time waiting for a CICS Task Control ENQ. For more information see “Clocks and time stamps” on page 557.

Note: This field is a subset of the task suspend time, SUSPTIME (014), field.

166 (TYPE-C, ‘TCLSNAME’, 8 BYTES)

Transaction class name. This field is null if the transaction is not in a TRANCLASS.

170 (TYPE-S, 'RMITIME', 8 BYTES)

Amount of elapsed time spent in the Resource Manager Interface (RMI). For more information see "Clocks and time stamps" on page 557, and Figure 65 on page 560.

171 (TYPE-S, 'RMISUSP', 8 BYTES)

Amount of elapsed time the task was suspended by the dispatcher while in the Resource Manager Interface (RMI). For more information see "Clocks and time stamps" on page 557, "A note about wait times" on page 559, and Figure 65 on page 560.

Note: The field is a subset of the task suspend time, SUSPTIME (014), field and also the RMITIME (170) field.

Performance data in group DFHTERM**002 (TYPE-C, 'TERM', 4 BYTES)**

Terminal or session identifier. This field is null if the task is not associated with a terminal or session.

APAR PN71965

Documentation for PN71965 added on 11 September 1995.

Note: IF SURROGATE=YES is coded on the DFHMCT TYPE=INITIAL macro used to define the monitoring control table, and the task has been transaction-routed from a TOR that is directly-connected to this region, this field contains the identifier of the surrogate terminal.

009 (TYPE-S, 'TCIOWTT', 8 BYTES)

Elapsed time for which the user task waited for input from the terminal operator, after issuing a RECEIVE request. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

034 (TYPE-A, 'TCMSGIN1', 4 BYTES)

Number of messages received from the task's principal terminal facility, including LUTYPE6.1 and LUTYPE6.2 (APPC) but not MRO (IRC).

035 (TYPE-A, 'TCMSGOU1', 4 BYTES)

Number of messages sent to the task's principal terminal facility, including LUTYPE6.1 and LUTYPE6.2 (APPC) but not MRO (IRC).

067 (TYPE-A, 'TCMSGIN2', 4 BYTES)

Number of messages received from the LUTYPE6.1 alternate terminal facilities by the user task.

068 (TYPE-A, 'TCMSGOU2', 4 BYTES)

Number of messages sent to the LUTYPE6.1 alternate terminal facilities by the user task.

069 (TYPE-A, 'TCALLOCT', 4 BYTES)

Number of TCTTE ALLOCATE requests issued by the user task for LUTYPE6.2 (APPC), LUTYPE6.1, and IRC sessions.

083 (TYPE-A, 'TCCHRIN1', 4 BYTES)

Number of characters received from the task's principal terminal facility, including LUTYPE6.1 and LUTYPE6.2 (APPC) but not MRO (IRC).

084 (TYPE-A, 'TCCHROU1', 4 BYTES)

Number of characters sent to the task's principal terminal facility, including LUTYPE6.1 and LUTYPE6.2 (APPC) but not MRO (IRC).

085 (TYPE-A, 'TCCHRIN2', 4 BYTES)

Number of characters received from the LUTYPE6.1 alternate terminal facilities by the user task. *(Not applicable to ISC APPC.)*

086 (TYPE-A, 'TCCHROU2', 4 BYTES)

Number of characters sent to the LUTYPE6.1 alternate terminal facilities by the user task. *(Not applicable to ISC APPC.)*

100 (TYPE-S, 'IRIOWTT', 8 BYTES)

Elapsed time for which the user task waited for control at this end of an MRO link. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

111 (TYPE-C, 'LUNAME', 8 BYTES)

VTAM logical unit name (if available) of the terminal associated with this transaction. If the task is executing in an application-owning or file-owning region then the luname is the generic applid of the originating connection for LUTYPE6.1 and LUTYPE6.2 (APPC). The luname is blank if the originating connection is either MRO (ISC) or external CICS interface (EXCI).

+
+
+
+
+
+

APAR PN71965

Documentation for PN71965 added on 11 September 1995.

Note: IF SURROGATE=YES is coded on the DFHMCT TYPE=INITIAL macro used to define the monitoring control table, and the task has been transaction-routed from a TOR that is directly-connected to this region, this field relates to the session ID, *not* to the surrogate terminal returned in the TERM field (002).

133 (TYPE-S, 'LU61WTT', 8 BYTES)

The elapsed time for which the user task waited for I/O on a LUTYPE6.1 connection or session. This time also includes the waits incurred for conversations across LUTYPE6.1 connections, but not the waits incurred due to LUTYPE6.1 syncpoint flows. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

|
|

134 (TYPE-S, 'LU62WTT', 8 BYTES)

The elapsed time for which the user task waited for I/O on a LUTYPE6.2 (APPC) connection or session. This time also includes the waits incurred for conversations across LUTYPE6.2 (APPC) connections, but not the waits incurred due to LUTYPE6.2 (APPC) syncpoint flows. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

|
|

135 (TYPE-A, 'TCM62IN2', 4 BYTES)

Number of messages received from the alternate facility by the user task for LUTYPE6.2 (APPC) sessions.

136 (TYPE-A, 'TCM62OU2', 4 BYTES)

Number of messages sent to the alternate facility by the user task for LUTYPE6.2 (APPC) sessions.

137 (TYPE-A, 'TCC62IN2', 4 BYTES)

Number of characters received from the alternate facility by the user task for LUTYPE6.2 (APPC) sessions.

138 (TYPE-A, 'TCC62OU2', 4 BYTES)

Number of characters sent to the alternate facility by the user task for LUTYPE6.2 (APPC) sessions.

Performance data in group DFHTEMP**011 (TYPE-S, 'TSIOWTT', 8 BYTES)**

Elapsed time for which the user task waited for VSAM temporary storage I/O. For more information see "Clocks and time stamps" on page 557, and "A note about wait times" on page 559.

044 (TYPE-A, 'TSGETCT', 4 BYTES)

Number of temporary-storage GET requests issued by the user task.

046 (TYPE-A, 'TSPUTACT', 4 BYTES)

Number of PUT requests to auxiliary temporary storage issued by the user task.

047 (TYPE-A, 'TSPUTMCT', 4 BYTES)

Number of PUT requests to main temporary storage issued by the user task.

092 (TYPE-A, 'TSTOTCT', 4 BYTES)

Total number of temporary-storage requests issued by the user task. This field is the sum of TSGETCT, TSPUTACT, and TSPUTMCT.

Exception class data

Exception records are produced after each of the following conditions encountered by a transaction has been resolved:

- Wait for storage in the CDSA
- Wait for storage in the UDSA
- Wait for storage in the SDSA
- Wait for storage in the RDSA
- Wait for storage in the ECDSA
- Wait for storage in the EUDSA
- Wait for storage in the ESDSA
- Wait for storage in the ERDSA
- Wait for auxiliary temporary storage
- Wait for auxiliary temporary storage string
- Wait for auxiliary temporary storage buffer
- Wait for file string
- Wait for file buffer
- Wait for LSRPOOL string.

Exception records are fixed format (their format is shown on page 556).

Exception data

EXCMNTRN (TYPE-C, 4 BYTES)

Transaction identification. This field is null if the task is not associated with a terminal or session.

EXCMNTER (TYPE-C, 4 BYTES)

Terminal identification. This field is null if the task is not associated with a terminal or session.

EXCMNUSR (TYPE-C, 8 BYTES)

User identification at task creation. This can also be the remote user identifier for a task created as the result of receiving an ATTACH request across an MRO or APPC link with attach-time security enabled.

EXCMNTST (TYPE-C, 4 BYTES)

Transaction start type. The low-order byte (0 and 1) is set to:

- 'TO' Attached from terminal input
- 'S' Attached by automatic transaction initiation (ATI) without data
- 'SD' Attached by automatic transaction initiation (ATI) with data
- 'QD' Attached by transient data trigger level
- 'U ' Attached by user request
- 'TP' Attached from terminal TCTTE transaction ID
- 'SZ' Attached by Front End Programming Interface (FEPI).

EXCMNSTA (TYPE-T, 8 BYTES)

Start time of the exception.

EXCMNSTO (TYPE-T, 8 BYTES)

Finish time of the exception.

Note: The performance class exception wait time field, EXWTTIME (103), is a calculation based on subtracting the start time of the exception (EXCMNSTA) from the finish time of the exception (EXCMNSTO).

EXCMNTNO (TYPE-P, 4 BYTES)

Transaction identification number.

EXCMNTPR (TYPE-C, 4 BYTES)

Transaction priority when monitoring was initialized for the task (low-order byte).

EXCMNLUN (TYPE-C, 4 BYTES)

VTAM logical unit name (if available) of the terminal associated with this transaction. This field is nulls if the task is not associated with a terminal.

EXCMNEXN (TYPE-A, 4 BYTES)

Exception sequence number for this task.

EXCMNRTY (TYPE-C, 8 BYTES)

Exception resource type. The possible values for EXCMNRTY are shown in Table 123 on page 577.

EXCMNRID (TYPE-C, 8 BYTES)

Exception resource identification. The possible values for EXCMNRID are shown in Table 123 on page 577.

EXCMNTYP (TYPE-A, 2 BYTES)

Exception type. This field can be set to one of the following values:

- X'0001' Exception due to a wait (EXCMNWT)
- X'0002' Exception due to a buffer wait (EXCMNBWT)
- X'0003' Exception due to a string wait (EXCMNSWT)

EXCMNTCN (TYPE-C, 8 BYTES)

Transaction class name. This field is null if the transaction is not in a transaction class.

The following table shows the value and relationships of the fields EXCMNTYP, EXCMNRTY, and EXCMNRID.

<i>Table 123. Possible values of EXCMNTYP, EXCMNRTY, and EXCMNRID. The relationship between exception type, resource type, and resource identification.</i>			
EXCMNTYP	EXCMNRTY	EXCMNRID	MEANING
Exception type	Resource type	Resource ID	
EXCMNWT	'STORAGE'	'UDSA'	Wait for UDSA storage
EXCMNWT	'STORAGE'	'EUDSA'	Wait for EUDSA storage
EXCMNWT	'STORAGE'	'CDSA'	Wait for CDSA storage
EXCMNWT	'STORAGE'	'ECDSA'	Wait for ECDSA storage
EXCMNWT	'STORAGE'	'SDSA'	Wait for SDSA storage
EXCMNWT	'STORAGE'	'ESDSA'	Wait for ESDSA storage
EXCMNWT	'STORAGE'	'RDSA'	Wait for RDSA storage
EXCMNWT	'STORAGE'	'ERDSA'	Wait for ERDSA storage
EXCMNWT	'TEMPSTOR'	TS Qname	Wait for temporary storage
EXCMNSWT	'FILE'	filename	Wait for string associated with file
EXCMNSWT	'LSRPOOL'	filename	Wait for string associated with LSRPOOL
EXCMNSWT	'TEMPSTOR'	TS Qname	Wait for string associated with DFHTEMP
EXCMNBWT	'LSRPOOL'	LSRPOOL	Wait for buffer associated with LSRPOOL
EXCMNBWT	'TEMPSTOR'	TS Qname	Wait for buffer associated with DFHTEMP

monitoring data

Chapter 20. CICS statistics

This chapter is divided into the following sections:

1. “**Introduction to CICS statistics**” describes the types of statistics data, and the use of the EXEC CICS COLLECT STATISTICS command.
2. “**CICS statistics record format**” on page 583 describes the format of CICS statistics SMF type 110 records.
3. “**Global user exit in the CICS statistics domain**” on page 588 suggests ways in which you can use the XSTOUT statistics exit.
4. “**Processing the output from CICS statistics**” on page 588 lists the methods of processing statistics data.

Introduction to CICS statistics

CICS statistics contain information about the CICS system as a whole—for example, its performance and usage of resources. Statistics data is therefore useful both for performance tuning and for capacity planning.

Statistics are collected during CICS online processing for later offline analysis. The statistics domain writes statistics records to a System Management Facility (SMF) data set. The records are of SMF type 110, subtype 0002.¹⁰

Types of statistics data

CICS produces five types of statistics: **interval**, **end-of-day**, **requested**, **requested reset**, and **unsolicited**.

Interval statistics

are gathered by CICS during a specified interval. CICS writes the interval statistics to the SMF data set automatically at the expiry of that interval if:

- Statistics recording status was set ON by the STATRCD system initialization parameter (and has not subsequently been set OFF by a CEMT SET STATISTICS or EXEC CICS SET STATISTICS RECORDING command). The default value of STATRCD is OFF.

or

- ON is specified on CEMT SET STATISTICS.

or

- The RECORDING option of the EXEC CICS SET STATISTICS command is set to ON.

When interval statistics are written, the statistics counters are reset. See “Resetting statistics counters” on page 582.

¹⁰ Monitoring records are also written to the SMF data set as type 110 records. (Some journaling type 110 records can be written there, too.) You might find it useful to process the statistics records and the monitoring records together, because statistics provide resource and system information that is complementary to the transaction data produced by CICS monitoring.

You can change the interval duration using CEMT SET STATISTICS and the EXEC CICS SET STATISTICS command. The default interval duration is 3 hours for a cold start of CICS.

End-of-day statistics

are a special case of interval statistics. They are the statistics for the duration between the last time the statistics counters were reset and:

- The end-of-day expiry time, or
- When CICS quiesces (normal shutdown), or
- When CICS terminates (immediate shutdown).

For details of the events at which statistics counters are reset, see “Resetting statistics counters” on page 582.

The end-of-day value defines a logical point in the 24-hour operation of CICS. You can change the end-of-day value using CEMT SET STATISTICS or the EXEC CICS SET STATISTICS command. CICS writes end-of-day statistics to SMF even if, on one of the following, you have specified OFF:

- The STATRCD system initialization parameter
- The CEMT SET STATISTICS command
- The RECORDING option of the EXEC CICS SET STATISTICS command.

The default end-of-day value is 12 midnight. When end-of-day statistics are written, the statistics counters are reset.

Requested statistics

are statistics which the user has asked for using one of the following commands:

- CEMT PERFORM STATISTICS RECORD
- EXEC CICS PERFORM STATISTICS RECORD
- EXEC CICS SET STATISTICS ON|OFF RECORDNOW.

CICS writes requested statistics to SMF even if, on one of the following, you have specified OFF:

- The STATRCD system initialization parameter
- The CEMT SET STATISTICS command
- The RECORDING option of the EXEC CICS SET STATISTICS command.

Statistics counters are not reset.

Requested reset statistics

differ from requested statistics in that all statistics are collected, and all statistics counters are reset. They are invoked by either of the commands:

- CEMT PERFORM STATISTICS RECORD ALL(RESETNOW)
- EXEC CICS PERFORM STATISTICS RECORD ALL(RESETNOW).

You can also invoke requested reset statistics when setting statistics recording status ON or OFF, using either of the commands:

- CEMT SET STATISTICS ON|OFF RECORDNOW RESETNOW
- EXEC CICS SET STATISTICS ON|OFF RECORDNOW RESETNOW.

Note that it is valid to specify the RECORDNOW RESETNOW options only when there is a genuine change of recording status. For example, coding

EXEC CICS SET STATISTICS ON RECORDNOW RESETNOW when STATISTICS is already set ON causes an error response.

CICS writes requested reset statistics to SMF even if, on one of the following, you have specified OFF:

- The STATRCD system initialization parameter
- The CEMT SET STATISTICS command
- The RECORDING option of the EXEC CICS SET STATISTICS command.

Unsolicited statistics

are automatically gathered by CICS for dynamically allocated and deallocated resources. CICS writes these statistics to SMF just before the resource is deleted, regardless of the status of statistics recording.

CICS collects unsolicited statistics for:

Autoinstall

Whenever an autoinstalled terminal entry in the TCT is deleted, CICS collects statistics covering the autoinstalled period since the last interval. This period includes any delay interval specified on the system initialization parameters AILDELAY or AIRDELAY.

If an autoinstall terminal logs on again before the expiry of the delay interval, then the accumulation of statistics continues until the next interval. At that interval, the accumulation of statistics is restarted.

DBCTL

Whenever CICS disconnects from DBCTL, CICS collects the statistics covering the whole of the DBCTL connection period.

FEPI nodes

Whenever an installed FEPI node definition is discarded, CICS collects the statistics covering the installed period since the last interval.

FEPI pools

Whenever an installed FEPI pool definition is discarded, CICS collects the statistics covering the installed period since the last interval.

FEPI targets

Whenever an installed FEPI target definition is discarded, CICS collects the statistics covering the installed period since the last interval.

Files

Whenever CICS closes a file, CICS collects statistics covering the period from the last interval.

LSRPOOL files

APAR PN65430

Documentation for PN65430 added on 19 December 1994.

Whenever CICS closes a file which is in a LSRPOOL, it collects LSRPOOL file statistics (as well as the file statistics), covering the period from the last interval.

#

|
|

|
|
|
|

|
|
|
|

|
|
|
|

+
+
+

+
+
+

LSRPOOLS

When CICS closes the last file in an LSRPOOL, it collects the statistics for the LSRPOOL.

Note that the following peak values are reset at each interval collection:

- Peak number of requests waiting for a string
- Maximum number of concurrent active file control strings.

However, the other statistics, which are not reset at an interval collection, cover the entire period from the time the LSRPOOL is created (when the first file is opened) until the LSRPOOL is deleted (when the last file is closed).

Programs

Whenever an installed program definition is discarded, CICS collects the statistics covering the installed period since the last interval.

Transactions

Whenever an installed transaction definition is discarded, CICS collects the statistics covering the installed period since the last interval.

Transaction classes

Whenever an installed transaction class definition is discarded, CICS collects the statistics covering the installed period since the last interval.

For information about how to use the CEMT statistics commands, refer to the *CICS/ESA CICS-Supplied Transactions* manual. For programming information about the EXEC CICS statistics commands, see the *CICS/ESA System Programming Reference* manual.

Resetting statistics counters

Statistics counters are reset in the following circumstances:

- At CICS startup.
- When interval statistics are written (but **not** when an interval occurs and no statistics are written).
- At end of day.
- When requested reset statistics are written.

However, you can cause statistics counters to be reset without writing records to the SMF data set. You do this by changing the statistics recording status, using either of the commands:

- CEMT SET STATISTICS ON|OFF RESETNOW
- EXEC CICS SET STATISTICS ON|OFF RESETNOW.

Note that it is valid to specify the RESETNOW option only when there is a genuine change of recording status. For example, coding EXEC CICS SET STATISTICS ON RESETNOW when STATISTICS is already set ON causes an error response.

Important

Statistics counters are reset in various ways. Specific counters may be reset to:

- 0
- 1
- A new peak value
- Not reset
- None of the above.

For information about the resetting of specific statistics counters, refer to the *CICS/ESA Performance Guide*.

The EXEC CICS COLLECT STATISTICS command

In addition to the types of statistics data described above, there is an online EXEC CICS COLLECT STATISTICS function. Online statistics are collected and returned to the invoking application.

The three sample programs DFH0STAT, DFH\$STCN, and DFH\$STAS show how you can use the EXEC CICS COLLECT STATISTICS and EXEC CICS INQUIRE commands to produce a useful analysis of a CICS/ESA system. The programs produce a report showing critical system parameters from the CICS dispatcher, together with loader statistics and an analysis of the CICS storage manager. DFH0STAT is provided in VS COBOL II; DFH\$STCN and DFH\$STAS are provided in assembler language.

For programming information about the COLLECT STATISTICS command, see the *CICS/ESA System Programming Reference* manual.

For information about installing and operating the sample statistics programs, see the *CICS/ESA System Definition Guide*. For information about the data produced by the programs, see the *CICS/ESA Performance Guide*.

CICS statistics record format

This section describes the format of CICS statistics SMF type 110 records in detail. You need this information if you write your own program to analyze the statistics data. The three components of a CICS statistics record are an SMF header, an SMF product section, and a CICS data section, as shown in Figure 68. Each of these is described in the sections that follow.

SMF Header	SMF Product Section	CICS Data Section
---------------	------------------------	----------------------

Figure 68. Format of an SMF type 110 statistics record

SMF header and SMF product section

The SMF header describes the system creating the output. The SMF product section identifies the subsystem to which the statistics data relates, which, in the case of CICS statistics, is the CICS region. Both the SMF header and the SMF product section can be mapped by the DSECT STSMFDS, which you can generate using the DFHSTSMF macro as follows:

```
STSMFDS DFHSTSMF PREFIX=SMF
```

The label 'STSMFDS' is the default DSECT name, and SMF is the default PREFIX value, so you could also generate the DSECT simply by coding

```
DFHSTSMF
```

.

The STSMFDS DSECT has the format shown in Figure 69.

```
*          START THE SMF HEADER
*
STSMFDS  DSECT
SMFSTLEN DS   XL2          RECORD LENGTH
SMFSTSEQ DS   XL2          SEGMENT DESCRIPTOR
SMFSTFLG DS   X           OPERATING SYSTEM INDICATOR (see note 1)
SMFSTRTY DC  X'6E'        RECORD TYPE 110 FOR CICS
SMFSTTME DS   XL4          TIME RECORD MOVED TO SMF
SMFSTDTE DS   XL4          DATE RECORD MOVED TO SMF
SMFSTSID DS   XL4          SYSTEM IDENTIFICATION
SMFSTSSI DS   CL4'CICS'    SUBSYSTEM IDENTIFICATION
SMFSTSTY DS   XL2          RECORD SUBTYPE X'0002' FOR STATISTICS
SMFSTRN  DS   XL2          NUMBER OF TRIPLETS
          DS   XL2          RESERVED
SMFSTAPS DS   XL4          OFFSET TO PRODUCT SECTION
SMFSTLPS DS   XL2          LENGTH OF PRODUCT SECTION
SMFSTNPS DS   XL2          NUMBER OF PRODUCT SECTIONS
SMFSTASS DS   XL4          OFFSET TO DATA SECTION
SMFSTASL DS   XL2          LENGTH OF DATA SECTION
SMFSTASN DS   XL2          NUMBER OF DATA SECTIONS
*
*          THIS CONCLUDES THE SMF HEADER
*
```

Figure 69 (Part 1 of 2). Format of the SMF header and product section for statistics records


```

*          START THE SMF PRODUCT SECTION
*
SMFSTRVN DS   XL2          RECORD VERSION
SMFSTPRN DS   CL8          PRODUCT NAME (GENERIC APPLID)
SMFSTSPN DS   CL8          PRODUCT NAME (SPECIFIC APPLID)
SMFSTMFL DS   XL2          RECORD MAINTENANCE INDICATOR
          DS   XL2          RESERVED
          DS   XL2          RESERVED
SMFSTDTK DS   XL4          DOMAIN TOKEN
SMFSTDID DS   CL2          DOMAIN ID
SMFSTRQT DS   CL3          USS/EOD/REQ/INT STATISTICS TYPE
SMFSTICD DS   CL3          YES IF INCOMPLETE DATA RECORDED
+          ***** APAR PN69653 Documentation added 8 December 1995. *****
+          SMFSDAT DS   CL8          COLLECTION DATE MMDDYYYY
          SMFSTCLT DS   CL6          COLLECTION TIME HHMMSS
          SMFSTINT DS   CL6          INTERVAL HHMMSS. See note 3.
          SMFSTINO DS   XL4          INTERVAL NUMBER. See note 3.
          SMFSTRTK DS   XL8          REQUEST TOKEN
          SMFSTLRT DS   CL6          LAST RESET TIME HHMMSS
          SMFSTCST DS   XL8          CICS START TIME
          SMFSTJBN DS   CL8          JOBNAME
          SMFSTRSD DS   XL4          JOB DATE
          SMFSTRST DS   XL4          JOB TIME
          SMFSTUIF DS   CL8          USER IDENTIFICATION
          SMFSTPDN DS   CL8          OPERATING SYSTEM PRODUCT LEVEL
*
*          THIS CONCLUDES THE SMF PRODUCT SECTION

```

Figure 69 (Part 2 of 2). Format of the SMF header and product section for statistics records

Notes:

1. CICS sets only the subsystem-related bits of the operating system indicator flag byte in the SMF header (SMFSTFLG). SMF sets the remainder of the byte according to the operating system level and other factors. For an explanation of the setting of the other bits, refer to the *MVS/ESA System Management Facilities* manual.
2. The copy book DFHSMFDS is also provided and can be used to map the SMF header and the SMF product sections of all three subtypes of SMF 110 records written by CICS journaling, CICS monitoring, and CICS statistics.
3. Fields SMFSTINT and SMFSTINO are only relevant if SMFSTRQT is 'INT'. Otherwise both values should be ignored.

CICS statistics data section

The format of the CICS statistics data section is shown in Figure 70.

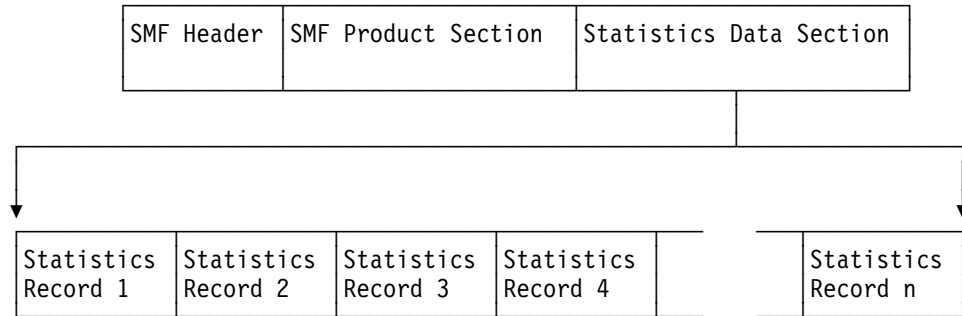


Figure 70. Format of the statistics data section

If the data records are incomplete, the flag field SMFSTICD is set to 'YES'. In this case, the statistics data section is not present.

For complete data records, the statistics data section is made up of one or more statistics data records. There are different formats of data records. Each has a common format for the first 5 bytes. These 5 bytes are described in the extract from copy book DFHSTIDS in Figure 71.

DFHSTIDS	DSECT		Statistics record header
*			
	DS	0F	Fullword alignment
STILEN	DS	H	Length of the record
STID	DS	AL2	Statistics identifier
STIVERS	DS	CL1	Statistics record version

Figure 71. Extract from copy book DFHSTIDS

STILEN

is the length of the data record.

STID

identifies which type of statistics record you have (see Figure 72 on page 587).

You can use the STID symbolic name or value to determine which copy book to use when processing the statistics data records. For details about the relationship between the STID name or value and the copy book, see Figure 72 on page 587. For further guidance information about the fields within the statistics data records, see the *CICS/ESA Performance Guide*.

STIVERS

takes the value '1' for this release of CICS.

STID Symbolic name	STID Value	Copy book	Type of record
STISMDSA	2	DFHSMDS	Storage manager DSA id
STISM	5	DFHSMDDS	Storage mgr domain subpool id
STISMT	6	DFHSMTDS	Storage manager task subpool id
STIXMG	10	DFHXMGDS	Transaction manager (Globals) id
STIXMR	11	DFHXRDS	Transaction manager (Trans) id
STIXMC	12	DFHXMCD	Transaction manager (Tclass) id
STIFEPIP	16	DFHA22DS	FEPI pool id
STIFEPI	17	DFHA23DS	FEPI connection id
STIFEPI	18	DFHA24DS	FEPI target id
STIVT	21	DFHA03DS	VTAM stats id
STIAUSS	22	DFHAUSDS	Terminal Autoinstall USS id
STIPAU	23	DFHPPGDS	Program Autoinstall id
STIAU	24	DFHA04DS	Terminal Autoinstall stats id
STILDR	25	DFHLDRDS	Loader (Resid) id
STIDBU	28	DFHDBUD	DBCTL USS id
STILDG	30	DFHLDGDS	Loader (Globals) id
STIDTB	33	DFHA05DS	DTB statistics id
STITCR	34	DFHA06DS	Terminal control (resid) id
STILSRR	39	DFHA08DS	LSRPOOL pool stats (resid) id
STILSRFR	40	DFHA09DS	LSRPOOL File statistics (by file)
STITDQR	43	DFHA10DS	TDQUEUE (Resid) id
STITDQG	45	DFHA11DS	TDQUEUE (globals) id
STITSQ	48	DFHA12DS	TSQUEUE statistics id
STIJCR	49	DFHA13DS	Journal control (Resid) id
STICONSR	52	DFHA14DS	ISC/IRC system entry (resid) id
STICONSS	54	DFHA21DS	ISC connection - system security
STIDS	56	DFHDSGDS	Dispatcher stats id
STIUSG	61	DFHUSGDS	User domain stats id
STITM	63	DFHA16DS	Table manager statistics id
STIST	66	DFHSTGDS	Statistics statistics id
STIFCR	67	DFHA17DS	File Control (resid) id
STIDLIG	72	DFHA25DS	DL/I (Global) id
STIDLIR	73	DFHA18DS	DL/I (Resid) id
STIIRCB	75	DFHA19DS	IRC Batch statistics id
STICONMR	76	DFHA20DS	ISC/IRC mode entry (resid) id
STIMNR	80	DFHMNTDS	Monitoring stats (Resid) id
STIM	81	DFHMNGDS	Monitoring stats (global) id
STITDR	85	DFHTDRDS	Transaction dump (resid) id
STITDG	87	DFHTDGDS	Transaction dump (global) id
STISDR	88	DFHSDRDS	System dump (resid) id
STISDG	90	DFHSDGDS	System dump (global) id

Figure 72. Statistics data record copy books related to STID name and value

Global user exit in the CICS statistics domain

There is one global user exit point (XSTOUT) in the CICS statistics domain. The exit is invoked before the contents of a statistics data buffer is written to SMF. At this exit, the following information is available:

- The address of the statistics buffer
- The length of the statistics buffer
- The address of the statistics type.

This applies to all five types of statistics: interval, end-of-day, requested, requested reset, and unsolicited statistics.

If you write a global user exit program to be invoked at this exit, you can examine this information and tell CICS either to write the contents of the buffer to SMF or to suppress its output.

For more information about global user exits in general, and about the statistics exit in particular, refer to Chapter 1, “Global user exit programs” on page 3.

Processing the output from CICS statistics

There are several utilities to help you process statistics output. You can use:

The CICS-supplied DFHSTUP program

For information about how to run DFHSTUP, refer to the *CICS/ESA Operations and Utilities Guide*. For information about how to interpret the report produced by DFHSTUP, see the *CICS/ESA Performance Guide*.

Your own program

to report and analyze the data in the statistics records.

Service Level Reporter (Version 3 Release 3, program number 5665-397)

This runs under MVS, and can be used to process CICS statistics output, combining it with data from system management facilities (SMF) records and producing joint reports. For further guidance information about SLR, refer to the *Service Level Reporter General Information* manual.

Enterprise Performance Data Manager/MVS

The SystemView Enterprise Performance Data Manager/MVS (EPMD) enables you to store CICS statistics (and other data) into a DB2 data set, and to present the data in a variety of forms. For information about EPMD, see the *CICS/ESA Performance Guide*.

Part 6. Customizing CICS compatibility interfaces

A general note about user-written programs	590	Chapter 22. The dynamic allocation	
Chapter 21. Using TCAM with CICS	591	sample program	613
CICS with TCAM SNA	592	Installing the program and transaction	
The TCAM application program interface	594	definitions	614
The CICS–TCAM interface	596	Terminal operation	614
TCAM devices	604	Help feature	614
TCAM user exits	607	Values	615
Starting and terminating TCAM	607	The flow of control when a DYNALLOC	
CICS and TCAM: program interrelationship	609	request is issued	616

A general note about user-written programs

The following comment applies to all user-written programs mentioned in Part 6 of this book:

- Upon return from any customer-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

Chapter 21. Using TCAM with CICS

This chapter describes the use of the telecommunications access method (TCAM) with CICS/ESA 4.1.

Important

CICS/ESA 4.1 supports only the DCB interface to TCAM, and **not** the ACB interface. This means that, in order for CICS to use it, TCAM has to be run as a VTAM application—it cannot be used independently of ACF/VTAM.

The chapter is divided into the following sections:

1. **“CICS with TCAM SNA”** on page 592 describes the use of TCAM in an SNA network, with reference to protocol management, FMH processing, and error processing.
2. **“The TCAM application program interface”** on page 594 includes information about the process control block and the TPROCESS control block.
3. **“The CICS–TCAM interface”** on page 596 includes information about terminal entries (TCTTEs) and line entries (TCTLEs) data flow, logic flow, the terminal error program, message routing, pooling, and segment processing.
4. **“TCAM devices”** on page 604 deals with message formats for devices (in particular, the 3270-system devices) being used on a TCAM line.
5. **“TCAM user exits”** on page 607 describes the three TCAM exits that may be specified in the terminal control program.
6. **“Starting and terminating TCAM”** on page 607 describes the processes of starting up, restarting after an abend, and terminating TCAM under CICS.
7. **“CICS and TCAM: program interrelationship”** on page 609 describes the TCAM message control program (MCP) and its relationship to the application program (in this case, CICS).

The majority of independent teleprocessing applications require a dedicated network. The telecommunications access method (TCAM) permits multiple applications to share a single network, resulting in more efficient use of terminals and lines. The CICS–TCAM interface enables CICS to run as an application program under TCAM.

One practical use of the CICS–TCAM interface is to run a production CICS system in one region and a test CICS system in another. If they run in separate regions, the applications are protected from one another. Operating under TCAM, terminals and lines can be shared by the two CICS applications. Other TCAM applications, such as the time sharing option (TSO), can also run concurrently.

CICS user tasks that run under BTAM can, in general, run under TCAM without modification to the task code. This assumes that you have correctly designed and coded the TCAM message control program. However, to obtain the benefits of TCAM SNA and to maintain an acceptable operator interface, it is usually necessary to change the CICS application programs to use EXEC CICS CONVERSE and EXEC CICS SEND LAST facilities so that the MCP is provided

with sufficient information about the transaction to maintain the optimum SNA message flows.

CICS with TCAM SNA

The CICS–TCAM interface has an enhanced data stream support that enables an appropriate TCAM message control program (MCP) to control the SNA session. The TCAMFET=SNA operand in DFHTCT TYPE=LINE allows TCTTEs to be specified for SNA devices. You must be prepared to write an appropriate TCAM SNA message control program to complement the CICS support and the SNA devices attached to the system. In order to obtain a good operator interface, the CICS application programs should be designed to inform the MCP of their intentions. Thus, it is better to design the MCP and the application programs together.

Sample TCAM SNA MCPs are provided in CICS410.SDFHSAMP. The second sample MCP (DFHSPTM2) uses the information passed in the CCB to optimize the message flows to the actual logical unit. This represents transaction-oriented processing.

TCAM provides data stream support for SNA devices running under CICS. Both the SNA character string (SCS) and the 3270 data streams are supported.

To understand how CICS works with TCAM in an SNA environment, it is important to understand the TCAM SNA structure. The device message handler (DMH) is the logical unit in SNA terms. All data flow control (DFC), session startup and shutdown, and response handling are provided in the DMH. There is no CICS control of these SNA functions, so the application programmer need not be concerned with them. For a more detailed discussion of the TCAM SNA functions provided, see the *ACF/TCAM System Programmer's Guide*.

Protocol management

There may be many different protocols in an SNA network. The various protocols are established on a session basis using the BIND image. You decide which protocols to use with which SNA session, and you should understand the requirements of the installation's application programs before deciding on a specific protocol.

Some of the more common of these SNA protocols are: bracket, half-duplex flip-flop (HDX-FF), and half-duplex contention (HDX-CON). The enforcement of these protocols is a function of the device message handler (DMH).

There are two methods of protocol management in a CICS–TCAM system:

- Device message handler control
- Transaction control.

These methods are discussed in the sections that follow.

Device message handler control

The device message handler method of protocol management is used when the transaction does not need to know which device it is communicating with. Although the communication control bytes (CCBs) are passed between CICS and TCAM, they are not used to control the SNA session. All the protocol control is provided in the DMH. You (the message handler writer, not the application programmer) choose the appearance at the outboard LU.

Transaction control

In the transaction control method of protocol management, the transaction controls the protocol. The SNA session should be bound with a protocol of HDX-FF with brackets when running this type of management.

When using transaction control, the communication control byte (CCB) is used to relay information from the transaction to the DMH. For example:

- EXEC CICS SEND LAST should be used to end a transaction. This causes an indicator to be set in the CCB requesting that the DMH send an end-of-bracket (EB) character.
- EXEC CICS CONVERSE should be used when terminal input is required after a SEND request. This causes an indicator to be set in the CCB requesting that the DMH send the CHANGE DIRECTION indicator to the device.
- EXEC CICS ISSUE DISCONNECT should be used to end the logical unit session. This causes an indicator to be set in the CCB requesting that the DMH terminate the LU-LU session (that is, issue the IEDHALT macro).

Function management header processing

The function management header (FMH) enables function management information to be directed to particular components within the logical unit. The FMH also provides a mechanism in which control information relating to the operation of those components may be passed. FMH processing is a BIND-time option (that is, a BIND parameter is available to indicate whether an FMH may or may not appear in the LU-LU session).

CICS–TCAM SNA provides support for the logical device code (LDC), which is transmitted in the FMH to the logical unit. The LDC provides for the communication of the logical disposition of output to the logical unit. It can represent any meaning that is useful to the installation.

There are two ways that FMH handling can be provided. The first is for the transaction to provide the FMH as part of the data passed to TCAM, by issuing an EXEC CICS SEND FMH command. An indicator is set in the CCB so that the DMH can set the “FMH included” indicator in the request handler (RH) by using the IEDRH macro. On input, the DMH should interrogate the RH (using the IEDRH macro) to determine whether an FMH is included in the data. If the FMH indicator is set in the RH, the DMH should set the FMH indicator in the CCB relating to the transaction in which the input data contains an FMH.

A second method of FMH handling is to provide the entire function in the DMH. The DMH should remove the FMH before passing the input data to the transaction, and insert the necessary FMH into the output data. For the DMH to build the correct FMH for output, some form of private interface must be established between the DMH and the application program. For example, the first byte of data following

the CCB can contain unique values that request specific FMH functions such as begin data set, erase record, and so on.

It is recommended that if FMH processing is required, the transaction (or preferably BMS) be used to provide the appropriate FMH.

Batch processing

When running a batch logical unit, a point to consider is how to get the transaction identification to CICS on the “begin data set” condition. The alternative methods are discussed below.

The first method is for the DMH to recognize the “begin data set” condition by interrogating the FMH and by editing the transaction ID into the input data. This method is demonstrated in the sample message control programs DFHSPTM1 and DFHSPTM2.

The second method of providing the transaction ID is for the DMH to concatenate the “begin data set” chain with the first chain of the data set, using the SETEOM macro. When you use this method, the first chain of the data set must contain the transaction ID. Alternatively, the transaction ID could be set with the TCTTE beforehand by means of a permanent TRANSID or by using EXEC CICS RETURN TRANSID.

Error processing for batch logical units

During batch processing with a logical unit, there are some logical errors from which the DMH cannot recover (for example, ‘data set overflow’ or ‘incorrect data set name’). A transaction can be provided to handle these error conditions. If the transaction builds the data set on the TCAM queue and ends before the data set is transmitted, an error transaction should be created. The DMH should generate the appropriate error message or pass the SNA sense bytes to this error transaction, which then handles the error condition. If the transaction that builds the data set remains active throughout the transmission of that data set to the device, then the same transaction could be coded to recognize the error indicators passed to it from the DMH, rather than a separate error transaction created for that purpose.

Error processing

All error conditions, other than logical errors, are handled by the DMH. The *ACF/TCAM System Programmer's Guide* contains a discussion on the handling of the various sense codes returned by SNA devices. The transaction is not involved in error processing and recovery.

The TCAM application program interface

The TCAM application program interface is a portion of the TCAM message control program (MCP). It consists of two types of control blocks, the process control block (PCB) and the TPROCESS block.

The PCB defines the application program interface of a region in the system using TCAM. Its purpose is to control communication and storage protection across region boundaries. It also defines the user-written message handler (MH) responsible for processing messages to and from the application program.

Because a PCB is required for each application program running with the MCP, one PCB is also required to define the CICS application program.

The TPROCESS control block controls communication to and from the application program. A separate block is required for both input and output to the application program. A TPROCESS block is required for each input queue to CICS, and for each output queue from CICS. In CICS, there are corresponding terminal control table line entries (TCTLEs) for each input queue, and for each output queue (that is, for each TPROCESS block).

DD cards (such as those shown in Figure 73) are used to correlate the TCAM control blocks with the CICS control blocks. The CICS terminal control table contains the DCB. The DDNAME specified in the terminal control table macro (DFHTCT TYPE=SDSCI,DDNAME=name) names the DD card. In the DD card, the QNAME field names the TCAM TPROCESS block.

You do not need to make any exceptions for CICS to the TCAM application program interface described above. For more information, see the *ACF/TCAM Version 3 Application Programming* manual.

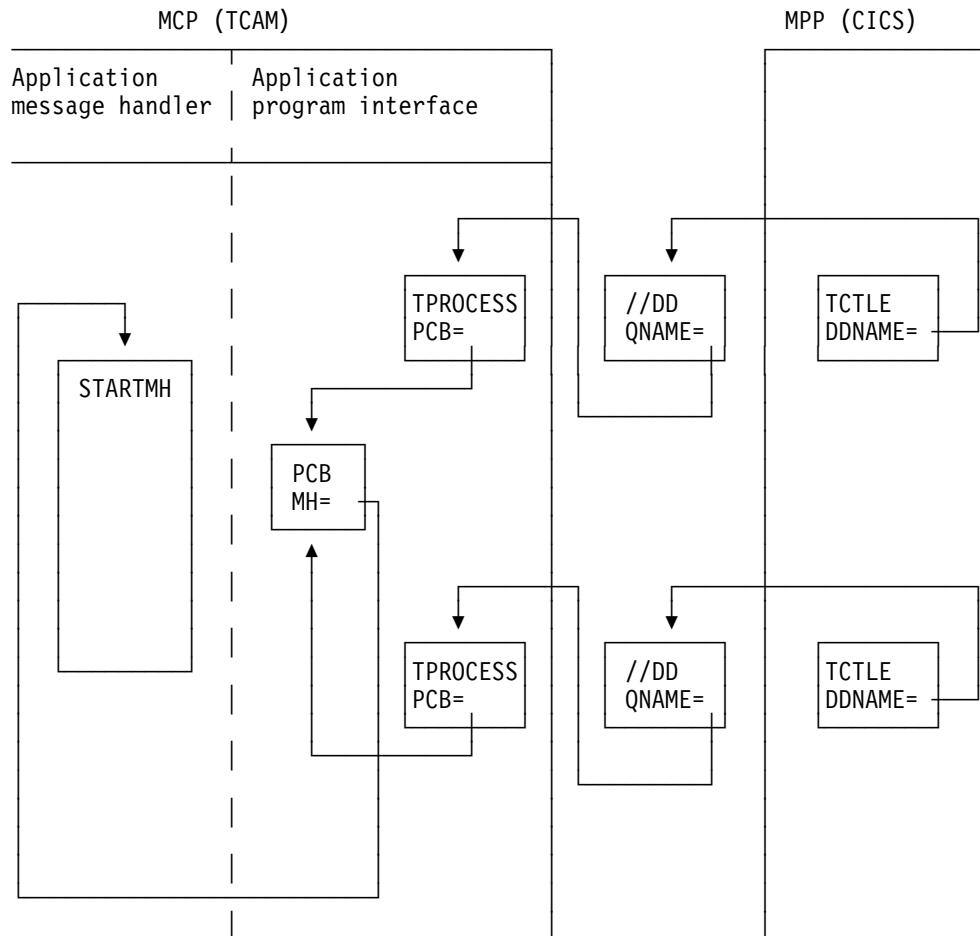


Figure 73. DD card correlation of TCAM and CICS control blocks

The CICS–TCAM interface

CICS treats a TCAM input process queue as a “line”. For each input process queue there is a CICS terminal control table line entry (TCTLE). Note that TCAM requires the application program (CICS) to have a DCB for each TPROCESS block; separate TPROCESS blocks are required for input to and output from the application program. Therefore, each TCAM output process queue is also treated as a line and has a corresponding CICS TCTLE. Each TCTLE references its own DCB, which is generated by the DFHTCT TYPE=SDSCI macro in CICS.

The CICS terminal control table terminal entries (TCTTEs) define the terminals associated with a particular line entry (TCTLE). For each physical terminal communicating with CICS through TCAM, a corresponding TCTTE containing the terminal identification must be associated with a TCTLE. Duplicating individual TCTTEs for both the input TCTLE and the output TCTLE is avoided by attaching a single, special TCTTE to the input TCTLE and attaching all the individual TCTTEs to the output TCTLE. Although attached to the output TCTLE, they are used for both input and output processing.

Each input record from TCAM must contain the source terminal identification. Using this identification as a search argument, the corresponding TCTTE can be located by CICS by comparing against the NETNAME value for each TCTTE.

Note: The usual way of ensuring that the input records contain the source terminal identification is to specify OPTCD=W in the CICS DFHTCT TYPE=SDSCI macro. If this specification is omitted, the TCAM user is responsible for ensuring that the record contains a suitable source terminal identification.

Using the POOL=YES operand of the DFHTCT TYPE=LINE macro, you can establish a pool of common TCTTEs on the output TCTLE that do not contain terminal identifiers. As required, terminal identifiers are assigned to the TCTTEs or removed from association with the TCTTEs. POOL=YES necessarily imposes a number of restrictions and should be thoroughly understood before being implemented. For additional information, see the discussion of the POOL operand in “Line pool specifications” on page 601.

Data format

When TCAM is specified, CICS assumes that the user transaction data passed to it from the TCAM queue is in the proper format to be passed directly to the user task. Except for the removal of the source terminal identification and the 2-byte CCB (if on a TCAM SNA line), CICS does not alter the data it receives. It is your responsibility (using your message control program (MCP)) to prepare the data, for example, by translating to EBCDIC, removing function management headers (FMHs), stripping line control characters, and deblocking. You may optionally bypass the CICS routine that removes the source terminal identification, by returning from the user-written input exit (XTCTIN) in TCP with a displacement of 0 bytes.

Similarly, CICS assumes that the user transaction data passed to it for TCAM has been properly formatted and can be placed directly on the TCAM output process queue. Except for the insertion of the destination identification, the CCB, and the data stream control characters, CICS does not alter the data it receives. It is your responsibility (using your MCP) to prepare the data for the destination terminal, for example, by translating and inserting line control characters.

Optionally, BMS can be used with TCAM to prepare the input data for the user task, and the output data for the specific terminal type. When BMS is required with TCAM, the TRMTYPE operand in DFHTCT TYPE=LINE or in DFHTCT TYPE=TERMINAL must indicate the specific terminal type for 3270 data streams. TRMTYPE=TCAM can be used to obtain EBCDIC data stream support. For BMS support within SNA, the TCAMFET=SNA and SESTYPE operands must also be specified in DFHTCT TYPE=LINE, and in DFHTCT TYPE=TERMINAL, respectively.

For information about the DFHTCT macros, see the *CICS/ESA Resource Definition Guide*.

Logic flow

The following is a generalized description of the sequence of events that occurs in CICS when interfacing with TCAM.

INPUT STEP	ACTION
A	TCAM notifies CICS that it has data for a particular input TCTLE, by posting its ECB.
B	CICS gets a TIOA and attaches it to the special input TCTTE in the TCTLE.
C	CICS issues a READ to TCAM that results in TCAM passing the data over the region boundaries to the CICS TIOA. CICS then indicates that it has data to process. (See Figure 74 on page 598.)
D	The input TCTLE points to the corresponding output TCTLE in response to the OUTQ specification of the DFHTCT TYPE=LINE macro.
E	The individual TCTTEs on the output TCTLE are searched for a matching source terminal netname. If POOL=YES has been specified, a free TCTTE is assigned to this source terminal identification. (See Figure 75 on page 599.)
F	If an input user exit (XTCTIN) has been specified, CICS links to the user exit routine where you may edit input data before passing it to a task (refer to page 155). If no exit has been specified, CICS removes the 8-byte source terminal identification field inserted by TCAM. For SNA devices, the input communication control byte (CCB) is removed. No other editing of the data is performed.
G	A check is made to determine whether a task is attached to the individual TCTTE. If there is no task attached, see Step H. If a task is attached, a check is made to see if the task has issued a READ. If a READ request exists, see Step J. If not, CICS halts the processing of data in the queue until the TCTTE is available or the attached task issues a READ.
H	CICS attaches the appropriate task. A user exit is available before the actual attach (refer to page 155). If the task could not be attached (for example, if a 'short on storage' or 'maximum tasks' condition exists), CICS records that it has data to process, and exits from DFHTCP.

the CICS–TCAM interface

- I After a task is attached, and if segment processing was specified by including the C parameter on the OPTCD operand of the DFHTCT TYPE=SDSCI macro, CICS stores the TCAM segment identifier in the TCTTE.
- J CICS passes control to the attached task.

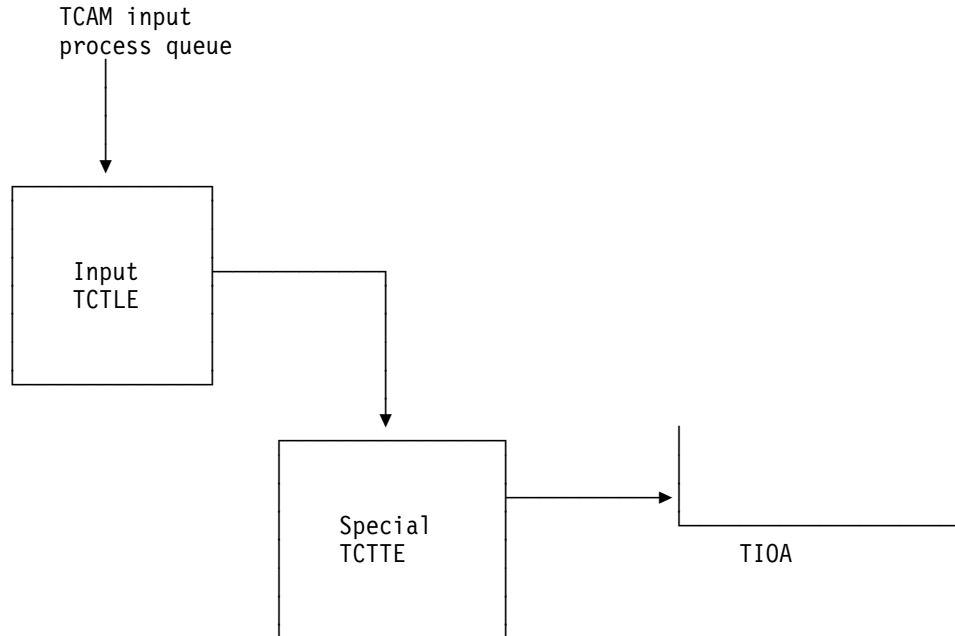


Figure 74. CICS issues a TCAM read

OUTPUT STEP ACTION

- A You issue an EXEC CICS SEND request in your application program.
- B The TCP terminal scan recognizes the SEND request.
- C CICS checks whether an output user exit (XTCTOUT) has been specified. If it has, CICS links to the user exit routine, where you may edit your output data before passing it to TCAM. (See the discussion of XTCTOUT under “TCAM user exits” on page 607.)
- D CICS checks the 4-byte TCTTE field TCTTEDES for a destination saved as a result of DEST(name) having been specified on the EXEC CICS SEND instruction. If it is present, CICS inserts it in the 8-byte destination field and left-aligns the field, padding blanks to the right. Otherwise, CICS moves the source terminal netname from the TCTTE to the destination field.
- E CICS moves the communication control byte (or bytes in TCAM SNA) into the 9th byte (9th and 10th bytes in TCAM SNA) of the TCAM work area. (See “TCAM devices” on page 604.)
- F CICS issues a TCAM WRITE to transfer the data to TCAM.
- G After checking for successful completion of the WRITE to TCAM, CICS flags the user task “dispatchable” if a task is still attached to the TCTTE. Otherwise, CICS frees the TCTTE for a new task.

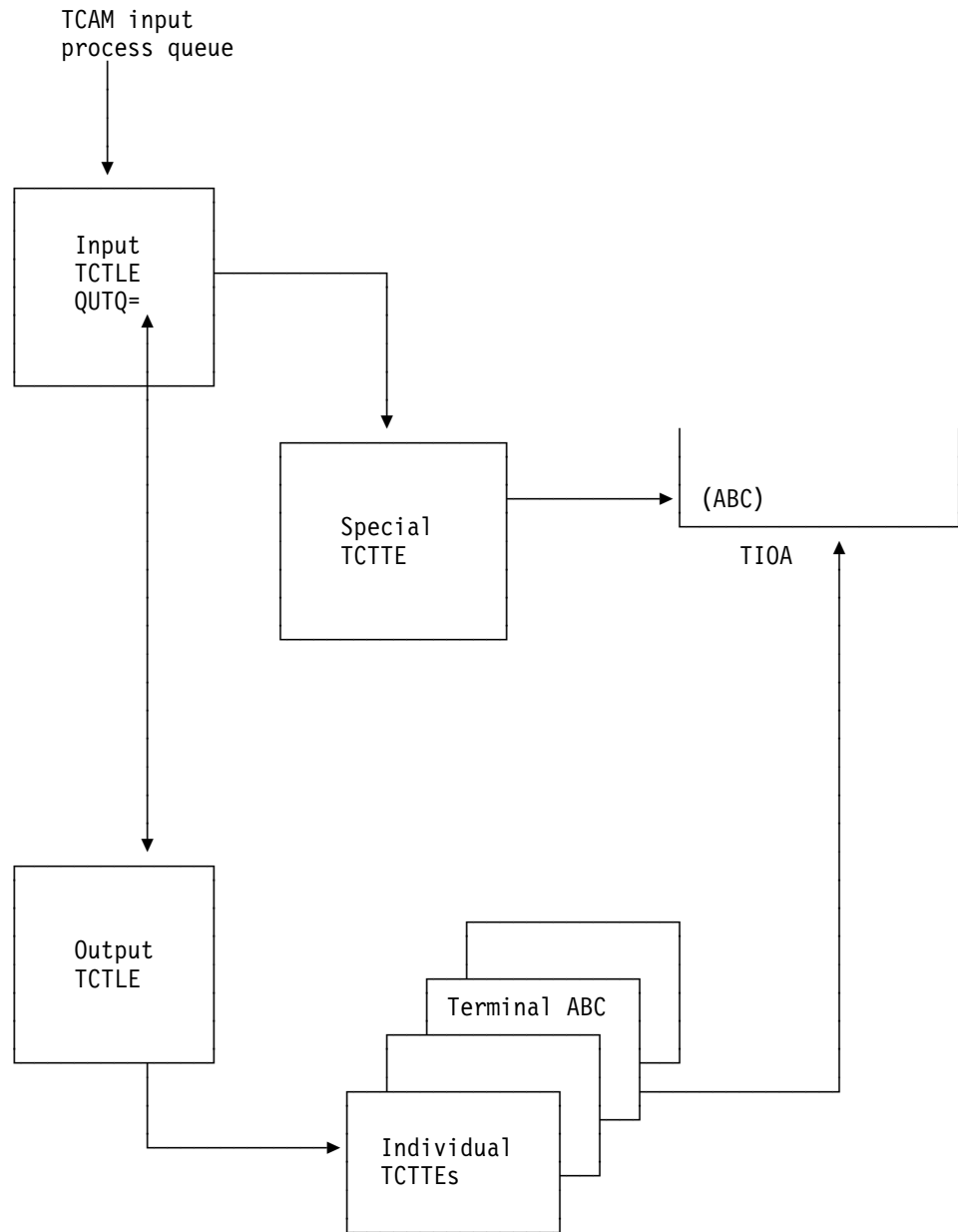


Figure 75. After a TCAM read, CICS attaches a TIOA to the corresponding TCTTE

Terminal error program

The CICS–TCAM interface implementation has resulted in the expansion of the CICS terminal error program (DFHTEP) error codes and conditions. The errors and actions shown in Table 124 are unique to TCAM, and should be considered in DFHTEP.

<i>Table 124. DFHTEP error codes and conditions unique to TCAM</i>		
Error code	Condition	Action
X'87'(TCEMCUI)	Unsolicited input Terminal 'receive only' Terminal 'out of service' Task has not issued a read No available TCTTE from pool.	a a, b No default No default
X'9F'(TCEMIDR)	TCAM has issued an invalid destination return code to CICS.	c

+

+

+

+

+

+

where:

a = Release TCAM TIOA (X'04' in TEPCAACT)

b = Terminal out of service (X'20' in TEPCAACT)

c = Abend transaction (X'10' in TEPCAACT).

Message routing

The DEST option of the EXEC CICS SEND command can be used to route an output message to a destination that you defined in the TCAM MCP. This operand can be used to send a message to a destination other than the source terminal (such as to another terminal, a list of terminals, or another application program).

CICS moves the data from TCTTEDES into the destination identification field before placing the data on the TCAM output process queue. You may bypass the CICS routine that inserts the destination field by taking the XTCMOUT user exit and returning to CICS from the exit with a displacement of zero. In this case, you must ensure that the TCAM header is correctly formatted for output.

If the DEST option is omitted, CICS inserts the source terminal NETNAME from the TCTTE into the destination identification field.

Segment processing

The CICS–TCAM interface supports TCAM segment processing, except when BMS is used. It permits segments of a message to be forwarded to CICS rather than waiting for the entire message to be received. If you specify segment processing (by including the parameter C on the OPTCD operand of the DFHTCT TYPE=SDSCI macro), CICS passes the segment to you, and places the position field control byte in the TCTTE field labeled TCTTETCM. Your application program can read the contents of TCTTETCM by means of an EXEC CICS INQUIRE TERMINAL command:

```
EXEC CICS INQUIRE TERMINAL(4-character data-value)
                    TCAMCONTROL(1-character data-area)
```

This returns a 1-byte hexadecimal value that indicates which segment of the message is being passed.

The possible values of TCTTETCM are:

Value	Message segment
X'00'	Null
X'40'	Intermediate portion of message
X'F1'	First portion of message
X'F2'	Last portion of message
X'F3'	Entire message
X'F4'	Intermediate portion of message, end of record
X'F5'	First portion of message, end of record
X'F6'	Last portion of message, end of record
X'F7'	Entire message, end of record
X'FE'	TCAM is not active
X'FF'	INQUIRE attempted on non-TCAM terminal.

On output, your application program must supply the control byte in TCTTETCM for CICS to pass to TCAM. The required command is:

```
EXEC CICS SET TERMINAL(4-character data-value)
          TCAMCONTROL(1-character data-value)
```

The possible values that can be passed are as listed above. For further information about EXEC CICS INQUIRE|SET TERMINAL commands, refer to the *CICS/ESA System Programming Reference* manual.

If multiple terminals have been defined for one output line (that is, multiple terminals related to one TPROCESS queue), you must ensure that an entire message is passed to TCAM for a specific destination before putting the first segment for another destination on the queue. In other words, an error is returned to you if a “PUT first segment to destination A” is followed by a “PUT first segment to destination B”. For additional information about segment processing, see the discussion of the OPTCD operand of the application input and output DCB in the *ACF/TCAM Version 3 Application Programming* manual.

Line pool specifications

When generating the TCAM message control program, you define each physical terminal and logical unit to TCAM by means of a TCAM TERMINAL macro. Because CICS also requires terminal definitions, you must prepare a terminal control table terminal entry (TCTTE) for each terminal, or logical unit, in a DFHTCT TYPE=TERMINAL macro. As a result, there is a one-to-one correlation between terminal definitions in TCAM and in CICS.

In a highly restricted environment, this duplication of terminal definitions can be reduced by using the POOL feature (DFHTCT TYPE=LINE,POOL=YES), and by specifying LASTTRM=POOL in DFHTCT TYPE=TERMINAL on the last TCTTE. Instead of a one-to-one relationship, a “pool” of generalized TCTTEs is defined for a TCAM process queue (line). When a transaction is received over the TCAM line, a search is made for an available TCTTE in the pool. When one is found, it is assigned a source terminal identification and netname for the duration of the task. When the task has been completed, the TCTTE can be reassigned. If there are no available TCTTEs to handle the next transaction from the line, the line remains locked until a TCTTE becomes available because the task has ended. The number of TCTTEs in the pool influences the degree of multitasking.

You should consider providing enough terminal entries in the pool to avoid an ‘unsolicited input’ condition (see “TCAM queues” on page 603), because there is no available entry in the pool. When DFHTCAM scans for a free entry, it does not scan the entire table because it carries a pseudo-end-of-table value. Therefore, unused entries at the end of the table can never be referred to.

Line pool restrictions

You must be aware of the following line pool restrictions:

- Because of device dependencies within CICS, only one terminal type is permitted for each TCAM line (process queue).
- Automatic transaction initiation (ATI), transaction routing, and BMS message routing are not applicable in the pool environment. If ATI is required for functions and you still want to use pooling, you should consider using a special queue of nonpooled entries suitable for ATI. For example, 3270 displays could be put on a pooled entry, and 3270 printers on a nonpooled queue to cause ATI to the printers.
- Statistics are accumulated for each TCTTE in the pool; however, the statistics cannot be correlated to the physical terminals or specific logical units.
- Only one signon can exist for all terminal entries in a given line pool at any one time. The first signon received by CICS is communicated to all terminals in the pool. Any subsequent signon is rejected. A sign-off clears the signon data from all terminal entries in the pool; a subsequent signon is then accepted.
- Terminal and line requests issued by the master terminal are not valid for pooled terminals.
- TCAM segment processing is not supported for pooled terminals.

Line locking

Two types of line locking can occur:

- A temporary lock that resolves itself in time
- A permanent lock that remains permanent unless you take action in the terminal error program.

A temporary line lock occurs when no TCTTEs are available in the pool and a new transaction appears on the input queue. CICS locks the queue until an existing task completes execution, thus freeing a TCTTE. In this case, the completion of existing tasks is not dependent upon additional input from the queue.

A permanent line lock can occur when multiple reads are required to complete a task. For example, suppose that there are two TCTTEs in the pool, that a task is attached to each, and that the messages in the input queue are in the following order:

1. Message 1 for a third transaction
2. Subsequent messages for the two active tasks.

Because no TCTTE is available in the pool for the third transaction, it must wait for a task to complete for a TCTTE to become available. Because the TCAM input queue is processed sequentially, the two active tasks are unable to receive their subsequent messages. Hence, they cannot complete, and the queue remains permanently locked.

TCAM queues

Because a queue is a sequential data set, the second message on the queue cannot be retrieved until the first message has been processed. To keep messages flowing smoothly through the queue, it is essential that each message be processed as soon as it arrives. In the CICS–TCAM interface, “processing the message” means detaching the message from the special input TCTTE and attaching it to the individual TCTTE correlated to the actual physical terminal or logical unit. Each individual TCTTE may be considered to be a “destination” for the purpose of this discussion.

If a particular destination (TCTTE) is not ready to accept the current message on the queue, the queue locks until the destination can accept the message. Queue locks are only a problem when a queue is serving more than one destination. In this case, if a queue locks, any new transaction on the queue, or messages queued for existing tasks, are not processed until the required destination has accepted the current message.

Because queue locks can adversely affect system performance, it is important that you understand their cause and effect, as described below. Proper configuration of the TCAM process queue and CICS terminal control tables reduces the occurrence and duration of queue locks to a minimum.

The maximum number of terminals that can be attached to one queue is governed by the amount of activity expected, and by the response time required from the system. For high activity and low response times, you should not exceed 25 terminals. Note that only a real performance test can verify whether this figure is acceptable.

Because TCAM can read ahead from the terminals, it is possible for TCAM to present to CICS a new transaction message destined for a TCTTE that is already processing a task. Also, TCAM can present a message for an existing task before that task issues a READ request. In either case, CICS cannot process the message (as described above) until the TCTTE is ready to accept the new TIOA. Such input is called “unsolicited input”.

These conditions can produce unsolicited input:

1. The CICS TCTTE for which the data is destined is out of service.
2. The CICS TCTTE for which the data is destined is in RECEIVE status.
3. The CICS TCTTE for which the data is destined has an associated task that has not issued a READ, and for which the period of time indicated by the NPDELAY specification has expired.
4. A terminal in a pool has entered data and is unable to find an available TCTTE.

In all cases, the action taken by the CICS–TCAM interface is to place the input line out of service, and attach DFHTACP to process the error condition.

The default action taken by DFHTACP (which can be altered by a user-written DFHTEP) for conditions 1 and 2 is to discard the data and place the input line in service. No default action is taken by DFHTACP for condition 3 or 4; therefore, the input line is placed in service, but with the same message still to be processed, thereby preventing CICS from reading any subsequent messages from the input queue.

To allow processing of input to continue, DFHTEP may take either of the following steps:

- If the input line is placed in service by DFHTEP, the CICS–TCAM interface retries the operation. In this case, a count mechanism is recommended in DFHTEP to prevent a loop occurring if the task never issues a READ, or a TCTTE never becomes available.
- Perhaps when a count limit is reached, DFHTEP might abend the task, dispose of the data, and place the line in service.

For further information about DFHTEP, see “Terminal error program” on page 362.

The problem of unsolicited input caused by condition 4 can be eliminated entirely by having a separate TCAM input process queue for each CICS terminal (TCTTE). However, as the number of terminals increases, this solution may quickly use up too much main storage.

You should analyze the type of traffic that you anticipate over the queues. If a 2770 Data Communication System, or a 2780 Data Transmission Terminal, is to read in volumes of cards, you should consider having separate queues for these devices. However, devices can share queues if traffic is conversational with short-lived tasks. The same TCAM output process queue can be specified for multiple input process queues. (See the discussion of the DFHTCT TYPE=LINE,OUTQ=symbolic name specification in the *CICS/ESA Resource Definition Guide*.)

You need not be concerned with locking the TCAM output process queue, because TCAM requeues the data according to its final destination once it arrives over the output queue.

It is possible for the TCAM output process queue to become congested because of lack of queuing space. In this case, CICS has a WRITE to the queue outstanding until TCAM accepts the data.

If the length of a message passed to a TCAM queue exceeds the queue's block size (specified in the DCB), DFHTCAM causes DFHTACP to be attached with 'output area exceeded' (error code X'8F').

TCAM devices

In a non-TCAM environment, the CICS terminal control program is responsible for polling and addressing terminals, code translation, transaction initiation, task and line synchronization, and the line control necessary to read from or write to a terminal. When TCAM is specified, terminal control relinquishes responsibility to the TCAM MCP for polling and addressing terminals, code translation, and line control. To take advantage of TCAM facilities, code, in the MCP message handler, functions, such as code translation, that were previously handled by the CICS terminal control program.

For some terminal services, it is necessary for CICS to pass the user request on to the TCAM MCP message handler. A communication control byte (2 bytes in TCAM SNA) in the TCAM work area has been established for this purpose. It is passed to TCAM along with the 8-byte destination name field. You must execute the

appropriate MCP message-handler macros according to the contents of the communication byte.

The terminal services parameters that do not set bits in the communication control byte are SEND, WAIT TERMINAL, and SAVE. Bits in the communication control byte are set for the ISSUE DISCONNECT and CONVERSE parameters, and for the FMH and LAST parameters on the EXEC CICS SEND command.

The CICS–TCAM interface does not support the RESET parameter or the 3270 parameters READB and COPY.

All messages to TCAM from CICS are prefixed with the standard CICS–TCAM communication area. This is one byte for the non-SNA TCAM interface, and two-bytes for the TCAM SNA interface (that is, when TCAMFET=SNA is specified in DFHTCT TYPE=LINE). This area is used to convey to TCAM special requests and options that cannot be used within CICS.

The format of the communication area is:

First byte

- X'01' FMH present in stream
- X'04' Extended CCB (2-byte CCB)
- X'08' DISCONNECT request
- X'10' READL (read keyboard)
- X'20' WRITEL (write keyboard).

Second byte (present if extended CCB is on)

- X'01' Last output from transaction (WRITE, LAST)
- X'02' READ requested after this WRITE (WRITE, READ request or CONVERSE).

All other flags are reserved and are set to 0.

Generalized TCAM message format

Messages passed between CICS and TCAM and *vice versa* have the format shown in Table 125.

<i>Table 125. General format of TCAM messages</i>				
Destination	CCB	Device-dependent data	FMH	Message
8 bytes	2 bytes (optional) (SNA only)	x bytes (device-dependent)	y bytes (SNA only)	

Destination Destination name (8 bytes) taken from TCTTE's netname parameter or from DEST specification on output.

TCAM devices

CCB	Communication control bytes. This determines the options specified for the message (for example, whether an FMH is present or not). The length of the CCB varies, and can be: 0 bytes (input message, non-SNA) 1 byte (output message, non-SNA) 2 bytes (input/output messages, SNA).
Device-dependent data	Dependent on the device: 3270, or other. See the following sections on the relevant devices.
FMH	Function management header. SNA only = length in first byte non-SNA = not applicable.
Message	User data.

TCAM with 3270 devices

The CCB and device-dependent data for an input message from TCAM to CICS have the following format:

CCB1 1 byte SNA only	CCB2 1 byte SNA only	Attention identifier 1 byte	CURS0R 2 bytes
----------------------------	----------------------------	-----------------------------------	-------------------

The CCB is present for TCAM SNA lines only.

The CCB and device-dependent data for 3270 output messages from CICS to TCAM have the following format:

CCB1 1 byte	CCB2 1 byte	1 1 byte	2 1 byte	3 1 byte
----------------	----------------	-------------	-------------	-------------

(2 bytes - SNA) device-dependent data
(1 byte - non-SNA)

- 1 Escape character
- 2 Command
- 3 WCC (write control character)

All SOH% status messages input to CICS are passed to DFHTACP or DFHTEP.

Terminal control copy and read buffer requests are not supported by the CICS–TCAM interface.

In addition to normal read and write functions, the ERASEAUP, CTLCHAR, and UCTRAN operands are also valid for the 3270.

All 3270 printer scheduling and error handling is provided by the TCAM message handler.

Note: For 3270 SDLC devices, the escape character must be removed by the message handler.

TCAM user exits

There are three TCAM global user exit points:

1. **XTCATT**, invoked before issuing a transaction manager ATTACH for a transaction identification received in response to polling
2. **XTCTIN**, invoked following the completion of any TCAM input event, (that is, after the individual TCTTE has been located, but just before CICS checks to see if a task is attached to the TCTTE)
3. **XTCTOUT**, invoked for output events before placing data on the TCAM output process queue.

For more information about global user exits, you should read Chapter 1, “Global user exit programs” on page 3.

Starting and terminating TCAM

CICS–TCAM startup

The TCAM MCP must be in operation before completing CICS system initialization. When you start up CICS with the CICS–TCAM interface, CICS checks for the presence of a TCAM region and issues the operator message:

```
DFHSI1513 - CICS CHECKING FOR TCAM MCP
```

If CICS discovers the MCP is not operational, the following messages are issued:

```
DFHSI1520 - TCAM MCP NOT CURRENTLY AVAILABLE
DFHSI1520 - REPLY 'RETRY' OR 'CANCEL' OR 'CONT'
```

The operator must then respond:

```
RETRY
```

when the TCAM region becomes active; or

```
CANCEL
```

to terminate CICS; or

```
CONTINUE
```

to continue initialization of CICS in the absence of the TCAM region.

If the operator responds CONTINUE, all DD cards that refer to a TCAM queue must have been previously removed from the startup deck to avoid an abnormal termination of CICS. The CONTINUE response is applicable to a mixed BTAM/TCAM mode of operation when TCAM lines are not being used during execution of CICS.

starting and terminating TCAM

CICS–TCAM abend and restart

If the TCAM message control program (MCP) terminates abnormally, any TCAM application programs currently active are automatically terminated abnormally, if there is at least one open line group in the MCP. This also applies to the CICS application program. For further information, see the relevant sections in the *ACF/TCAM System Programmer's Guide* and the *ACF/TCAM Version 3 Application Programming* manual. CICS does not provide RESTART capability.

CICS–TCAM termination

CICS is terminated in the normal manner. No modifications to termination procedures are required to support the CICS–TCAM interface. If both CICS and TCAM are being terminated, CICS should be terminated first, to avoid an abnormal termination of CICS.

CICS and TCAM: program interrelationship

Figure 76 illustrates the interrelationship between the TCAM message control program (MCP) and the TCAM application program. CICS is regarded as an application program by TCAM.

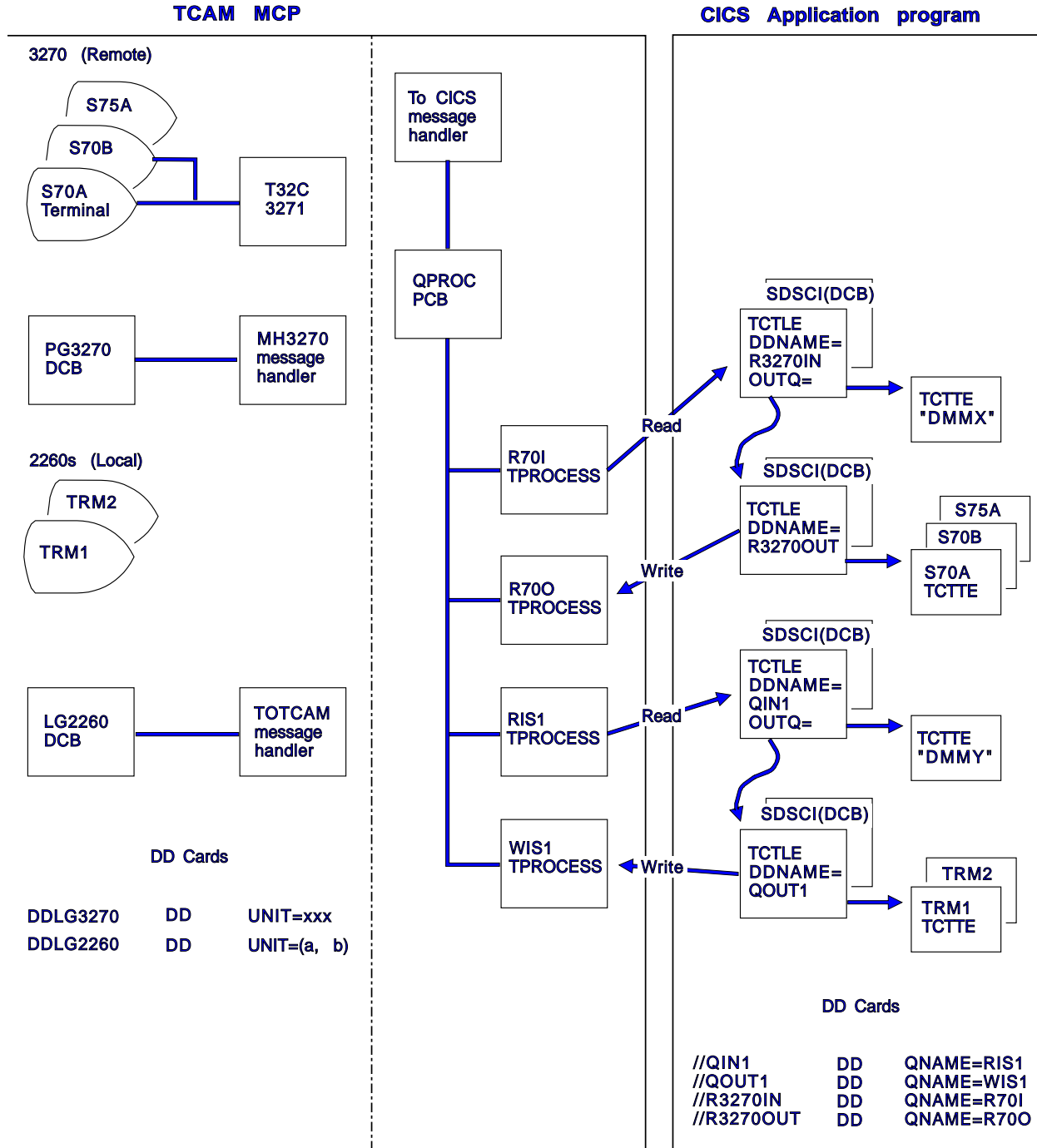


Figure 76. TCAM message control and application program

TCAM message control program (non-SNA)

This section gives an example of a non-SNA TCAM message control program. This MCP shows the relationship between the MCP definition and the CICS terminal control table generation. It should not be construed as a typical or usable MCP for CICS. For further information about MCP definition for a variety of devices, see the *ACF/TCAM Installation and Migration Guide*. An example of a CICS terminal control table for TCAM is given in the *CICS/ESA Resource Definition Guide*.

Note: Two sample TCAM message control programs for use in an SNA network are provided in CICS410.SDFHSAMP. These are DFHSPTM1 (in which control of SNA sessions is independent of the CICS application programs), and DFHSPTM2 (in which control of SNA sessions is according to the requirements of the CICS application programs).

```

MCPCICS    CSECT
TCAMINIT   INTRO   DISK=NO,                *
                                PROGID=TCAM/CICS,        *
                                LUNITS=40,                *
                                MSUNITS=20,               *
                                KEYLEN=132,               *
                                CROSSRF=4,                *
                                DLQ=TRM1,                 *
                                STARTUP=CY,               *
                                TRACE=10,                 *
                                LINETYP=BOTH,             *
                                OLTEST=0
                                LTR    15,15
                                BZ     OPENLINE
NOEXEC     ABEND    123,DUMP
OPENLINE   OPEN    (PG3270,(INOUT))
            TM      PG3270+48,DCBOFLGS
            BNO     NOEXEC
            WTO     'TIME TO START APPLICATION PROGRAM'
            READY
FINISH     CLOSE   (,PG3270)
            L       13,4(13)
            RETURN  (14,12)
PG3270    DCB      DSORG=TX,MACRV=(G,P),CPRI=S,DDNAME=DDPG3270, *
                                MH=MH3270,PCI=(N,N),BUFSIZE=464, *
                                INVLIST=(POLL70R,,),TRANS=EBCD
QPROC     PCB      MH=TOCICS, *
                                BUFSIZE=464, *
                                RESERVE=(20)
            TTABLE  LAST=TRM2
RIS1     TPROCESS PCB=QPROC,QUEUES=MO *
WIS1     TPROCESS PCB=QPROC
R70I     TPROCESS PCB=QPROC,QUEUES=MO
R700     TPROCESS PCB=QPROC
T32C     TERMINAL QBY=T,DCB=LG3270R,RLN=1,TERM=327C,QUEUES=MO
S70A     TERMINAL QBY=T,DCB=PG3270,RLN=1,TERM=327R,QUEUES=MO, *
                                ADDR=616140402D,NTBLKSZ=1856,SECTERM=YES
S70B     TERMINAL QBY=T,DCB=PG3270,RLN=1,TERM=327R,QUEUES=MO, *
                                ADDR=6161C1C12D,NTBLKSZ=1856,SECTERM=YES

```

Figure 77 (Part 1 of 2). Example of a TCAM message control program (non-SNA)

CICS-TCAM program interrelationship

```

S75A      TERMINAL QBY=T,DCB=PG3270,RLN=1,TERM=327R,QUEUES=MO,      *
          ADDR=E2E240402D,NTBLKSZ=1856
POLLST1   INVLIST ORDER=(TRM1+02)
POLLST2   INVLIST ORDER=(TRM2+02)
POLL70R   INVLIST ORDER=(T32C-C1C17F7F2D,S70A+C1C140402D,      *
          S70B+C1C1C1C12D,      *
          S75A+C2C27F7F2D,S75A+C2C240402D),      *
          EOT=37

*
TOTCAM    STARTMH LC=OUT      2260 MH
          INHDR      PROCESS INCOMING MSG HEADER
          CODE
          FORWARD DEST=C'RIS1'
          INMSG      PROCESS INCOMING COMPLETE MSG
          INEND      END OF INCOMING SECTION
          OUTHDR     PROCESS OUTGOING MSG HEADER
          OUTEND     END OF OUTGOING SECTION

*
MH3270    STARTMH LC=OUT,CONV=YES,STOP=YES 3270 MH
          INHDR,     PROCESS INCOMING MSG HEADER
          MSGTYPE X'6C' IS IT A STATUS MESSAGE
          B          SOH IF SO, BRANCH ROUND
          MSGTYPE    ALL OTHER MSGS TO HERE
          MSGEDIT ((R,CONTRACT,SCAN,(2))),BLANK=NO REMOVE CUDV BYTES
SOH       EQU * or DS 0H
          CODE
          FORWARD DEST=C'R70I'
          INBUF      INCOMING MSG SEGMENT SECTION
          INMSG      SECTION FOR COMPLETE MSGS
          INEND      END INCOMING SECTION
          OUTHDR
          SETSCAN 1
          MSGEDIT ((R,,SCAN)) REMOVE CCB
          MSGFORM    SETS IN LINE CONTROL CHARS
          CODE
          OUTBUF,
          OUTMSG
          OUTEND     END OUTPUT SECTION
          TOCICS    STARTMH LC=OUT      MH FOR APPLICATION
          INHDR
          CODE      TRANSLATE TO EBCDIC
          FORWARD DEST=PUT WRITE TO CICS
          INEND
          OUTHDR
          OUTEND
DCBOFLGS  EQU      X'10'
          END

```

Figure 77 (Part 2 of 2). Example of a TCAM message control program (non-SNA)

CICS–TCAM program interrelationship

Chapter 22. The dynamic allocation sample program

The dynamic allocation (DYNALLOC) sample application program makes available to the CICS/ESA 4.1 terminal operator most of the functions of DYNALLOC (SVC 99). These functions are described fully in the *MVS/ESA System Programming Library: Application Development Guide* Functions that require authorized program facility (APF) authorization are not supported.

The application consists of one command-level assembler-language program, DFH99, which is invoked by the transaction ADYN. The source code is provided in CICS410.SDFHSAMP.

Using DYNALLOC functions, the terminal operator can dynamically allocate or deallocate any data set that CICS can open and close. With suitable operating discipline and CEMT commands, these can include:

- Extrapartition transient data sets
- Journals
- Dump and trace data sets
- DL/I databases.

(Note that you can also use the IMS facility to allocate and deallocate DL/I databases dynamically.)

You should **not** use the dynamic allocation program to allocate and deallocate data sets that are to be associated with files managed by file control. You should use the dynamic allocation and deallocation facility which is part of CICS. If a file has not been allocated as part of CICS startup, CICS dynamic allocation occurs immediately before the file is opened, if sufficient information is in the file control table. The information needed is the data set name and disposition of the file. This information is set by the CEMT SET FILE master terminal transaction, described in the *CICS/ESA CICS-Supplied Transactions* manual, or the EXEC CICS INQUIRE and SET FILE commands, which provide additional inquiry and control facilities, and which are described in the *CICS/ESA System Programming Reference* manual.

To use the dynamic allocation sample program effectively, the terminal operator should:

- Have an understanding of MVS job control language, or TSO ALLOCATE and FREE commands.
- Have read the relevant sections of the *MVS/ESA System Programming Library: Application Development Guide* and have that manual available for reference while using the sample program, in particular for looking up error and reason codes returned by DYNALLOC.

The application uses a 3270 display screen terminal, and adjusts its formatting to suit the screen size. BMS is not required. The program is designed so that the installation can easily modify the functions supported to suit installation standards.

Installing the program and transaction definitions

Transaction and program definitions for the dynamic allocation sample program are provided in the sample utilities group DFH\$UTIL on the CSD. These definitions are installed using the CEDA command:

```
CEDA INSTALL GROUP(DFH$UTIL)
```

Notes:

1. DFH99 **must** be defined with EXECKEY(CICS).
2. If you make any changes to the sample program, you must run the DFH99BLD procedure before using the ADYN transaction.

Terminal operation

When transaction ADYN is entered at a terminal, the operator is presented with a formatted display. The top part of the display is for entering commands, the bottom part for receiving messages from the program.

The operator types a command in TSO-like syntax, for example,
verb {keyword[(value...)]}...

and presses the ENTER key. The program checks the command for correct syntax, builds a DYNALLOC parameter list, and, if no serious errors are detected, issues a DYNALLOC SVC. Messages are then displayed to diagnose syntax errors, give the DYNALLOC return codes, and show any values returned by DYNALLOC information retrieval features. The command remains on the display, and the editing features of the terminal can be used to correct it for reentry, or to enter a different command.

If there are too many messages to fit into the message area of the screen, messages that cannot be displayed are queued, and the messages already on the screen are displayed with a brighter intensity to indicate that there are more messages to come. The operator can correct those errors that are being displayed, and reenter the command for further checking, when the queued messages, if any, are regenerated.

The program is terminated by entering a null command, which consists of pressing the ERASE INPUT key, followed by the ENTER key. PA keys 1 and 2 are ignored by the program. If you press the CLEAR key, you redisplay the last command entered. Pressing a program function key is equivalent to pressing ENTER.

Help feature

The program includes a limited “help” feature, driven by the program’s keyword table.

In response to “?”, the verb keywords are displayed. In response to “verb?”, all the operand keywords of that verb are displayed. For “verb operand(?)” a short description of the value expected for that operand is displayed. When a command containing “?” is entered, no DYNALLOC SVC is issued. “?” is recognized only in the positions specified above; the rest of the command is ignored.

Values

Values are classified as follows:

Keyword value

Keyword values must be specified for some keywords. For example, the STATUS keyword may have a keyword value of SHR, NEW, MOD, or OLD (which can be abbreviated).

String of key letters

The value can be a string of letters in any order. The program does not check that the combination of letters provided is meaningful. For example, for the RECFM keyword, the value can be a string of letters from A, B, D, F, G, M, R, S, T, U, and V.

Returned values

No value should be provided by the terminal operator, because this keyword requests a value to be returned by the DYNALLOC information retrieval features. The further description refers to the kind of value that will be returned. This is usually in the form in which the operator would enter it, although in a few cases the value is as a hexadecimal string.

Not allowed

Some keywords do not require a value, and you must not provide one.

Required

A value must be provided if the keyword coded is designated as requiring a value.

Optional

Specification of a value is optional for some keywords.

Single

Only one value may be provided for some keywords.

Multiple

For some keywords, more than one value is permitted. (In some cases, DYNALLOC requires more than one value, although the dynamic allocation sample program does not enforce this.)

Character string

Any characters are permitted in this type of value, although in most cases there will be additional rules to follow, for example, for the DSNAME keyword.

Numeric string

Only numeric characters are allowed for this type of value, for example, for the EXPDT keyword.

Maximum and minimum lengths

For character and numeric values, the maximum and minimum lengths of the value are checked by the program. For a fixed-length string, these values are the same. The value is still passed to DYNALLOC as specified.

Convertible to n byte binary

A numeric value is required, of a magnitude representable in binary in the specified number of bytes. Values that are too large are truncated to the maximum possible for the width.

DYNALLOC flow of control

The dynamic allocation sample program does not support negative numbers. It does not cross-check operand keywords; errors of this type usually cause DYNALLOC to return error codes of the form 03xx.

Abbreviation rules for keywords

Keywords can be abbreviated. A word in the command matches a keyword if:

- The spelling is the same.
- The first letter is the same, and the remaining letters in the word appear in the same order as they do in the keyword.

If an ambiguity occurs, the program diagnoses the ambiguity, and lists the possible keywords.

System programming considerations

Keyword spellings are defined in the program's table, DFH99T, which is link-edited with the program. Where possible, these are the same as the corresponding job control or TSO keywords. Comments in the source code for DFH99T explain how the system programmer can:

- Change the spelling of keywords
- Define alternative spelling for keywords
- Divide the functions of a verb into subsets
- Add new verbs with subset function
- Add new operands as they become available in the SVC.

Member DFH99BLD in CICS410.SDFHINST is the job stream used to build the program. If part of the program has been modified, reassemble that part and link-edit the program again.

The macros IEFZB4D0 (DYNALLOC parameter list structure) and IEFZB4D2 (symbolic key equates), provided by MVS, are used in the dynamic allocation program and its keyword table. The meaning of each keyword in the table is defined in terms of a symbolic name, defined by one of the macros IEFZB4D0 or IEFZB4D2. These symbolic names are also documented in the *MVS/ESA System Programming Library: Application Development Guide*. The definitions of command keywords given in that manual should be regarded in preference to those from any other source. To obtain a list of command keywords and their symbolic values, for use as a cross-reference to the MVS manual, assemble DFH99T with option SYSPARM(LIST), and print the resulting object code. If the table is changed, repeat the assembly to obtain a new list.

The flow of control when a DYNALLOC request is issued

The flow in a normal invocation is as follows. The main program, DFH99M, receives control from CICS and carries out initialization. This includes determining the screen size, allocating input and output buffer sections, and issuing initial messages. It then invokes DFH99GI to get the input command from the terminal. Upon return, if the command was null, the main program terminates, issuing a final message.

The command obtained has its start and end addresses stored in the global communication area, COMM. The main program allocates storage for tokenized

text, and calls DFH99TK to tokenize the command. If errors were detected at this stage, further analysis of the command is bypassed.

Following successful tokenizing, the main program calls DFH99FP to analyze the verb keyword. DFH99FP calls DFH99LK to look up the verb keyword in the table, DFH99T. DFH99LK calls DFH99MT if an abbreviation is possible. Upon finding the matching verb, DFH99FP puts the address of the operand section of the table into COMM, and puts the function code into the DYNALLOC request block.

The main program now calls DFH99KO to process the operand keywords. Each keyword in turn is looked up in the table by calling DFH99LK, and the value coded for the keyword is checked against the attributes in the table. DFH99KO then starts off a text unit with the appropriate code and, depending on the attributes the value should have, calls a conversion routine.

For character and numeric strings, DFH99CC is called. It validates the string, and puts its length and value into the text unit.

For binary variables, DFH99BC is called. It validates the value, converts it to binary of the required length, and puts its length and value into the text unit.

For keyword values, DFH99KC is called. It looks up the value in the description part of the keyword table using DFH99LK, and puts the coded equivalent value and its length into the text unit.

When a keyword specifying a returned value is encountered, DFH99KO makes an entry on the returned value chain, which is anchored in COMM. This addresses the keyword entry in DFH99T, the text unit where the value is returned, and the next entry. In this case the conversion routine is still called, but it only reserves storage in the text unit, setting the length to the maximum and the value to zeros.

When all the operand keywords have been processed, DFH99KO returns to the main program, which calls DFH99DY to issue the DYNALLOC request.

DFH99DY sets up the remaining parts of the parameter list and, if no errors too severe have been detected, a subtask is attached to issue the DYNALLOC SVC. A WAIT EVENT is then issued against the subtask termination ECB. When the subtask ends and CICS dispatches the program again, the DYNALLOC return code is captured from the subtask ECB and the error and reason codes from the DYNALLOC request block and a message is issued to give these values to the terminal.

DFH99DY then returns to the main program, which calls DFH99RP to process returned values. DFH99RP scans the returned value chain, and for each element issues a message containing the keyword and the value found in the text unit. If a returned value corresponds to a keyword value, DFH99KR is called to look up the value in the description, and issue the message.

Processing of the command is now complete, and the main program is reinitialized for the next one, and loops back to the point where it calls DFH99GI.

Messages are issued at many places, using macros. The macro expansion ends with a call to DFH99MP, which ensures that a new line is started for each new message, and calls DFH99ML, the message editor. Input to the message editor is a list of tokens, and each one is picked up in turn and converted to displayable text.

DYNALLOC flow of control

For each piece of text, DFH99TX is called, which inserts the text into the output buffer, starting a new line if necessary. This ensures that a word is never split over two lines.

At the end of processing the command, the main program calls DFH99MP with no parameters, which causes it to send the output buffer to the terminal, and initialize it to empty.

Part 7. Customizing CICS security processing

General notes 620		Early verification processing 629
Chapter 23. Invoking a user-written external security manager 621	#	Chapter 24. Changes to link security for MRO 633
An overview of the CICS–ESM interface . . . 621		
The MVS router 621		Chapter 25. Writing a “good night” program 635
How ESM exit programs access CICS-related information 624		The sample “good night” program, DFH0GNIT 637
CICS security control points 627		

General notes

The following comments apply to the chapters in Part 7 of this book:

- The only aspects of CICS security processing covered are:
 - Invoking a user-written external security manager
 - Writing a security identification program for MRO connections
 - Writing a “good night” transaction.

For a comprehensive view of security processing using the Resource Access Control Facility (RACF) product, see the *CICS/ESA CICS-RACF Security Guide* .

- Upon return from any customer-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general-purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

Chapter 23. Invoking a user-written external security manager

CICS provides an interface to an external security manager (ESM), which may be user-written or may be the Resource Access Control Facility (RACF) program product. This chapter gives an overview of the CICS–ESM interface, and describes how you can use the MVS router exit to pass control to a user-written ESM. It describes how ESM exit programs can access CICS-related information. Finally, it lists the control points at which CICS invokes the ESM.

Note that this chapter is intended primarily for non-RACF users. For definitive information about security processing using RACF, you should refer to the *CICS/ESA CICS-RACF Security Guide*.

An overview of the CICS–ESM interface

CICS security uses, via the RACROUTE macro, the MVS system authorization facility (SAF) interface to route authorization requests to the ESM. Normally, if RACF is present, the MVS router passes control to it. However, you can modify the action of the MVS router by invoking the router exit. The router exit can be used, for example, to pass control to a user-written or vendor-supplied ESM. (If you want to use your own security manager, you **must** supply an MVS router exit routine.)

The control points at which CICS issues a RACROUTE macro to route authorization requests are described in “CICS security control points” on page 627.

The MVS router

SAF provides your installation with centralized control over security processing, by using a system service called the MVS router. The MVS router provides a common system interface for all products providing resource control. The resource-managing components and subsystems (such as CICS) call the MVS router as part of certain decision-making functions in their processing, such as access control checking and authorization-related checking. These functions are called **control points**. This single SAF interface encourages the use of common control functions shared across products and across systems.

If RACF is available in the system, the MVS router may pass control to the RACF router, which in turn invokes the appropriate RACF function. (The parameter information and the RACF router table, which associates router invocations with RACF functions, determine the appropriate function.) However, before calling the RACF router, the MVS router calls an optional, installation-supplied security-processing exit, if one has been installed.

The MVS router exit

The MVS router provides an optional installation exit that is invoked whether or not RACF is installed and active on the system. If your installation does not use RACF, you can use the router exit to pass control to your own ESM. If you do use RACF, you could use the exit for preprocessing before RACF is invoked.

The MVS router exit routine is invoked whenever CICS (or another component of your system) issues a RACROUTE macro. The router passes a parameter list (generated by the RACROUTE macro) to the exit routine. In addition, the exit receives the address of a 150-byte work area.

On entry to the exit routine, register 1 contains the address of the area described in Table 126.

Table 126. Area addressed by register 1, on entry to exit routine		
Offset	Length	Description
0	4	Parameter list address: points to the MVS router parameter list. (See "The MVS router parameter list.")
4	4	Work area address: points to a 150-byte work area that the exit can use.

The exit must be named ICHRTX00 and must be located in the link pack area (LPA).

+ **Note:** During signon processing, CICS/ESA 4.1 issues the RACROUTE
 + REQUEST=VERIFY macro with the ENVIR=VERIFY option, in
 + problem-program state. (For an explanation of why CICS does this, see
 + "Early verification processing" on page 629.) RACF requires RACROUTE
 + calls with the ACEE option to be issued in supervisor state. Therefore, if
 + you use an ICHRTX00 exit that intercepts CICS RACROUTE calls, and
 + replaces them with its own RACROUTE requests, your exit program should
 + not assume that a REQUEST=VERIFY call was made in supervisor state.

+ When intercepting a REQUEST=VERIFY call, your exit program should
 + check the settings of the two high-order bits of the byte at offset 3 in the
 + RACINIT parameter list. If ENVIR=VERIFY was specified on the call (as in
 + CICS early verification), these bits are both set on. If this is the case, your
 + exit program should not issue any further RACROUTE macros. To do so
 + could cause abends in RACF.

The MVS router parameter list

The MVS router parameter list is generated when the RACROUTE macro is issued, and describes the security processing request by providing the request type. If the router exit routine exists, the router passes the parameter list to this exit. (If it does not exist, and if RACF is active, the router passes the parameter list to the RACF router.)

You can map the MVS router parameter list using the ICHSAFP macro. Its format is shown in the *MVS/ESA Diagnosis: Data Areas* manual.

Router exit return codes

Your exit routine must return a return code in register 15. The hexadecimal values of the return code are shown in Table 127.

Code	Meaning
0	The exit has completed successfully. Control proceeds to the RACF front-end routine for further security processing and an invocation of RACF.
C8	The exit has completed successfully. The MVS router translates this return code to a router return code of '0' and returns control to the issuer of the RACROUTE macro (CICS), bypassing RACF processing. (See the next section.)
CC	The exit has completed successfully. The MVS router translates this return code to a router return code of '4' and returns control to CICS, bypassing RACF processing. (See the next section.)
D0	The exit has completed successfully. The MVS router translates this return code to a router return code of '8' and returns control to CICS, bypassing RACF processing. (See the next section.)
Other	If the exit routine sets any return code other than those described above, the MVS router returns control directly to CICS and passes the untranslated code as the router return code. Further RACF processing is bypassed.

Passing control to a user-supplied ESM

Normally, a caller (such as CICS) invokes the MVS router and passes it request type, requester, and subsystem parameters via the RACROUTE exit parameter list. Using these parameters, the MVS router calls the router exit which, on completing its processing, passes a return code to the router. If the return code is '0', as defined above, the router invokes RACF. RACF reports the result of that invocation to the router by entering return and reason codes in register 15 and register 0 respectively. The router converts the RACF return and reason codes to router return and reason codes and passes them to the caller. The router provides additional information to the caller by placing the unconverted RACF return and reason codes in the first and second words of the router input parameter list.

If your installation does not use RACF, you can make the MVS router exit pass control to an alternative ESM. However, if you do so you must still provide CICS with the RACF return and reason codes that it expects to receive. You set the router exit return code, as defined in Table 127, so that RACF is not invoked; and you **simulate** the results of a RACF invocation by coding the exit so that it places the RACF return and reason codes in the first and second fullwords of the router input parameter list. RACF return and reason codes are documented in the *MVS/ESA Authorized Assembler Programming Reference* manual.

Note: Remember that it is possible for a subsystem other than CICS to call the MVS router by issuing a RACROUTE macro. (Application programs too, may issue RACROUTE macros directly.) Your router exit program can establish whether the caller is CICS by checking the “eyecatcher” fields (UXPARROW, UXPDFHXS, and UXPKID) in the installation data parameter list—see “The installation data parameter list” on page 625.

How ESM exit programs access CICS-related information

When CICS invokes the ESM, it passes information about the current CICS environment, for use by an ESM exit program, in an **installation data parameter list**. How your exit programs access the installation data parameter list depends on whether or not your ESM is RACF.

For non-RACF users — the ESM parameter list

CICS (or another caller) passes information to your external security manager in the ESM parameter list, the address of which can be calculated using field SAFPRACP of the MVS router parameter list.

When the caller is CICS, the “INSTLN” field of the ESM parameter list points to the installation data parameter list, which contains CICS-related information that can be used by ESM exit programs.

The format of the ESM parameter list, and the actual name of the “INSTLN” field, vary, depending on which CICS security event is being processed. (The “request type” field (SAFPREQT) of the router parameter list shows why the ESM is being called by indicating the RACROUTE REQUEST type.) Table 128 shows how some formats of the ESM parameter list can be mapped using MVS macros.

RACROUTE REQUEST type	Parameter list mapping macro	INSTLN field name
VERIFY	IRRPRIPL	INITIPTR (X'10')
AUTH	ICHACHKL	ACHKIN31 (X'20')
FASTAUTH	Not available	Offset X'18'
LIST	Not available	Offset X'0C'
EXTRACT	Not available	None

Note: The INSTLN field points to the installation parameter list only if you specify INSTLN on the ESMEXITS system initialization parameter. The default value of this parameter is NOINSTLN, which means that no installation data is passed.

For RACF users — the RACF user exit parameter list

If you are a RACF user, you can find the address of the installation data parameter list directly from the RACF user exit parameter list. The name of the relevant field in the user exit parameter list varies according to the RACROUTE REQUEST type and the RACF user exit that is invoked. The relationships between REQUEST type, exit name, and field name are shown in Table 129 on page 625.

Table 129. Obtaining the address of the installation data parameter list

RACROUTE REQUEST type	RACF exit	Exit list mapping macro	Parameter list field name
VERIFY	ICHRIX01	ICHRIXP	RIXINSTL
	ICHRIX02	ICHRIXP	RIXINSTL
AUTH	ICHRCX01	ICHRCXP	RCXINSTL
	ICHRCX02	ICHRCXP	RCXINSTL
FASTAUTH	ICHRFX01	ICHRFXP	RFXANSTL
	ICHRFX02	ICHRFXP	RFXANSTL
LIST	ICHLX01	ICHLX1P	RLX1INST
	ICHLX02	ICHLX2P	RLX2PRPA See note 2.
EXTRACT	Not available	Not available	None

Notes:

1. The “xxxINSTL” field points to the installation parameter list only if you specify INSTLN on the ESMEXITS system initialization parameter. The default value of this parameter is NOINSTLN, which means that no installation data is passed.
2. RLX2PRPA contains the address of the ICHRLX01 user exit parameter list (RLX1P). Field RLX1INST of RLX1P in turn points to the installation data parameter list.

For full descriptions of the RACF exit parameter lists, see the *SPL: Resource Access Control Facility (RACF)* manual. For more information about CICS security processing using RACF, see the *CICS/ESA CICS-RACF Security Guide*.

The installation data parameter list

The installation data parameter list gives your ESM exit programs access to the following information:

- The CICS security event being processed.
- Details of the current CICS environment. That is:
 - The applid of the CICS region
 - The common work area (CWA)
 - The transaction being invoked
 - The program being executed
 - The CICS terminal identifier
 - The VTAM LU name
 - The terminal user area.

You can map the installation parameter list using the macro DFHXSUXP. The DSECT DFHXSUXP contains the following fields:

UXPLEN A halfword containing the length of this parameter list in bytes.

UXPARROW Arrow “eyecatcher” (>).

UXPDFHXS The name of the owning component (DFHXS).

ESM exit programs

UXPBLKID The name of the block identifier (UXPARMS).

UXPPHASE Address of a 1-byte code that indicates the reason for the call to the ESM (that is, the security event being processed). The code can have one of the following values:

	DEFAULT_SIGN_ON (X'01')	Signon of default userid
	PRESET_SIGN_ON (X'02')	Signon of preset security terminal
	IRC_SIGN_ON (X'03')	Link signon for IRC (MRO) links
	LU61_SIGN_ON (X'04')	Link signon for LUTYPE6.1 links
	LU62_SIGN_ON (X'05')	Link signon for APPC links
	XRF_SIGN_ON (X'06')	XRF tracking of signon
	ATTACH_SIGN_ON (X'07')	Attach-time signon of link user
#	NON_TERMINAL_SIGN_ON (X'08')	Signon of a non-terminal userid
#		
	USER_SIGN_ON (X'10')	Normal user signon
#	PRESET_SIGN_OFF (X'22')	Sign-off of a preset security terminal
#		
	LINK_SIGN_OFF (X'25')	Sign-off when link is closed
	XRF_SIGN_OFF (X'26')	XRF tracking of sign-off
	ATTACH_SIGN_OFF (X'27')	End-of-task sign-off of link user
#		
#	NON_TERMINAL_SIGN_OFF (X'28')	Sign-off of a non-terminal userid
	USER_SIGN_OFF (X'30')	Normal user sign-off
#	TIMEOUT_SIGN_OFF (X'31')	Sign-off forced by the terminal abnormal condition program, or time-out by the CSSC transaction
#		
#		
#		
#	USRDELAY_SIGN_OFF (X'32')	Sign-off forced by expiry of USRDELAY interval
#		
#		
	DEFERRED_SIGN_OFF (X'33')	Sign-off deferred to task end
	USER_ATTACH_CHECK (X'40')	Transaction attach check for user
	LINK_ATTACH_CHECK (X'41')	Transaction attach check for link
	EDF_ATTACH_CHECK (X'42')	Transaction attach check for CEDF
	USER_COMMAND_CHECK (X'50')	Command checking for user
	LINK_COMMAND_CHECK (X'51')	Command checking for link
#	EDF_COMMAND_CHECK (X'52')	Command checking for EDF
	USER_RESOURCE_CHECK (X'60')	Resource checking for user
	LINK_RESOURCE_CHECK (X'61')	Resource checking for link
#	EDF_RESOURCE_CHECK (X'62')	Resource checking for EDF
#	USER_SURROGATE_CHECK (X'68')	Surrogate checking for user
#	LINK_SURROGATE_CHECK (X'69')	Surrogate checking for link
#	EDF_SURROGATE_CHECK (X'6A')	Surrogate checking for EDF
	USER_QUERY_CHECK (X'70')	Query checking for user
	LINK_QUERY_CHECK (X'71')	Query checking for link
#	EDF_QUERY_CHECK (X'72')	Query checking for EDF
	INITIALIZE_SECURITY (X'80')	Initialization of CICS security

REBUILD_SECURITY (X'81')	CEMT or command-level SECURITY REBUILD
XRF_TRACK_INITIALIZE (X'82')	XRF tracking of initial or rebuild.

UXPSUBSY Address of an area containing the CICS subsystem identifier.

UXPAPPL Address of an area containing the CICS application ID.

UXPCWA Address of the Common Work Area.

UXPTRAN Address of an area containing the transaction identifier.

UXPPROG Address of an area containing the program name. The address may be zero if no program name can be identified.

UXPTERM Address of an area containing the terminal identifier. The address may be zero if no terminal is associated with the request.

UXPLUNAM Address of an area containing the VTAM LU name. The address may be zero if no terminal is associated with the request, or the area may be blank if the terminal is not a VTAM terminal.

UXPTCTUA Address of the TCT user area.

UXPTCTUL Address of a fullword containing the length of the TCTUA.

UXPCOMM Address of a 2-word communication area.

CICS security control points

The following list summarizes the RACROUTE macros used by CICS to invoke the ESM, and the control points at which they are issued.

RACROUTE

The “front end” to the macros described below, it invokes the MVS router. If RACF is not present on the system, RACROUTE can route to an alternative ESM, via the MVS router exit.

RACROUTE REQUEST=VERIFY

Issued at operator signon (with the parameter ENVIR=CREATE), and at sign-off (with the parameter ENVIR=DELETE). This macro creates or destroys an access control environment element (ACEE). It is issued at the following CICS control points:

Normal signon through EXEC CICS SIGNON

Signon of the default userid DFLTUSER

Signon of preset security terminals

Signon of MRO sessions

Signon of LUTYPE6.1 sessions

Signon of APPC sessions

Signon for XRF tracking of the above

Signon of the userid on attach requests (for all values of ATTACHSEC except LOCAL)

Normal sign-off through EXEC CICS SIGNOFF

Sign-off when deleting a terminal

Sign-off when TIMEOUT expires

Sign-off of MRO sessions

Sign-off of LUTYPE6.1 sessions

Sign-off of APPC sessions

Sign-off for XRF tracking of the above
Sign-off of the userid on attach requests (for all values of ATTACHSEC except LOCAL).

RACROUTE REQUEST=VERIFYX

This creates or deletes an ACEE in a single call. It is issued at the following CICS control points:

- Signon, as an alternative to VERIFY, when an optimized signon is performed for subsequent signons across an LU6.2 link with ATTACHSEC(VERIFY).
- When an invalid password, or a passticket is presented, or an EXEC CICS VERIFY PASSWORD command is issued.

RACROUTE REQUEST=FASTAUTH

Issued during resource checking, on behalf of a user who is identified by an ACEE. It is the high-performance form of REQUEST=AUTH, using in-storage resource profiles, and is issued at the following CICS control points:

When attaching local transactions
When checking link security for transaction attach
Transaction validation for MRO tasks
CICS resource checking
Link security check for a CICS resource
Transaction validation for EDF
Transaction validation for the transaction being tested (by EDF)
DBCTL PSB scheduling resource security check
DBCTL PSB scheduling link security check
Local DL/I PSB scheduling resource check
Remote DL/I PSB scheduling resource check
QUERY SECURITY with the RESTYPE option.

RACROUTE REQUEST=AUTH

This is a higher path length form of resource checking. It is used:

- After a call to FASTAUTH indicates an access failure that requires logging.
- When a QUERY SECURITY request with the RESCLASS option is used. This indicates a request for a resource for which CICS has not built in-storage profiles. (If CICS **has** in fact built in-storage profiles, REQUEST=AUTH uses them.)

RACROUTE REQUEST=LIST

Issued to create and delete the in-storage profile lists needed by REQUEST=FASTAUTH. (One REQUEST=LIST macro is required for each resource class.) It is issued at the following CICS control points:

When CICS security is being initialized
When an EXEC CICS REBUILD SECURITY is issued
When XRF tracks either of these events.

RACROUTE REQUEST=EXTRACT

Issued (with the parameters SEGMENT=SESSION,CLASS=APPCLU) during verification of APPC BIND security, at the following CICS control point:

BIND of APPC sessions.

It is also issued (with the parameters SEGMENT=CICS,CLASS=USER) during signon, at all the control points listed under RACROUTE REQUEST=VERIFY.

- + For a detailed description of these macros, see the *RACF External Security Interface (RACROUTE) Macro Reference for MVS and VM* manual.

Early verification processing

The CICS signon routine invokes the SAF interface, using the RACROUTE REQUEST=VERIFY macro with the ENVIR=VERIFY option in problem-program state. Invoking this version of the macro has no effect if the ESM is RACF, but other external security manager products can get control through the SAF exit interface, and perform their own **early verification** routine.

CICS defers the creation of the accessor environment element until the RACROUTE REQUEST=VERIFY macro with the ENVIR=CREATE option is issued to perform the **normal verification** routine. The ENVIR=CREATE version of the macro is issued by the security manager domain running in supervisor state.

CICS passes the following information on the ENVIR=VERIFY version of the RACROUTE REQUEST=VERIFY macro:

USERID

The userid of the user signing on to the CICS region.

GROUP

The group name, if specified, of the group into which the user wants to sign on.

PASSWRD

The user's password to verify the userid.

NEWPASS

A new value, if specified, for the user's password. This changes the existing password and is to be used for subsequent signons.

OIDCARD

The contents, if supplied, of an operator identification card.

APPL

The APPLID of the CICS region on which the user is signing on. Which APPLID is passed depends on what is specified as system initialization parameters.

INSTLN

A pointer to a vector of CICS-related information, which you can map using the DFHXSUXP mapping macro. This pointer is valid only if ESMEXITS=INSTLN is specified as a system initialization parameter for the CICS region.

The installation data referenced by the INSTLN parameter includes a pointer, UXPCOMM, to a two-word communications area that can be used to pass information between the two phases of the signon verification process—between the early verification routine initiated by ENVIR=VERIFY, and the normal verification routine initiated by ENVIR=CREATE.

CICS maintains a separate communications area for each task, in CICS-key storage.

Writing an early verification routine

An early verification routine, written for the ENVIR=VERIFY option, receives control from SAF in the usual way from the external security manager whose entry point is addressed by field SAFVRACR in the SAF vector table. It receives control in the same state as its caller, as follows:

- Problem-program state
- Task mode (usually the CICS quasi-reentrant TCB)
- PSW storage key 8
- 31-bit addressing mode
- Primary address translation mode.

Register 13 points to a standard 18-word save area. Register 1 points to a 2-word parameter list, where:

- The first word is the address of the SAF parameter list for the VERIFY function.
- The second word is the address of a 152-byte work area.

Using CICS API commands in an early verification routine

An early verification routine can use CICS application programming interface (API) commands, provided it obeys the following interface rules:

- The routine must be written in assembler.
- Entry to the routine must be via the DFHEIENT macro, which saves the caller's registers and establishes a CICS early verification API environment.
- Exit from the routine must be via the DFHEIRET macro, which releases the CICS early verification API environment and restores the caller's registers.
- The routine *must* be link-edited with the special security domain API stub, DFHXSEAI, *instead of* the normal CICS API stub, DFHEAI0. The CICS early verification stub causes linkage to a special interface routine that is aware of the SAF interface linkage requirements, and saves the current CICS command environment. In addition, the standard EXEC interface stub DFHEAI should also be included, immediately before the early verification routine, with an ORDER statement:

```
INCLUDE SYSLIB(DFHXSEAI)
INCLUDE SYSLIB(DFHEAI)
ORDER  DFHEAI,verify-program,DFHEAI0
ENTRY  verify-program
```

The DFHEIENT and DFHEIRET macros are inserted by the CICS translator unless you specify

```
*ASM XOPTS(NOPROLOG,NOEPILOG)
```

as the first statement of the program. The DFHEIENT macro assumes that register 15 points to its first executable instruction.

Upon return from the DFHEIENT macro, a CICS storage area mapped by the DFHEISTG macro has been established. The pointer DFHEIBP (and the register specified in the EIBREG parameter of DFHEIENT) contains the address of an EXEC interface block (EIB). DFHEICAP contains the pointer to the original parameter list supplied by the SAF interface.

Return and reason codes from the early verification routine

Before returning control, the early verification routine should set a return code and reason code in fields SAFPRRET and SAFPRREA of the SAF parameter list. It should also pass a value to be returned as the SAF return code in a register that is specified in the RCREG keyword of the DFHEIRET macro that is used to exit the program. These return codes are examined by the CICS signon function, and any non-zero value in SAFPRRET is interpreted as a verification failure and causes the signon to fail. A zero return code allows the signon to proceed, and eventually CICS issues a RACROUTE REQUEST=VERIFY,ENVIR=CREATE macro in supervisor state and under control of the CICS resource-owning TCB. It is only at this invocation that CICS accepts an ACEE address from the external security manager.

early verification processing

Chapter 24. Changes to link security for MRO

The user-replaceable security identification program, DFHACEE, which is used in
earlier releases to obtain an address space's userid, is withdrawn. In CICS/ESA
4.1, most CICS programs that need to know their caller's userid obtain this by
invoking the RACROUTE REQUEST=EXTRACT macro. For details of the
purposes for which CICS invokes this macro, see the *CICS/ESA CICS-RACF*
Security Guide. In those cases where CICS invokes the REQUEST=RACROUTE
macro, you can customize the userid by using the MVS security router exit,
ICHRTX00, to initialize the relevant ACEE fields.

Exceptions to the use of the RACROUTE REQUEST=RACROUTE macro are the
CICS interregion communication program (DFHIRP) and the external CICS
interface (EXCI) program request handler (DFHXCPRH). Because of performance
or authorization considerations, both DFHIRP and DFHXCPRH obtain their caller's
userid directly from the appropriate accessor environment element (ACEE). **Thus**
for MRO logon, bind, and link security signon functions, you cannot
“customize” the CICS region userid, as you can in earlier releases using
DFHACEE. This change affects those CICS regions running under an earlier
release that rely on DFHACEE to simulate “equivalent systems” in an MRO
environment. This inability to specify equivalent systems by customizing userids
affects only earlier releases of CICS that communicate through the CICS/ESA 4.1
DFHIRP. CICS/ESA 4.1 regions can specify the region userid parameter on the
SESSIONS resource definition to define equivalent systems; see the *CICS/ESA*
CICS-RACF Security Guide for more information about MRO security and
equivalent systems definition.

security identification for MRO

Chapter 25. Writing a “good night” program

You can use the GNTRAN system initialization parameter to specify a “good night” transaction that you want CICS to invoke when a user’s terminal-timeout period expires.

APAR PN89036

Documentation for APAR PN89036 added 30th January 1997.

The default value for GNTRAN is 'NO', which means that CICS does not schedule a “good night” transaction, but instead tries to sign off the terminal user. (Whether or not the sign off is successful depends on the value of the SIGNOFF attribute on the terminal’s TYPETERM definition.)

Notes:

1. Any transaction that you specify on the GNTRAN parameter must be able to handle the type of communication area it is passed when terminal timeout occurs. The CICS sign-off transaction, CESF, can do this, but CESN and all other CICS-supplied transactions cannot.

2. APAR PQ08546:

Documentation for APAR PQ08546 added 6th October 1997.

GNTRAN=NO must be specified if TCAM terminals are being used and users’ terminal-timeout values are specified in their RACF profiles.

For further information about GNTRAN, see the *CICS/ESA System Definition Guide*.

Writing your own “good night” program allows you to include functions in addition to, or instead of, sign-off. For example, your program could prompt the terminal user to enter their password, and allow the session to continue if the correct response is received. CICS supplies a sample “good night” program, DFH0GNIT, that demonstrates this, and a sample transaction definition, GNIT, that points to DFH0GNIT.

CICS passes the “good night” program a parameter list in the communications area shown in Figure 78 on page 636. If a terminal times out during a pseudoconversational transaction, your program could, using information in the parameter list:

- Ask for and check a response from the user
- Restore the screen left by the timed-out transaction
- Restore the cursor position
- Receive the communications area of the timed-out transaction, which is passed to the “good night” transaction as an input message
- Return with the TRANSID of the next transaction in the conversation.

	DFHSNGS			
	DFHSNGS_FIXED	DS	0CL64	Fixed part of parameter list
	GNTRAN_START_TRANSID	DS	CL4	TRANSID that invoked GNTRAN
	GNTRAN_PSEUDO_CONV_FLAG	DS	CL1	Pseudoconversational flag
	GNTRAN_SCREEN_TRUNCATED	DS	CL1	Screen buffer truncation flag
+	GNTRAN_TRANSLATE_TIOA	DS	CL1	Uppercase translation flag
		DS	CL10	Reserved
	GNTRAN_TIMEOUT_TIME	DS	CL8	Time of terminal timeout
	GNTRAN_TIMEOUT_REASON	DS	CL1	Reason for timeout
		DS	CL11	Reserved
	GNTRAN_PSEUDO_CONV_TRANSID	DS	CL4	Next transaction ID
	GNTRAN_SCREEN_LENGTH	DS	FL2	Length of screen buffer
	GNTRAN_CURSOR_POSITION	DS	FL2	Cursor position
	GNTRAN_SCREEN_WIDTH	DS	FL2	Width of screen
	GNTRAN_SCREEN_HEIGHT	DS	FL2	Height of screen
	GNTRAN_USER_FIELD	DS	CL16	Available to user program
	DFHSNGS_VARIABLE	DS	0X	Variable part of parameter list
	GNTRAN_SCREEN_BUFFER	DS	0X	Contents of screen buffer

Figure 78. Communications area passed to the "good night" program (assembler)

GNTRAN_START_TRANSID

The identifier of the transaction that started the "good night" transaction. If it was started by CICS because of a terminal timeout, GNTRAN_START_TRANSID is set to 'CEGN'. Your program should examine this field to check that timeout processing is appropriate (that is, that the "good night" transaction was started because of a terminal timeout and for no other reason).

GNTRAN_PSEUDO_CONV_FLAG

A flag indicating whether the terminal timed out during a pseudoconversational transaction.

- Y** The terminal timed out between transactions that form part of a pseudoconversational application.
- N** The terminal did not time out between transactions that form part of a pseudoconversational application.

GNTRAN_SCREEN_TRUNCATED

A flag indicating whether the 3270 screen buffer had to be truncated.

- Y** The screen buffer was truncated.
- N** The screen buffer was not truncated.

GNTRAN_TRANSLATE_TIOA

APAR PN 84487

Documentation for APAR PN 84487 added July 1996.

An internal flag indicating whether CICS is to translate the terminal input/output area (TIOA) to uppercase, if uppercase translation is specified on the TYPETERM or PROFILE definition.

- Y** The TIOA is to be translated to uppercase.
- N** Uppercase translation is to be bypassed.

GNTRAN_TIMEOUT_TIME

The time that the terminal timed out, in CICS ABSTIME format.

GNTRAN_TIMEOUT_REASON

The reason for the timeout:

- T** No input from the terminal
- X** An XRF takeover.

GNTRAN_PSEUDO_CONV_TRANSID

The identifier of the next transaction, if the terminal timed out during a pseudoconversational sequence. (If the terminal did *not* time out during a pseudoconversational sequence, the value of this field is meaningless.)

GNTRAN_SCREEN_LENGTH

The length of the screen buffer.

GNTRAN_CURSOR_POSITION

The cursor position.

GNTRAN_SCREEN_WIDTH

The width of the screen in use when the terminal timed out.

GNTRAN_SCREEN_HEIGHT

The height of the screen in use when the terminal timed out.

You can use GNTRAN_SCREEN_WIDTH and GNTRAN_SCREEN_HEIGHT to decide whether to use the ERASE DEFAULT or ERASE ALTERNATE option when restoring the user's screen.

GNTRAN_USER_FIELD

This field is available for use by your "good night" user program. It is initialized to binary zeroes and is not changed by CICS. You can use it to help develop a pseudoconversational "good night" transaction.

GNTRAN_SCREEN_BUFFER

A variable length field containing the contents of the screen buffer.

The sample "good night" program, DFH0GNIT

The sample "good night" program is a pseudoconversational COBOL program named DFH0GNIT. Copy books of the communications area passed to the "good night" program are supplied in assembler language, COBOL, PL/I, and C/370. The names of the supplied program, copy books, and mapset, and the CICS410 libraries in which they can be found, are summarized in Table 130 on page 638.

Table 130. Sample "good night" program, copy books, and mapset		
Language	Member name	Library
Program source: COBOL only	DFH0GNIT	SDFHSAMP
Copy books: Assembler COBOL PL/I C/370	 DFHSNGSD DFHSNGSO DFHSNGSL DFHSNGSH	 SDFHMAC SDFHCOB SDFHPL1 SDFHC370
Mapset:	DFH\$GMAP	SDFHSAMP

What the sample program does

The DFH0GNIT sample program:

1. Checks that it has been invoked for a terminal timeout, by testing the GNTRAN_START_TRANSID field of the communications area passed by CICS. If this contains anything other than 'CEGN', it quits.
2. If a flag within GNTRAN_USER_FIELD shows that this is the first invocation for this timeout:
 - a. If GNTRAN_PSEUDO_CONV_FLAG indicates that the terminal timed out during a pseudoconversation, issues EXEC CICS RECEIVE to retrieve the communications area.
 - b. Saves the length of the communications area in another field within GNTRAN_USER_FIELD.
 - c. Writes the communication area, if any, to a temporary storage queue.
 - d. Displays a screen asking the user to input his or her password, and sets the flag indicating that this has been done.
 - e. Issues EXEC CICS RETURN with TRANSID GNIT and the COMMAREA option, to continue the timeout process as a pseudoconversation.
3. If this is **not** the first invocation for this timeout:
 - a. Recovers the original communication area, if any, from the temporary storage queue.
 - b. Checks the password received from the user, and redisplay the timeout screen with an error message if it is incorrect.
4. If the number of incorrect responses exceeds the maximum specified to your external security manager, DFH0GNIT returns immediately with TRANSID CESF, which tries to sign off the userid.
5. If the correct password is entered, DFH0GNIT:
 - Restores the screen contents
 - Restores the cursor position.

If the terminal timed out during a pseudoconversational transaction, DFH0GNIT also:

- Restores the communications area of the timed-out transaction
- Returns with the TRANSID of the next transaction in the interrupted conversation.

Customizing the sample program

You can write your “good night” program in any of the languages supported by CICS, with full access to the CICS application and system programming interfaces.

If you customize the supplied program, or write your own “good night” program, note the following:

- Like the sample, your program should be pseudoconversational, because it could be invoked simultaneously for many users (if, for example, many terminals time out during the lunch period). If your program is conversational, CICS maximum number of tasks (MXT) could quickly be reached.

When you are continuing your timeout program’s pseudoconversation, always specify the name of your “good night” transaction (for example, GNIT) as the next TRANSID. If you do not, CICS does not know that you are still handling the timeout, and results may be unpredictable.

- Your program should always start, like the sample program, by testing the GNTRAN_START_TRANSID field of the communications area passed by CICS. If it finds that the “good night” transaction was started for any reason other than a terminal timeout (for example, by an EXEC CICS START request), timeout processing may not be appropriate.

- To obtain the communications area of the timed-out transaction in a pseudoconversation, your program must issue an EXEC CICS RECEIVE command. (The communication area passed to it on invocation is **not** that of the timed-out transaction, but contains information about the timed-out transaction.)

- If your program tries to sign off the terminal user, the result depends on what is specified on the SIGNOFF option of the terminal’s TYPETERM definition:

YES	The terminal is signed off, but not logged off.
NO	The terminal remains logged on and signed on.
LOGOFF	The terminal is both signed off and logged off.

- Specify the identifier (TRANSID) of your “good night” transaction on the GNTRAN system initialization parameter.

If you have customized the sample program, DFH0GNIT, specify the supplied sample transaction definition, GNIT.

If you have written your own “good night” program, named something other than DFH0GNIT, you must create and install a transaction definition that points to your program, and specify this definition on the GNTRAN SIT parameter.

sample good night program

Part 8. Examining and modifying resource attributes

A general note about user-written programs	642	DFHCSDUP as a batch program	644
		Invoking DFHCSDUP from a user program	653
Chapter 26. User programs for the system definition utility program (DFHCSDUP)	643	Chapter 27. The programmable interface to the RDO transaction, CEDA	663
An overview of DFHCSDUP	643	Using DFHEDAP in a DTP environment	664

A general note about user-written programs

The following comment applies to all user-written programs mentioned in Part 8 of this book:

- Upon return from any customer-written program, CICS must always receive control in primary-space translation mode, with the original contents of all access registers restored, and with all general purpose registers restored (except for those which provide return codes or linkage information).

For information about translation modes, refer to the *IBM ESA/370 Principles of Operation* manual.

Chapter 26. User programs for the system definition utility program (DFHCSDUP)

This chapter tells you how to write programs for use with the CICS system definition utility program (DFHCSDUP). It is divided into the following sections:

1. “**An overview of DFHCSDUP**” contains background information.
2. “**DFHCSDUP as a batch program**” on page 644 describes the DFHCSDUP EXTRACT command, and tells you how to write a user program to be invoked from DFHCSDUP.
3. “**Invoking DFHCSDUP from a user program**” on page 653 tells you how to write a program from which DFHCSDUP itself can be invoked.

An overview of DFHCSDUP

The CICS system definition utility program (DFHCSDUP) is a component of resource definition online (RDO). DFHCSDUP is an offline utility program that allows you to read from and write to a CICS system definition (CSD) file, either while CICS is running or while it is inactive.

Using DFHCSDUP, you can do the following:

- Add a group to the end of a named list in a CSD file
- Append a group list from one CSD file to a group list in another, or in the same, CSD file
- Copy all of the resource definitions in one group to another group in the same, or in a different, CSD file
- Define a single resource, or a group of resources, on the CSD
- Alter the definition of a single resource, on the CSD
- Delete from the CSD a single resource definition, all of the resource definitions in a group, or all of the group names in a list
- Extract requested data from the CSD and pass it to a named user program for processing
- Initialize a new CSD file, and add to it the CICS-supplied resource definitions
- List selected resource definitions, groups, and lists
- Migrate the contents of a table from a CICS load library to a CSD file
- Remove a single group from a list on the CSD file
- Apply service to a CSD file when necessary
- Upgrade the CICS-supplied resource definitions in a primary CSD file for a new release of CICS
- Verify a CSD file by removing internal locks on groups and lists.

DFHCSDUP as a batch program

You can invoke DFHCSDUP in two ways:

- As a batch program. The next section refers to this method.
- From a user program running either in batch mode or in a TSO environment. “Invoking DFHCSDUP from a user program” on page 653 describes this method.

DFHCSDUP as a part of the resource definition process is described in the *CICS/ESA Resource Definition Guide*. Guidance information about the execution JCL for DFHCSDUP, and the formats of the DFHCSDUP commands, are given in the *CICS/ESA Operations and Utilities Guide*.

DFHCSDUP as a batch program

This section refers to DFHCSDUP as a batch program. It describes the DFHCSDUP EXTRACT command, and the three sample programs that can be invoked during EXTRACT processing.

Writing a program to be invoked during EXTRACT processing

The DFHCSDUP LIST command produces reports about the current status of the CSD file that vary only according to the input parameters you provide. Another DFHCSDUP command, EXTRACT, causes the CSD data you select to be passed unformatted to a user program. The user program can then create reports of the CSD data that meet local requirements. For example, you could cross-refer related definitions (such as TERMINALs and TYPETERMs), or you could sort the data by attribute values, such as security keys or processing priorities. The user program could also write the requested resource attributes to a data set to be used as input to a database product, such as SQL, DB2, or the Data Extract program product.

The user program must be linked RMODE(24), AMODE(24). It receives control in 24-bit primary-space translation mode. (For information about translation modes, see the *IBM ESA/370 Principles of Operation* manual.) The contents of the access registers are unpredictable. The program must return control in 24-bit primary-space translation mode, and it must restore any access registers that it modifies (in addition to restoring the general purpose registers).

There are three sample programs that can be invoked from DFHCSDUP during EXTRACT processing. The sample programs, and how to replace them with your own versions, are described on page 647.

The EXTRACT command

The EXTRACT command takes requested data from the CSD and passes it to a user program for processing. The command has the following format:

```
EXTRACT {GROUP(name) | LIST(name)} USERPROGRAM(name) [OBJECTS]
```

GROUP

selects only those resource definitions within the named group. You can specify a generic group name, as on the DFHCSDUP LIST command.

LIST

selects only those resource definitions within the groups contained in the
named list. You can specify a generic list name only if you do not specify the
OBJECTS option.

OBJECTS

returns the detail of each resource definition. You can extract resource
definition data at two levels of detail:

- # 1. If you omit the OBJECTS option, the command extracts one of the
following:
 - # • The names of all the resource defintions within the specified group
 - # • The names of all the groups within the specified list.
- # 2. If you specify the OBJECTS option, all the attributes of the resource
definitions are also extracted.

USERPROGRAM

is the name of the user-written program that is to process the data retrieved by
the EXTRACT command. You must supply a USERPROGRAM value.

When the user program is invoked

The user program can be invoked at nine different points during the processing of
the EXTRACT command by DFHCSDUP. However, your program is invoked at all
of these points only if you specify both LIST and OBJECTS on the EXTRACT
command. The invocation points are as follows:

1. At the beginning of EXTRACT processing. This is to allow for activities such as
file opening and storage acquisition.
 2. At the beginning of LIST processing, but only if you have specified a LIST
value on the EXTRACT command.
 3. At the start of every group being processed by the EXTRACT command.
 4. At the start of each object (that is, resource type—TERMINAL, PROGRAM, and
so on) that is being processed, to allow for selection on an object or group
basis.
- # **Note:** If you have specified LIST but not OBJECTS on the EXTRACT
command, this invocation does not occur.
5. For every keyword (attribute) in the extracted object, but only if you have
specified OBJECTS on the EXTRACT command. This is to allow for the
detailed processing that may be necessary for cross-referencing.
 6. At the end of every object—that is, when all of the keywords within an object
have been processed. This is to allow for the processing of data built up from
the detailed items, and it occurs once for each object.
 7. At the end of every group, to allow for processing of the accumulated data.
 8. At the end of LIST processing, if you have specified a LIST value on the
EXTRACT command.
 9. When EXTRACT processing is complete, to allow for closing of files, release of
storage, and so on.

Parameters passed from DFHCSDUP to the user program

On every invocation of the user program, DFHCSDUP passes a parameter list addressed by general register 1. The parameter list consists of a series of fullwords that address the fields described in more detail below. The addresses set in the parameter list vary, depending on the point that EXTRACT processing has reached.

The parameter list contains the following fields:

Function Type Ptr

The address of a halfword field that contains a code defining the point in EXTRACT processing reached.

The function codes are as follows:

- 0 Initial call
- 2 List start call
- 4 Group start call
- 6 Object start call
- 8 Keyword detail call
- 10 Object end call
- 12 Group end call
- 14 List end call
- 16 Final call.

Workarea Ptr

This is the address of a field containing the address of a fullword to be used by the user application to store the address of any user-acquired work area.

Back translated command Ptr

The address of a fullword that contains the address of a 75-byte area of storage that contains the EXTRACT command that is being processed.

List name Ptr

The address of an 8-byte field that identifies the RDO list from which the current object is taken. This value is set only on the 'list start' and 'list end' calls.

Group name Ptr

The address of an 8-byte field that identifies the RDO group from which the current object is taken. This value is set on the 'group start', 'group end', 'object start', 'object end', and 'keyword' calls.

Object type Ptr

The address of a 12-byte field that identifies the type of object (such as TRANSACTION, PROGRAM, and so on), and is set only on the 'object start', 'object end', and 'keyword' calls.

Object name Ptr

The address of an 8-byte field that contains the name of the object, and is set only on the 'object start', 'object end', and 'keyword' calls.

Keyword name Ptr

The address of a 12-byte field that contains the name of the keyword being processed, and is set only on 'keyword' calls.

Keyword length Ptr

The address of a halfword field that contains the length of the value associated with the keyword, and is set only on 'keyword' calls.

Keyword Value Ptr

The address of the storage area that contains the value associated with the keyword, and is set only on 'keyword' calls.

Note: Fields not set with a pointer value contain a null value.

The sample EXTRACT programs

There are three CICS-supplied sample programs that can be invoked during DFHCSDUP EXTRACT processing. Two of these are provided in COBOL, PL/I, and assembler language, and the third is provided in VS COBOL II only. They are outlined in Table 131.

Program names	Languages	Description
DFH\$CRFA DFH0CRFC DFH\$CRFP	Assembler VS COBOL II PL/I	Produces a cross-reference listing of the resource definitions defined in the group or list you specify on the EXTRACT command.
DFH\$FORA DFH0FORC DFH\$FORP	Assembler VS COBOL II PL/I	Formats the group or list of resource definitions you specify on the EXTRACT command into a form suitable for the DB2 table load utility.
DFH0CBDC	VS COBOL II	Writes the list or group of resource definitions you specify on the EXTRACT command in the form of DEFINE commands, suitable for use as a backup copy of the resources extracted.

You can use the sample programs as supplied, or as models on which to base your own programs.

Only the assembler-language versions of DFH\$CRFx and DFH\$FORx are supplied in executable form in CICS410.SDFHLOAD, but for each of these programs the source statements are supplied in CICS410.SDFHSAMP.

The CICS-supplied sample DB2 formatting programs (DFH\$FORx) cannot be used when the CSD compatibility option (COMPAT) is specified on the DFHCSDUP utility program. The output from the CSD cross-reference listing and CSD backup utility programs depends on whether the compatibility option is specified. If the compatibility option is specified, the output includes obsolete keywords from releases before CICS/ESA 4.1; if the option is not specified, only keywords from CICS/ESA 4.1 are output. For further information about running the DFHCSDUP utility program with the compatibility option, see the *CICS/ESA System Definition Guide*.

Note that the sample programs require you to specify the OBJECTS keyword on the DFHCSDUP EXTRACT command.

DFHCSDUP as a batch program

The output data definition names (ddnames) for the sample programs are as follows:

CRFOUT CSD cross-referencing program
FOROUT DB2 formatting program
CBDOUT CSD backup utility program.

The sample programs are discussed in the next three sections.

The CSD cross-referencing program

The CICS-supplied sample CSD cross-referencing program produces a cross-reference listing of objects and keywords on the CSD. The data gathered by the EXTRACT command is passed to the sample program, where it is saved in a cross-reference table. On the final call to this sample program, the contents of the table are printed in collating sequence.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

For this program only, in addition to the EXTRACT command, you must define, in a sequential data set, the objects and keywords for which you want a cross-reference listing. The data set is read by the sample program using the ddname CRFINPT.

CRFINPT is a sequential file containing 80-byte records. Each record contains one object or keyword to be cross-referenced. You can cross-reference any valid resource type or attribute known to CEDA. For example, your CRFINPT file may contain the following entries (one per line):

```
PROGRAM  
TRANSACTION  
TYPETERM  
DSNAME
```

For each record in the file, a report is produced detailing the different values assigned to the keyword, where they are defined, and where they are used. Note that keyword values longer than 44 characters are truncated.

You should define the DCB subparameters for CRFINPT as DSORG=PS, RECFM=F, LRECL=80, and BLKSIZE=80.

The DB2 formatting program

The CICS-supplied sample DB2 formatting program organizes the CSD data passed to it from DFHCSDUP into a format suitable for the DB2 table load utility. The data is organized into columns that correspond to the columns defined in the load utility's input. Each selected resource causes a record to be written to this program's output file, with the first 4 characters identifying the resource type.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

The CSD backup utility program

The CICS-supplied sample CSD backup utility program produces a file of DFHCSDUP DEFINE control statements. The file can be used:

- For later editing and commenting to document CSD resources
- For distribution, in part or as a whole, to other CICS installations
- To recreate or add resource definitions to any CSD using DFHCSDUP.

The program must be run against an EXTRACT command of the form:

```
EXTRACT GROUP(group name) OBJECTS USERPROGRAM(program-name)
```

or:

```
EXTRACT LIST(list name) OBJECTS USERPROGRAM(program-name)
```

Note that the sample program requires you to specify the OBJECTS keyword.

Note the following points when using DFH0CBDC:

- It can deal with only one set of data during each invocation of DFHCSDUP; if two EXTRACT commands are issued, the second set of data overwrites the first.
- In the file produced by DFH0CBDC, any DEFINE statements that relate to CICS-supplied resources are preceded by an asterisk (*) in column 1; in other words, they are commented out. This is important if you use the file as input to define resources to a CSD. (The CICS-supplied definitions are already present in the CSD, having been produced automatically when it was initialized.)
- If you remove an asterisk from column 1 (to reinstate the DEFINE statement), do so by deleting it, **not** by overtyping it with a blank. This ensures that the resulting command is no more than 72 characters long; if it is longer than this, errors occur when the output is passed back through DFHCSDUP.

Assembling and link-editing EXTRACT programs

You must assemble (or compile) and link-edit DFHCSDUP user programs as batch programs, not as CICS applications, and you need link-edit control statements appropriate to the language in which they are written.

+

Note: DFHCSDUP user programs should not be translated, or unpredictable results could occur.

+

When you compile the COBOL versions of the sample programs, you must specify the compiler attributes NORENT and NORES.

DFHCSDUP as a batch program

When you link-edit the programs, you must specify the following link-edit control statements:

- An ENTRY statement that defines the entry name as DFHEXTRA
- An INCLUDE statement for a CICS-supplied stub that must be included in your user program
- A CHANGE statement to change the dummy CSECT name in the CICS-supplied stub from EXITEP to the name of your user program.

These requirements are explained in more detail for each of the three languages (assembler, VS COBOL II, and PL/I) shown in the following sample job streams.

An assembler-language version

The sample job in Figure 79 shows the link-edit statements you need for an assembler-language version of a DFHCSDUP user program.

```
//DFHCRFA JOB (accounting information),CLASS=A,MSGCLASS=A,NOTIFY=userid
//* .
//* Assembler job step here
//* .
//LINK EXEC PGM=IEWL,PARM='XREF,LIST,LET'
//OBJLIB DD DSN=object.module.library,DISP=SHR
//SYSLIB DD DSN=CICS410.SDFHLOAD,DISP=SHR
//SYSLMOD DD DSN=user.library,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//SYSLIN DD *
ENTRY DFHEXTRA
CHANGE EXITEP(csectname)
INCLUDE SYSLIB(DFHEXAI)
INCLUDE OBJLIB(obj-name)
NAME progame(R)
```

1
2
3
4
5

Figure 79. Link-edit control statements for a DFHCSDUP user program (assembler-language)

Notes for the assembler job:

- 1** Specify the entry name as DFHEXTRA, which is the entry name in the CICS-supplied stub, DFHEXAI. (See **3**.)
- 2** The CICS-supplied stub, DFHEXAI, is generated with a link to the user program using a dummy CSECT name (EXITEP). Use the link-edit CHANGE statement to change the CSECT name from EXITEP to the name of the CSECT in the user program. In the two CICS-supplied assembler-language sample programs, these names are:
CREFCSD The CSECT name in DFH\$CRFA, the cross-reference listing user program.
FORMCSD The CSECT name in DFH\$FORA, the DB2-formatting user program.
- 3** Include DFHEXAI in any assembler-language user program that you write for use with the DFHCSDUP EXTRACT command. DFHEXAI is the interface stub between DFHCULIS, a module in DFHCSDUP, and the user program.

4 obj-name is the name of the library member that contains the assembled object module.

5 progname is the name you want to call the load module; this is the name that you specify on the USERPROGRAM parameter of the EXTRACT command.

A VS COBOL II version

The sample job in Figure 80 shows the link-edit statements you need for a VS COBOL II version of a DFHCSDUP user program.

```
//DFHCRFC JOB (accounting information),CLASS=A,MSGCLASS=A,NOTIFY=userid
// .
// COBOL compile job step here
// .
//LINK EXEC PGM=IEWL,PARM='XREF,LIST,LET'
//SYSLIB DD DSN=SYS1.COBOL2.COB2LIB,DISP=SHR
//CICSLIB DD DSN=CICS410.SDFHLOAD,DISP=SHR
//OBJLIB DD DSN=object.module.library,DISP=SHR
//SYSLMOD DD DSN=user.library,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//COBLIB DD DSN=SYS1.COBOL2.COB2LIB,DISP=SHR
//SYSLIN DD *
ENTRY DFEXTRA
CHANGE EXITEP(prog-id)
INCLUDE CICSLIB(DFHEXCI)
INCLUDE SYSLIB(ILBOSRV)
INCLUDE SYSLIB(ILBOCMM)
INCLUDE SYSLIB(ILBOBEG)
INCLUDE OBJLIB(obj-prog)
NAME progname(R)
//
```

1
2
3
4
4
4
5
6

Figure 80. Link-edit control statements for a DFHCSDUP user program (COBOL)

Notes for the VS COBOL II job:

1 Specify the entry name as DFEXTRA, which is the entry name in the CICS-supplied stub, DFHEXCI. (See **3**.)

2 The CICS-supplied stub, DFHEXCI, is generated with a link to the user program using a dummy CSECT name (EXITEP). Use the link-edit CHANGE statement to change the CSECT name from EXITEP to the name specified on the PROGRAM-ID statement in the user program. In the three CICS-supplied COBOL sample programs, these names are:

CREFCSD The PROGRAM-ID in DFH0CRFC, the cross-reference listing user program.

FORMCSD The PROGRAM-ID in DFH0FORC, the DB2-formatting user program.

BDEFCS The PROGRAM-ID in DFH0CBDC, the CSD backup definitions user program.

3 Include DFHEXCI in any COBOL language user program that you write for use with the DFHCSDUP EXTRACT command. DFHEXCI is the interface stub between DFHCULIS, a module in DFHCSDUP, and the COBOL user program.

DFHCSDUP as a batch program

- 4** Specify the COBOL routines on the INCLUDE statements as shown.
- 5** obj-prog is the name of the object program.
- 6** progname is the name you want for the load module; this is the name that you specify on the USERPROGRAM parameter of the EXTRACT command.

A PL/I version

The sample job in Figure 81 shows the link-edit statements you need for a PL/I version of a DFHCSDUP user program.

```
//DFHCRFP JOB (accounting information),CLASS=A,MSGCLASS=A,NOTIFY=userid
//* .
//* PL/I compile job step here
//* .
//LINK EXEC PGM=IEWL,PARM='XREF,LIST,LET'
//SYSLIB DD DSN=object.module.library,DISP=SHR
// DD DSN=SYS1.PLI.PLIBASE,DISP=SHR
// DD DSN=CICS410.SDFHLOAD,DISP=SHR
//OBJLIB DD DSN=object.module.library,DISP=SHR
//SYSLMOD DD DSN=user.library,DISP=SHR
//SYSUT1 DD UNIT=SYSDA,SPACE=(1024,(100,10))
//SYSPRINT DD SYSOUT=A
//SYSLIN DD *
ENTRY DFHEXTRA 1
CHANGE EXITEP(CREFCSD) 2
INCLUDE SYSLIB(DFHEXPI) 3
INCLUDE SYSLIB(DFH$PDUM) 4
INCLUDE SYSLIB(DFH$CRFP)
INCLUDE OBJLIB(obj-prog) 5
NAME progname(R) 6
```

Figure 81. Link-edit control statements for a DFHCSDUP user program (PL/I)

Notes for the PL/I job:

- 1** Specify the entry name as DFHEXTRA, which is the entry name in the CICS-supplied stub, DFHEXPI (see **3**).
- 2** The CICS-supplied stub, DFHEXPI, is generated with a link to the user program using a dummy CSECT name (EXITEP). Use the link-edit CHANGE statement to change the CSECT name from EXITEP to the name specified on the PROC statement in the PL/I user program. In the two CICS-supplied PL/I sample programs, these names are:
CREFCSD The PROC name in DFH\$CRFP, the cross-reference listing user program.
FORMCSD The PROC name in DFH\$FORP, the DB2-formatting user program.
- 3** Include DFHEXPI in any PL/I language user program that you write for use with the DFHCSDUP EXTRACT command. DFHEXPI is the interface stub between DFHCULIS, a module in DFHCSDUP, and the PL/I user program.

4 Include the PL/I dummy program, DFH\$PDUM, to establish the PL/I environment, and enable the PL/I user program to run as a subroutine. Before you can include DFH\$PDUM, by specifying it on the INCLUDE statement as shown, you must compile it using your PL/I compiler. The compiled version of the dummy program must be included in one of the libraries available to the linkage editor (for example, SYS1.PLI.PLIBASE).

5 obj-prog is the name of the object program.

6 progname is the name you want for the load module; this is the name that you specify on the USERPROGRAM parameter of the EXTRACT command.

Invoking DFHCSDUP from a user program

It is possible to invoke DFHCSDUP from a user program. This method enables you to create a flexible interface to the utility. By specifying the appropriate entry parameters, your program can cause DFHCSDUP to pass control to an exit routine at any of five exit points. The exits can be used, for example, to pass commands to DFHCSDUP, or to respond to messages produced by DFHCSDUP processing.

You can run your user program:

- In batch mode
- Under TSO.

Notes:

1. In a TSO environment, it is normally possible for the terminal operator to interrupt processing at any time by means of an ATTENTION interrupt. In order to protect the integrity of the CSD file, DFHCSDUP does not respond to such an interrupt until after it has completed the processing associated with the current command. It then writes message number 'DFH5618' to the put-message exit (see "The put-message exit" on page 661), where this is available, and also to the default output file:

```
AN ATTENTION INTERRUPT WAS REQUESTED
DURING DFHCSDUP PROCESSING
```

Your put-message exit routine can terminate DFHCSDUP, if desired. (Note that you **must** supply a put-message routine if you want your operators to regain control after an ATTENTION interrupt.)

2. Suitably authorized TSO operators can use the CEDA INSTALL transaction to install resources that have previously been defined with DFHCSDUP. For information about the CEDA INSTALL command, see the *CICS/ESA Resource Definition Guide*.

Entry parameters for DFHCSDUP

When invoking DFHCSDUP, your program passes a parameter list addressed by register 1. It may pass up to five parameters, as described below:

OPTIONS

A list of character strings, separated by commas. (The information passed here is that which would otherwise be passed on the PARM keyword of the EXEC statement of JCL.) A maximum of four options can be specified:

CSD({READWRITE|READONLY}) specifies whether you require read-write or read-only access to the CSD.

PAGESIZE(nnnn) specifies the number of lines per page on output listings. Valid values for nnnn are 4 through 9999. The default value is 60.

NOCOMPAT|COMPAT specifies whether DFHCSDUP is to be invoked in compatibility mode. By default, it is invoked in noncompatibility mode. For details of compatibility mode, see the *CICS/ESA Resource Definition Guide*.

UPPERCASE specifies that output listings are to be printed entirely in uppercase characters. The default is to print in mixed case.

DDNAMES

A list of ddnames that, if specified, are substituted for those normally used by DFHCSDUP.

HDING

The starting page number of any listing produced by DFHCSDUP. You can use this parameter to ensure that subsequent invocations produce logically numbered listings. If this parameter is not specified, the starting page number is set to 1.

The length of the page number data (field 'bb' in Figure 82 on page 655) must be 0 or 4. The page number, if supplied, must be four numeric EBCDIC characters. The field, if present, is updated upon exit from DFHCSDUP with a number one greater than that of the last page printed.

DCBs

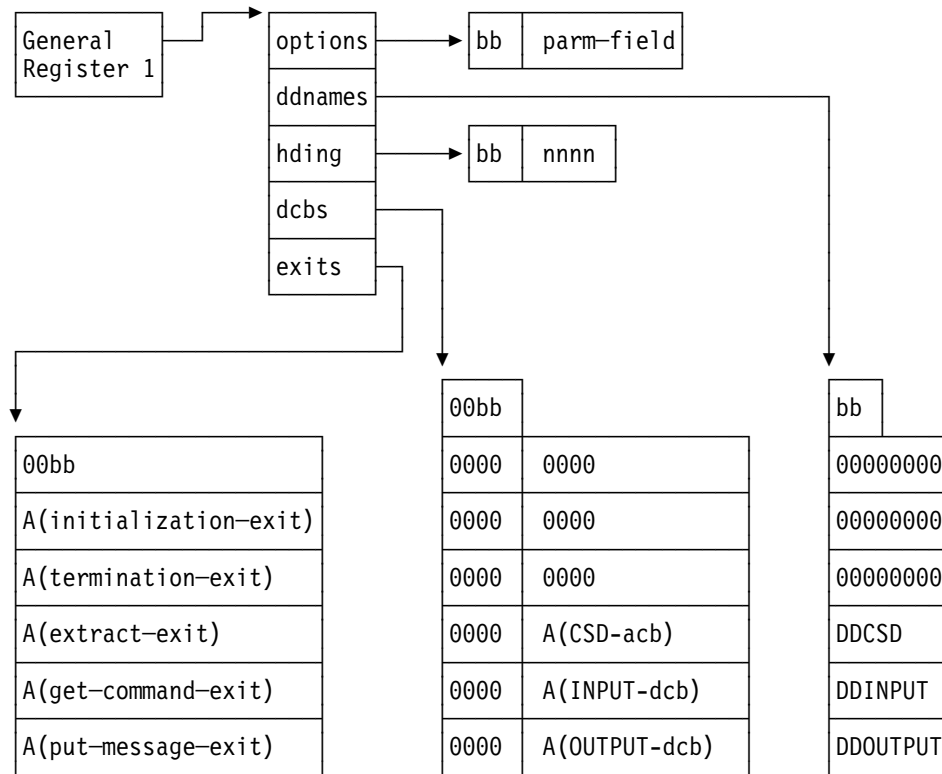
The addresses of a set of data control blocks for use internally by DFHCSDUP. Any DCBs (or ACBs) that you specify are used internally, instead of those normally used by DFHCSDUP.

Note that if you specify both replacement DDNAMES and replacement DCBs, the alternative DCBs are used, but the alternative DDNAMES are disregarded.

EXITS

The addresses of a set of user exit routines to be invoked during processing of DFHCSDUP.

The structure of the parameter list is shown in Figure 82 on page 655.



bb is a two-byte field containing the length of the functional data
 00 represents two bytes of binary zeros
 A() means "address of"

Figure 82. Entry parameters for DFHCSDUP

You should note the following:

- Each parameter contains a length field, followed by some functional data.
- The functional data for the DDNAMES, DCBS, and EXITS parameters contains multiple subentries.
- The parameters OPTIONS, DDNAMES, and HDING are aligned on a halfword boundary, and the first two bytes 'bb' contain the binary number of **bytes** in the following functional data.
- The parameters DCBS and EXITS are aligned on a fullword boundary, and the first four bytes '00bb' contain the binary number of **fullwords** in the following functional data.
- If the 'bb' field for any parameter is zero, the parameter is ignored.
- If a subentry in the functional data is all binary zeros, it is ignored.
- If any subentry is not within the length indicated by 'bb', it is ignored.
- In the DDNAMES functional data, each subentry consists of an 8-byte ddname to replace a default ddname used by DFHCSDUP. DFHCSDUP does not use the first three subentries of the DDNAMES parameter. The fourth, fifth, and sixth subentries, if present, replace the ddnames of DFHCSD, SYSIN, and SYSPRINT, respectively.

invoking DFHCSDUP from a user program

- In the DCBS functional data, each subentry consists of two fullwords. The first word is not used by CICS. The second word contains the address of an open DCB or ACB. You must ensure that the DCB or ACB has been opened with the correct attributes, which are:

PRIMARY CSD AM=VSAM,MACRF=(KEY,DIR,SEQ,IN,OUT)

INPUT FILE DSORG=PS,MACRF=GL,LRECL=80,RECFM=FB

The address of any end-of-data routine (EODAD) or I/O error routine (SYNAD) in the DCB is overlaid by DFHCSDUP.

OUTPUT FILE DSORG=PS,MACRF=PL,LRECL=125,RECFM=VBA

DFHCSDUP does not use the first three subentries of the DCBS parameter. The fourth, fifth, and sixth subentries, if present, are used instead of the internal DCBs or ACBs for DFHCSD, SYSIN, and SYSPRINT, respectively.

- In the EXITS parameter, each subentry consists of a single fullword containing the address of an exit routine. You must specify the exit routines in the order shown in Figure 82 on page 655. (The user exits are described in “The user exit points in DFHCSDUP.”)

Responsibilities of the user program

Before invoking DFHCSDUP, your calling program must ensure that:

- AMODE(24) and RMODE(24) are in force
- System/370 register conventions are obeyed
- If the EXITS parameter is passed, any programming environment needed by the exit routines has been initialized
- Any ACBs or DCBs passed for use by DFHCSDUP are OPEN.

The user exit points in DFHCSDUP

There are five user exit points in DFHCSDUP. By specifying the appropriate entry parameters, you can cause DFHCSDUP to pass control to an exit routine at any of these points.

None of the user exits supports XPI calls.

Parameters passed to the user exit routines

The address of a parameter list is passed to the user exit routine in register 1. The list contains some standard parameters that are passed to all of the exit routines, and may also contain some exit-specific parameters that are unique to the exit point from which the exit routine is being invoked.

The format of the parameter list is identical to that used by CICS global user exits. For a description of the standard parameters, see “DFHUEPAR standard parameters” on page 8. Explanations of the exit-specific parameters are included in the descriptions of the individual exits, which follow.

The initialization exit

The initialization exit is invoked once during DFHCSDUP initialization. Its purpose is to allow a routine to perform exit-related initialization. For example, the routine may obtain its own global work area and save its address in UEPGAA and its length in the halfword pointed to by UEPGAL. These values are retained by DFHCSDUP and become available at the other exit points.

Initialization exit	
	Invoked once, on entry to DFHCSDUP.
Exit-specific parameters	None
Return codes	UERCNORM (X'00') Continue processing UERCERR Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.
XPI calls	Must not be used.

The get-command exit

The purpose of the get-command exit is to read in command lines. If it is specified, no commands are read from SYSIN.

On invocation, your exit routine must supply the address and length of a **complete** command. It must return control with either the normal return code 'UERCNORM' or with the code 'UERCDONE', signifying that it has no more commands to pass. After it has processed each command, DFHCSDUP reinvokes the exit until return code 'UERCDONE' is received.

Get-command exit		
		Invoked multiple times, at the point where DFHCSDUP would otherwise read commands from SYSIN.
Exit-specific parameters	UEPCMDA	Address of a fullword containing a pointer to a command
	UEPCMDL	Address of a halfword containing the length of the command text. The maximum length that can be specified is 1536 bytes.
Return codes	UERCNORM (X'00')	Continue processing
	UERCDONE (X'04')	No more commands to process. (This is equivalent to reaching end-of-file on the SYSIN file.)
	UERCERR	Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.
XPI calls		Must not be used.

The extract exit

The extract exit is invoked at various points during processing of the EXTRACT command. The points are listed on page 645.

Notes:

1. If you do not specify an EXTRACT user exit routine on the entry linkage to DFHCSDUP, or on the USERPROGRAM keyword, a syntax error occurs.
2. A user exit routine specified on the USERPROGRAM keyword is used in preference to one specified on the entry linkage.

Extract exit

	Invoked multiple times during processing of the EXTRACT command.
Exit-specific parameters	<p>EXTRACT_FUNCTION_CODE_PTR Address of a halfword containing a code that defines the point in EXTRACT processing reached. The EXTRACT function codes are listed on page 646.</p> <p>EXTRACT_WORK_AREA_PTR Address of a fullword containing the address of the EXTRACT work area.</p> <p>EXTRACT_BACKTRAN_COMMAND_PTR Address of a fullword containing the address of the EXTRACT command being processed.</p> <p>EXTRACT_CSD_LIST_NAME_PTR Address of an 8-byte field containing the name of the list whose data is being extracted. This value is set only on 'list start' and 'list end' calls.</p> <p>EXTRACT_CSD_GROUP_NAME_PTR Address of an 8-byte field containing the name of the group whose data is being extracted. This value is set on 'group start', 'group end', 'object start', 'object end', and 'keyword' calls.</p> <p>EXTRACT_CSD_OBJECT_TYPE_PTR Address of a 12-byte field that identifies the type of object (such as TRANSACTION, PROGRAM, and so on). This value is set only on 'object start', 'object end', and 'keyword' calls.</p>

Extract exit (continued)	
EXTRACT_CSD_OBJECT_NAME_PTR	Address of an 8-byte field containing the name of the object. This value is set only on 'object start', 'object end', and 'keyword' calls.
EXTRACT_KEYWORD_NAME_PTR	Address of an 12-byte field containing the name of the keyword being processed. This value is set on 'keyword' calls only.
EXTRACT_KEYWORD_LENGTH_PTR	Address of a halfword containing the length of the value associated with the keyword. This value is set on 'keyword' calls only.
EXTRACT_KEYWORD_VALUE_PTR	Address of a character string which contains the value associated with the keyword. This value is set on 'keyword' calls only.
<p>Note that these parameters are similar to those passed when DFHCSDUP is invoked as a batch program. (See "Parameters passed from DFHCSDUP to the user program" on page 646.) However, when DFHCSDUP is invoked from a user program, the parameter list also includes the standard parameters mentioned under "Parameters passed to the user exit routines" on page 656.</p>	
Return codes	<p>UERCNORM (X'00') Continue processing</p> <p>UERCERR Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.</p>
XPI calls	Must not be used.

The put-message exit

The put-message exit is invoked whenever a message is to be issued. If you are running under TSO, you could use this exit to terminate DFHCSDUP after the operator inputs an ATTENTION interrupt. (See “Invoking DFHCSDUP from a user program” on page 653.) Or you could use it to provide messages in the operator’s national language.

Even if this exit is supplied, messages are always additionally written to the default output file (that is, to SYSPRINT, or to the replacement ddname specified on the entry linkage to DFHCSDUP).

Put-message exit																																																					
Invoked when a message is to be issued.																																																					
Exit-specific parameters	<p>UEPNUM Address of a 4-character field containing the message number</p> <p>UEPMDOM Reserved</p> <p>UEPINSN Address of a 2-byte field containing the number of insert fields</p> <p>UEPINS A Address of the following message structure:</p> <table border="0"> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> <tr> <td>INS_1_TEXT_PTR</td> <td>DS</td> <td>A</td> <td>Address of insert 1</td> </tr> <tr> <td>INS_1_LEN_PTR</td> <td>DS</td> <td>A</td> <td>Address of a fullword containing the length of insert 1</td> </tr> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> <tr> <td>INS_2_TEXT_PTR</td> <td>DS</td> <td>A</td> <td>Address of insert 2</td> </tr> <tr> <td>INS_2_LEN_PTR</td> <td>DS</td> <td>A</td> <td>Address of a fullword containing the length of insert 2</td> </tr> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> <tr> <td></td> <td>...</td> <td></td> <td></td> </tr> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> <tr> <td>INS_n_TEXT_PTR</td> <td>DS</td> <td>A</td> <td>Address of insert n</td> </tr> <tr> <td>INS_n_LEN_PTR</td> <td>DS</td> <td>A</td> <td>Address of a fullword containing the length of insert n</td> </tr> <tr> <td></td> <td>DS</td> <td>F</td> <td>Reserved</td> </tr> </table> <p>The exit-specific parameters provide a message number and insert fields only, to enable you to provide messages in the language of your TSO operators. The structure pointed to by UEPINS A is repeated as many times as UEPINS N requires.</p>		DS	F	Reserved	INS_1_TEXT_PTR	DS	A	Address of insert 1	INS_1_LEN_PTR	DS	A	Address of a fullword containing the length of insert 1		DS	F	Reserved		DS	F	Reserved	INS_2_TEXT_PTR	DS	A	Address of insert 2	INS_2_LEN_PTR	DS	A	Address of a fullword containing the length of insert 2		DS	F	Reserved		...				DS	F	Reserved	INS_n_TEXT_PTR	DS	A	Address of insert n	INS_n_LEN_PTR	DS	A	Address of a fullword containing the length of insert n		DS	F	Reserved
	DS	F	Reserved																																																		
INS_1_TEXT_PTR	DS	A	Address of insert 1																																																		
INS_1_LEN_PTR	DS	A	Address of a fullword containing the length of insert 1																																																		
	DS	F	Reserved																																																		
	DS	F	Reserved																																																		
INS_2_TEXT_PTR	DS	A	Address of insert 2																																																		
INS_2_LEN_PTR	DS	A	Address of a fullword containing the length of insert 2																																																		
	DS	F	Reserved																																																		
	...																																																				
	DS	F	Reserved																																																		
INS_n_TEXT_PTR	DS	A	Address of insert n																																																		
INS_n_LEN_PTR	DS	A	Address of a fullword containing the length of insert n																																																		
	DS	F	Reserved																																																		
Return codes	<p>UERCNORM (X'00') Continue processing</p> <p>UERCERR Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.</p>																																																				
XPI calls	Must not be used.																																																				

The termination exit

The purpose of the termination exit is to allow you to perform final housekeeping duties. It is invoked before a normal or an abnormal termination of DFHCSDUP.

Termination exit		
Invoked once, before termination of DFHCSDUP.		
Exit-specific parameters	UEPTRMFL	A 1-byte field that indicates the mode of termination. Its possible values are: 0 Normal termination 1 Abnormal termination.
Return codes	UERCNORM (X'00')	Continue processing
	UERCERR	Irrecoverable error. This causes DFHCSDUP to terminate with a return code of '8'.
XPI calls	Must not be used.	

The sample program, DFH\$CUS1

The CICS-supplied sample program, DFH\$CUS1, illustrates how DFHCSDUP can be invoked from a user program. It is written as a command processor (CP) for execution under the TSO/E operating system.

Note that DFH\$CUS1 uses different DCB and ACB names from those normally used by DFHCSDUP. Ensure that these are allocated before running the program under TSO/E.

Chapter 27. The programmable interface to the RDO transaction, CEDA

This chapter describes a programmable interface to the resource definition online (RDO) transaction, CEDA. The functions provided by RDO can be invoked from application programs, by a command such as:

```
EXEC CICS LINK PROGRAM('DFHEDAP')
          COMMAREA(CEDAPARM)
```

where DFHEDAP is the name of the entry point in the RDO program, and CEDAPARM is a user-defined name of a parameter list consisting of five 31-bit addresses (each contained in a fullword) as follows:

1. Address of a field containing the RDO command in source form.
2. Address of a halfword binary field specifying the length of the command. The maximum length of the input command is 1022 bytes.
3. Address of a 1-byte indicator field defined as follows:

X'80'	Display output at terminal instead of returning it to caller.
X'00'	Do not display output at terminal.
4. Address of a field in which output is to be placed by DFHEDAP.
5. Address of a halfword binary field specifying the maximum length of output that the application can handle.

If the indicator in address 3 is X'80', output is displayed at the terminal. In this case, you can enter any number of CEDA commands at the terminal, in response to the output displayed on your screen. Control is returned to your application program when you press PF3.

However, if the indicator is X'00' (output is not to be displayed at the terminal), DFHEDAP returns control to your application program immediately after processing the RDO command specified in the first address. At the same time, DFHEDAP returns the output as one or two concatenated, structured fields. The output from a single request comprises one field for the translation stage and one or none for the execution stage. Each field has the format:

- Binary halfword containing inclusive length of field.
- Binary halfword containing the number of messages produced.
- Binary halfword containing the highest message-severity: '0' and '4' continue to execution; '8' and '12' do not continue to execution.
- Variable-length data containing:
 - For the translation stage: diagnostic messages if there are any.
 - For the execution stage: data that would normally appear on the CEDA screen, including messages. Each line begins with a new line (NL) character and, otherwise, consists of blanks and uppercase alphanumeric characters.

The format of this data is not guaranteed from release to release, but it is the same as that displayed by CEDA. (Analysis of this data should not normally be necessary. Typically, your program is interested only in whether or not the

command was successful.) If the total output is longer than the maximum length specified by the user, it is truncated.

Notes:

1. An attempt to start CEDA from an application program by an EXEC CICS START command must fail. This is because CEDA's first action is to request input from its associated terminal, whereas an automatically initiated transaction must first send data to the terminal.

An attempt to start CEDA under CECI by an EXEC CICS START command fails for similar reasons.

2. The RDO command passed in address 1 of the CEDAPARM parameter list must be valid. (For example, spelling errors such as PRORGAM for PROGRAM are not corrected automatically when you use the programmable interface.)

- +
+
+
+
3. You cannot use the programmable interface to change the values of CEDA keywords that are obsolete in this release of CICS, but which are retained for compatibility with earlier releases. That is, the interface does not support *compatibility mode*.

- +
+
+
+
4. CEDA issues various syncpoints as part of its processing. Therefore, when your program links to DFHEDAP the current logical unit of work (LUW) of the transaction is completed. This may result in problems if, for example, there are outstanding browse operations against VSAM datasets.

Use of the programmable interface

Remember that you can use the offline utility program, DFHCSDUP, to examine and amend CSD files; and that DFHCSDUP can be invoked from a user program, running either in batch mode or under TSO. (See Chapter 26, "User programs for the system definition utility program (DFHCSDUP)" on page 643.)

Using DFHCSDUP is the recommended method for updating CSD files in bulk.

You should only use the interface described in this chapter where the required function includes the INSTALL command, which is not available from DFHCSDUP.

Using DFHEDAP in a DTP environment

The LINK DFHEDAP function is intended to be used in a single environment. It is not supported within a distributed transaction programming (DTP) environment—using it in such an environment can result in abends.

In a DTP environment, CICS may attempt to propagate SYNCPOINT and SYNCPOINT ROLLBACK requests across sessions to other systems. These requests are issued by CEDA modules that are invoked by the use of LINK DFHEDAP. Note that the issuing of SYNCPOINT ROLLBACK means that LINK DFHEDAP cannot be used in a DTP environment that owns LU6.1 links.

Generally, a session should be in SEND state to initiate a SYNCPOINT, but the session may not remain in SEND state once a LINK DFHEDAP command is issued. (For information about valid commands and states, see the *CICS/ESA Distributed Transaction Programming Guide*. This book also explains the APPC

| architecture rules on a session's state after SYNCPOINT and SYNCPOINT
| ROLLBACK requests are made.)

| The code invoked by LINK DFHEDAP can result in wrong sequence of commands.
| For example, if the code invoked by DFHEDAP issues a SYNCPOINT ROLLBACK
| from a back-end application program whose session is in SEND state (and which
| has never issued a SYNCPOINT), the session will be put into RECEIVE state. If
| the code invoked by DFHEDAP then issues a SYNCPOINT, an abend occurs. This
| can be prevented by all DTP applications issuing a SYNCPOINT request when they
| get into SEND state (on all of their sessions) and before they issue the LINK
| DFHEDAP command.

| Do not attempt to use LINK DFHEDAP when more than a pair of DTP application
| programs are involved—that is, one front end and one back end.

| The general rules for using LINK DFHEDAP within a simple DTP environment (one
| front end and one back end) are that all sessions in a DTP environment should be
| in SEND state when the LINK DFHEDAP command is issued, and they should
| revert to SEND state in the event of a SYNCPOINT ROLLBACK being issued by
| the DFHEDAP code.

Part 9. Appendixes

<p>Appendix A. Coding entries in the VTAM LOGON mode table 669</p> <p>TYPETERM device types and pointers to related LOGON mode data 671</p> <p>VTAM MODEENT macro operands 674</p> <p>PSERVIC screen size values for LUTYPE0, LUTYPE2, and LUTYPE3 devices 679</p> <p>Matching models and LOGON mode entries 680</p> <p>LOGON mode definitions for CICS-supplied autoinstall models 692</p>	<p>Default actions for system sense codes 709</p> <p>Action flag settings and meanings 711</p>
<p>Appendix B. Default actions of the node abnormal condition program 697</p> <p>Default actions for terminal error codes 697</p>	<p>Appendix C. Transient data write-to-terminal program (DFH\$TDWT) 713</p> <p>Appendix D. Uppercase translation of national characters 715</p> <p>Using the XZCIN exit 715</p> <p>Using DFHTCTDY 715</p>
	<p>Appendix E. The example program for the XTSEREQ global user exit, DFH\$XTSE 717</p>

Appendix A. Coding entries in the VTAM LOGON mode table

This appendix shows you what to code in your VTAM LOGON mode table for a terminal to be automatically installed.

CICS uses the logmode data when processing an automatic installation (autoinstall) request. Automatic installation functions properly only if the logmode entries that you define to VTAM have matches among the TYPETERMs and model TERMINAL definitions that you specify to CICS. "Matching models and LOGON mode entries" on page 680 and "LOGON mode definitions for CICS-supplied autoinstall models" on page 692 show examples of matching definitions.

The following tables show, for a variety of possible terminal devices, what you must code on the VTAM MODEENT macros that define your logmode table if you want to use autoinstall. Between them they show the values that must be specified for each of the operands of the MODEENT macro. Where all bit settings of an operand's value have significance for CICS, the data is shown in hexadecimal form. If some of an operand's bit settings are not significant to CICS, its data bytes are shown as bit patterns. The bit settings that have significance for CICS are shown set to the values that CICS expects. Those bits that have no significance to CICS are shown as periods. Thus, for example:

```
01..0011
```

shows that six bits in the subject byte must be given specific values; the remaining two have no significance.

Some of the examples shown here correspond exactly to entries in the CICS-supplied LOGON mode table called ISTINCLM. Where this is so, the table gives the name of the entry in ISTINCLM.

The PSERVIC setting shows fields called aaaaaaaaa, bbbbbbbb, and so on. The contents of these vary for LUTYPE0, LUTYPE2, and LUTYPE3 devices, according to how you specify certain attributes of the terminals. You can work out the values you need by looking at "PSERVIC screen size values for LUTYPE0, LUTYPE2, and LUTYPE3 devices" on page 679.

TYPETERM device types and pointers to related LOGON mode data

Search Table 132 for the TYPETERM device type that corresponds to the terminal you want to autoinstall. When you find the right one, use the number to its right to locate, in Table 133 on page 674, what has to be coded on the VTAM MODEENT macros.

Note that Table 132 is a complete list of TYPETERM device types; not all of these can be used with autoinstall. Those that cannot are marked with an asterisk (*). For details about coding TYPETERM definitions, and for a list of terminals that can be autoinstalled, see the *CICS/ESA Resource Definition Guide*.

<i>Table 132 (Page 1 of 3). TYPETERM device types, with cross-references to VTAM logmode entries</i>	
TYPETERM device type	Reference number in Table 133 on page 674
DEVICE(APPC)	24
DEVICE(BCHLU)	17
DEVICE(BCHLU) SESSIONTYPE(BATCHDI)	15
DEVICE(BCHLU) SESSIONTYPE(USERPROG)	16
DEVICE(CONTLU)	10
DEVICE(INTLU)	11
DEVICE(LUTYPE2)	18
DEVICE(LUTYPE2) TERMMODEL(1)	18
DEVICE(LUTYPE3)	19
DEVICE(LUTYPE3) TERMMODEL(1)	19
DEVICE(LUTYPE4)	12
DEVICE(SCSPRINT)	11, 13
DEVICE(TLX)	8
DEVICE(TLX) SESSIONTYPE(CONTLU)	8
DEVICE(TLX) SESSIONTYPE(INTLU)	9
DEVICE(TWX)	8
DEVICE(TWX) SESSIONTYPE(CONTLU)	8
DEVICE(TWX) SESSIONTYPE(INTLU)	9
DEVICE(3270)	2
DEVICE(3270) BRACKET(NO)	1
DEVICE(3270) TERMMODEL(1)	2
DEVICE(3270) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3270P)	2
DEVICE(3270P) BRACKET(NO)	1
DEVICE(3270P) TERMMODEL(1)	2
DEVICE(3270P) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3275)	2

TYPETERM device types

<i>Table 132 (Page 2 of 3). TYPETERM device types, with cross-references to VTAM logmode entries</i>	
TYPETERM device type	Reference number in Table 133 on page 674
DEVICE(3275) BRACKET(NO)	1
DEVICE(3275) TERMMODEL(1)	2
DEVICE(3275) TERMMODEL(1) BRACKET(NO)	1
DEVICE(3600)	16, 22, 23
DEVICE(3600) SESSIONTYPE(PIPELINE) *	21
DEVICE(3600) SESSIONTYPE(PIPELN) *	21
DEVICE(3614) *	3
DEVICE(3650) SESSIONTYPE(PIPELINE) *	21
DEVICE(3650) SESSIONTYPE(PIPELN) *	21
DEVICE(3650) SESSIONTYPE(USERPROG) BRACKET(YES)	6
DEVICE(3650) SESSIONTYPE(USERPROG) BRACKET(NO)	7
DEVICE(3650) SESSIONTYPE(3270)	5
DEVICE(3650) SESSIONTYPE(3270) BRACKET(NO)	4
DEVICE(3650) SESSIONTYPE(3653)	5
DEVICE(3650) SESSIONTYPE(3653) BRACKET(NO)	4
DEVICE(3767)	11
DEVICE(3767C)	10
DEVICE(3767I)	11
DEVICE(3770)	17
DEVICE(3770) SESSIONTYPE(BATCHDI)	15
DEVICE(3770) SESSIONTYPE(USERPROG)	16
DEVICE(3770B)	17
DEVICE(3770B) SESSIONTYPE(BATCHDI)	15
DEVICE(3770B) SESSIONTYPE(USERPROG)	16
DEVICE(3770C)	10
DEVICE(3770I)	11
DEVICE(3790)	20
DEVICE(3790) SESSIONTYPE(BATCHDI)	14
DEVICE(3790) SESSIONTYPE(SCSPRT)	13
DEVICE(3790) SESSIONTYPE(SCSPRINT)	13
DEVICE(3790) SESSIONTYPE(USERPROG)	16
DEVICE(3790) SESSIONTYPE(3277CM)	18

<i>Table 132 (Page 3 of 3). TYPETERM device types, with cross-references to VTAM logmode entries</i>	
TYPETERM device type	Reference number in Table 133 on page 674
DEVICE(3790) SESSIONTYPE(3284CM)	19
DEVICE(3790) SESSIONTYPE(3286CM)	19

VTAM MODEENT macro operands

Table 133 shows the VTAM LOGON mode table entry for each TYPETERM you might define. You should have reached this table by looking up the TYPETERM device types in Table 132 on page 671.

Look down the left hand side of the table for the reference number (RN) that brought you here from Table 132 on page 671. When you find it, look across to the middle column. This shows the macro operands that affect the way CICS handles automatic installation. Your MODEENT macro entries for devices to be installed must match what is specified there. Any MODEENT macro entries not shown in the table, such as PSERVIC for some reference numbers, are not tested by CICS. Any bit settings that do not matter to CICS during bind analysis for autoinstalled terminals appear as periods (.).

Note: Some fields in the PSERVIC data for LUTYPE0, LUTYPE2, and LUTYPE3 devices have values that depend on the ALTSCREEN and DEFSCREEN characteristics of the device. For this reason, you have to consult “PSERVIC screen size values for LUTYPE0, LUTYPE2, and LUTYPE3 devices” on page 679 to find out the values you need to specify instead of aaaaaaaa, bbbbbbbb, cccccccc, dddddddd, and eeeeeeee.

The right-hand column in the table names entries in the CICS-supplied LOGON mode table that could meet your needs. The CICS-supplied table is called ISTINCLM. For further VTAM information, refer to *ACF/VTAM Network Implementation Guide*.

RN	VTAM MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
1	FMPROF=X'02' TSPROF=X'02' PRIPROT=X'70' SECPROT=X'40' COMPROT=B'0000.000 00000.00'	
2	FMPROF=X'02' TSPROF=X'02' PRIPROT=X'71' SECPROT=X'40' COMPROT=B'0010.000 00000.00'	DSILGMOD D4B32781 D4B32782 D4B32783 D4B32784 D4B32785 NSX32702 S3270
3	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'0000.000 00000.00'	
4	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'B0' SECPROT=X'90' COMPROT=B'0100.000 00000.00'	

Table 133 (Page 2 of 5). LOGON mode table and ISTINCLM entries

RN	VTAM MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
5	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0110.000 00000.00'	
6	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'31' SECPROT=X'30' COMPROT=B'0110.000 00000.00'	INTRUSER
7	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'30' COMPROT=B'0100.000 00000.00'	
8	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 01000.00' PSERVIC=B'00000001 00000000 00000000 00000000. 00000000 00000000 00000000 00000000 00000000 00000000'	
9	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000001 00000000 00000000 00000000. 00000000 00000000 00000000 00000000 00000000 00000000'	SCS
10	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 01000.00' PSERVIC=X'01'	
11	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0011.000 10000.00' PSERVIC=X'01'	SCS See note 2

VTAM MODEENT macro operands

<i>Table 133 (Page 3 of 5). LOGON mode table and ISTINCLM entries</i>		
RN	VTAM MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
12	FMPROF=X'07' TSPROF=X'07' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0101.000 10000.01' PSERVIC=B'00000100 10101000 01000000 10100000 10101000 01000000 10100000 00000000 00001100 00000000'	
13	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0011.000 10000.00' PSERVIC=X'01'	SCS3790 See note 2
14	FMPROF=X'03' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00110001 00011000 0100000. 00000000 10010010 00000000 00000000 00000000 01010000'	
15	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00110001 00001100 0111000. 00000000 11010010 00000000 00000000 00000000 11010000'	
16	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 10000.00'	See note 3
17	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=X'90' COMPROT=B'0111.000 10000.00' PSERVIC=B'00000001 00100000 00000000 00000000. 00000000 11000010 00000000 00000000 00000000 11000000'	

Table 133 (Page 4 of 5). LOGON mode table and ISTINCLM entries

RN	VTAM MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
18	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=B'10..0000' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000010 10000000 00000000 00000000 00000000 00000000 aaaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee'	D329001 D4A32771 D4A32772 D4A32781 D4A32782 D4A32783 D4A32784 D4A32785 D4C32771 D4C32772 D4C32781 D4C32782 D4C32783 D4C32784 D4C32785 D6327801 D6327802 D6327803 D6327804 D6327805 EMUDPCX EMU3790 SNX32702 See note 1
19	FMPROF=X'03' TSPROF=X'03' PRIPROT=X'B1' SECPROT=B'10..0000' COMPROT=B'0011.000 10000.00' PSERVIC=B'00000011 10000000 00000000 00000000 00000000 00000000 aaaaaaaaa bbbbbbbb cccccccc dddddddd eeeeeeee'	BLK3790 DSC2K DSC4K D6328902 D6328904 See note 1
20	FMPROF=X'04' TSPROF=X'03' PRIPROT=X'31' SECPROT=X'B0' COMPROT=B'0111.000'	
21	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'50' SECPROT=X'10' COMPROT=B'0000.000 00000.00'	
22	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'0100.000 00000.00'	IBMS3650

+

VTAM MODEENT macro operands

<i>Table 133 (Page 5 of 5). LOGON mode table and ISTINCLM entries</i>		
RN	VTAM MODEENT macro entries that are needed for related CICS TYPETERM definitions	Suitable supplied entries
23	FMPROF=X'04' TSPROF=X'04' PRIPROT=X'B1' SECPROT=X'B0' COMPROT=B'0111.000 00000.00'	
24	TYPE=X'00' FMPROF=X'13' TSPROF=X'07' PRIPROT=X'B0' SECPROT=X'B0' COMPROT=B'101.000 10110.01' PSERVIC='00000110 00000010 00000000 00000000 00000000 00000000 00000000 00000000 0010..00 00000000'	
Notes: <ol style="list-style-type: none"> 1. PSERVIC (RN 18 and 19): BYTE 2 BIT 0 should be set on where extended data stream (EXTDS) support is required. 2. RN 11 or 13 is used to determine the MODEENT macro operands for device SCSPRINT. However, if you have specified any of the attributes EXTENDED DS, COLOR, PROGSYMBOLS, HILIGHT, SOSI, OUTLINE, QUERY(COLD), or QUERY(ALL) for the TYPETERM, then the COMPROT parameter of RN 13 should be modified to read COMPROT=B'0111.000 10000.00'. 3. This LOGMODE may be used for either device type 4700 in half duplex mode or device types BCHLU, 3770, 3770B and 3790 with SESSIONTYPE(USERPROG). To enable these devices to be autoinstalled with the correct model, the model names list supplied to the autoinstall exit will list the names of models defined as DEVICE(3600) after the names of all other eligible models. The exit can be coded to select a name from the end of the list for a 4700 half duplex device. 		

PSERVIC screen size values for LUTYPE0, LUTYPE2, and LUTYPE3 devices

Table 134 is to help you decide what screen size values you should specify on the PSERVIC operand of the VTAM MODEENT macro, for LUTYPE0, LUTYPE2, and LUTYPE3 devices.

If, on your CICS TYPETERM definition, you code the values shown in columns 1 through 4 of Table 134, the screen size values in the CICS model bind image are as shown in column 5. The values you code for screen sizes on the PSERVIC operand must match this.

<i>Table 134. Autoinstall model device definition options</i>				
Device-type	DEFSCRN	ALTSCRN	QUERY	MODEL BIND
0,2,3	00,00	?	?	INVALID
0,2,3	12,40	,	?	0000000001
0,2,3	12,40	00,00	?	0C2800007E
0,2,3	12,40	YY,YY	?	0C28YYYY7F
0,2,3	24,80	,	NO	0000000002
3	24,80	,	COLD/ALL	0000000002
0,2	24,80	,	COLD/ALL	0000000003
0,2,3	24,80	00,00	?	185000007E
0,2,3	24,80	YY,YY	?	1850YYYY7F
0,2,3	XX,XX	,	?	XXXX00007E
0,2,3	XX,XX	00,00	?	XXXX00007E
0,2,3	XX,XX	YY,YY	?	XXXXYYYY7F
Where:				
0	=	Local non-SNA 3270		
2	=	LUTYPE2		
3	=	LUTYPE3		
,	=	Blanks (the default)		
XX,XX	=	A screen size that is not 12,40 or 24,80		
YY,YY	=	A screen size that is not 00,00 or blanks		
?	=	Any (that is, QUERY=ALL COLD NO, and ALTSCRN=any)		

CICS treats some differently-coded PSERVIC screen size specifications as equivalent. See Table 135 on page 680.

matching models and LOGON mode entries

Table 135. Equivalent PSERVIC screen size values	
Bytes 20—24 of CICS model bind	Valid screen size values on PSERVIC definition
0000 0000 01	0000 0000 00 0000 0000 01 0C28 0000 7E
0000 0000 02	0000 0000 00 0000 0000 02 1850 0000 7E
0000 0000 03	0000 0000 00 0000 0000 03 1850 0000 03
xxxx 0000 7E	0000 0000 00 xxxx 0000 7E
Plus, if xxxx=1850	0000 0000 02
xxxx yyyy 7F	0000 0000 00 xxxx yyyy 7F
Where: xxxx = 2 bytes containing the default screen size, in hexadecimal yyyy = 2 bytes containing the alternate screen size, in hexadecimal	

Matching models and LOGON mode entries

This section contains a set of VTAM LOGON mode table definitions, and their matching CICS autoinstall definitions. Each entry consists of a VTAM logmode definition, the matching CICS TYPETERM and model TERMINAL definitions, and (for information) the BIND that CICS sends based on the specified model definition.

Note that the CICS-specific attributes are purely arbitrary. Only device attributes affect the match algorithm. It is the responsibility of the autoinstall user program to distinguish between matching models.

```
*****
1) LOCAL NON-SNA 3277 / 3278 / 3279 (without special features)
*****
MT32772 MODEENT LOGMODE=MT32772, 3277/8 MODEL 2
      TYPE=1,
      FMPROF=X'02',
      TSPROF=X'02',
      PRIPROT=X'71',
      SECPROT=X'40',
      COMPROT=X'2000',
      PSERVIC=X'00000000000000000000200'
OR
      PSERVIC=X'00000000000018502B507F00' Others
OR
      PSERVIC=X'000000000000185000007E00' Model 2, no AltScreen
```


TERMINAL definition

```
*****
AUTINSTNAME ==> M3278A
AUTINSTMODEL ==> ONLY
GROUP ==> PDATD
TYPETERM ==> T3278
INSERVICE ==> YES
```

TYPETERM definition

```
*****
TYPETERM ==> T3278
GROUP ==> PDATD
DEVICE ==> 3270
TERMMODEL ==> 2
LIGHTPEN ==> YES
AUDIBLEALARM ==> YES
UCTRAN ==> YES
IOAREALEN ==> 2000,2000
ERRLASTLINE ==> YES
ERRINTENSIFY ==> YES
USERAREALEN ==> 32
ATI ==> YES
TTI ==> YES
AUTOCONNECT ==> NO
LOGONMSG ==> YES
```

BIND SENT BY CICS depends on PSERVIC value on LOGMODE definition above:

```
EITHER : 01020271 40200000 00000080 00000000
          00000000 00000002 00009300 00300000
OR      : 01020271 40200000 00000080 00000000
          00000018 502B507F 00009300 00300000
OR      : 01020271 40200000 00000080 00000000
Real Model 2 00000018 5000007E 00009300 00300000
```

2) LOCAL SNA 3277/78/79 (without special features) LUTYPE2

```
S32782 MODEENT LOGMODE=S32782, SNA LUTYPE2 3270
          TYPE=1,
          FMPROF=X'03',
          TSPROF=X'03',
          PRIPROT=X'B1',
          SECPROT=X'B0',
          COMPROT=X'3080',
          RUSIZES=X'8585',
          PSERVIC=X'028000000000185018507F00'
```

TERMINAL definition

```
*****
AUTINSTNAME ==> M32782
AUTINSTMODEL ==> ONLY
GROUP ==> PDATD
TYPETERM ==> T32782
INSERVICE ==> YES
```

matching models and LOGON mode entries

TYPETERM definition

TYPETERM ==> T32782
GROUP ==> PDATD
DEVICE ==> LUTYPE2
TERMMODEL ==> 2
LIGHTPEN ==> YES
AUDIBLEALARM ==> YES
UCTRAN ==> YES
IOAREALEN ==> 256,256
ERRLASTLINE ==> YES
ERRINTENSIFY ==> YES
USERAREALEN ==> 32
ATI ==> YES
TTI ==> YES
LOGONMSG ==> YES
DISCREQ ==> YES
RECEIVESIZE ==> 256
BUILDCHAIN ==> YES

BIND SENT BY CICS : 010303B1 B0308000 0085C780 00028000
 00000018 5018507F 00000000 00000000

3) 3770 BATCH LU (3777)

BATCH MODEENT LOGMODE=BATCH, 3770 BATCH
 TYPE=1,
 FMPROF=X'03',
 TSPROF=X'03',
 PRIPROT=X'B1',
 SECPROT=X'B0',
 COMPROT=X'7080',
 PSERVIC=X'01310C70E100D20000E100D0'

TERMINAL definition

AUTINSTNAME ==> M3770
AUTINSTMODEL ==> ONLY
GROUP ==> PDATD
TYPETERM ==> T3770
INSERVICE ==> YES

TYPETERM definition

TYPETERM ==> T3770
 GROUP ==> PDATD
 DEVICE ==> 3770
 SESSIONTYPE ==> BATCHDI
 PAGESIZE ==> 12,80
 DISCREQ ==> YES
 AUTOPAGE ==> YES
 RECEIVESIZE ==> 256
 SENDSIZE ==> 256
 IOAREALEN ==> 256,2048
 BUILDCHAIN ==> YES
 BRACKET ==> YES
 ATI ==> YES
 TTI ==> YES
 AUTOCONNECT ==> NO
 HORIZFORM ==> YES
 VERTFORM ==> YES
 LDCLIST ==> LDC2

Needs LDC declaration in TCT :

LDC2 DFHTCT TYPE=LDC,LOCAL=INITIAL
 DFHTCT TYPE=LDC,LDC=BCHLU
 DFHTCT TYPE=LDC,LOCAL=FINAL

BIND SENT BY CICS : 010303B1 B0708000 00000080 0001310C
 70E100D2 0000E100 D0000000 00000000

4) 6670 LUTYPE4

S6670 MODEENT LOGMODE=S6670, 6670 LUTYPE4
 TYPE=1,
 FMPROF=X'07',
 TSPROF=X'07',
 RUSIZES=X'8585',
 PRIPROT=X'B1',
 SECPROT=X'B0',
 COMPROT=X'5081',
 PSERVIC=X'04A840A000A840A000000C00'

TERMINAL definition

AUTINSTNAME ==> M6670
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TYPETERM ==> T6670
 INSERVICE ==> YES

matching models and LOGON mode entries

TYPETERM definition

```
TYPETERM      ==> T6670
GROUP         ==> PDATD
DEVICE        ==> LUTYPE4
BUILDCHAIN    ==> YES
DISCREQ       ==> YES
RECEIVESIZE   ==> 256
UCTRAN        ==> YES
IOAREALEN     ==> 256,4096
FORMFEED      ==> YES
HORIZFORM     ==> YES
VERTFORM      ==> YES
ATI           ==> YES
TTI           ==> YES
PAGESIZE      ==> 50,80
AUTOPAGE      ==> YES
LOGONMSG      ==> NO
LDCLIST       ==> LDC1
```

Needs LDC declaration in TCT :

```
LDCS          DFHTCT TYPE=LDC,LDC=SYSTEM
LDC1          DFHTCT TYPE=LDC,LOCAL=INITIAL
              DFHTCT TYPE=LDC,DVC=(BLUCON,01),PROFILE=DEFAULT,LDC=PC,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(BLUPRT,02),PROFILE=BASE,LDC=PP,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(BLUPRT,08),PROFILE=BASE,LDC=P8,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(BLUPRT,08),PROFILE=DEFAULT,LDC=DP,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(BLUPCH,03),PROFILE=JOB,LDC=PM,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(BLUPCH,03),PROFILE=DEFAULT,LDC=DM,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED1,04),PROFILE=WPRAW,LDC=P1,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED1,04),PROFILE=DEFAULT,LDC=D1,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED2,05),PROFILE=OII1,LDC=P2,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED2,05),PROFILE=DEFAULT,LDC=D2,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED3,06),PROFILE=OII2,LDC=P3,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,DVC=(WPMED4,07),PROFILE=OII3,LDC=P4,
              PGESIZE=(50,80),PGESTAT=AUTOPAGE
              DFHTCT TYPE=LDC,LOCAL=FINAL
```

```
BIND SENT BY CICS :          010707B1 B0508100 00858580 0004A840
                              A000A840 A000000C 00000000 00000000
```

 5) 3790 FULL FUNCTION LU

S3790A MODEENT LOGMODE=S3790A, 3790 FULL FUNCTION LU
 TYPE=1,
 FMPROF=X'04',
 TSPROF=X'04',
 PRIPROT=X'B1',
 SECPROT=X'B0',
 RUSIZES=X'8585',
 COMPROT=X'7080'

TERMINAL definition

AUTINSTNAME ==> M3790A
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TYPETERM ==> T3790A
 INSERVICE ==> YES

TYPETERM definition

TYPETERM ==> T3790A
 GROUP ==> PDATD
 DEVICE ==> 3790
 SENDSIZE ==> 256
 RECEIVESIZE ==> 256
 SESSIONTYPE ==> USERPROG
 BRACKET ==> YES
 IOAREALEN ==> 256
 ATI ==> YES
 TTI ==> YES

BIND SENT BY CICS : 010404B1 B0708000 00858580 00000000

6) 3790 BATCH DATA INTERCHANGE

S3790B MODEENT LOGMODE=S3790B, 3790 BATCH
 TYPE=1,
 FMPROF=X'03',
 TSPROF=X'04',
 PRIPROT=X'B1',
 SECPROT=X'B0',
 COMPROT=X'7080',
 RUSIZES=X'8585',
 PSERVIC=X'013118400000920000E10050'

TERMINAL definition

AUTINSTNAME ==> M3790B
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TYPETERM ==> T3790B
 INSERVICE ==> YES
 TERMPRIORITY ==> 50

matching models and LOGON mode entries

TYPETERM definition

```
*****
TYPETERM      ==> T3790B
GROUP         ==> PDATD
DEVICE        ==> 3790
SESSIONTYPE   ==> BATCHDI
AUTOPAGE      ==> YES
BUILDCHAIN    ==> YES
OBOPERID      ==> YES
IOAREALEN     ==> 256,2048
RELREQ        ==> YES
SENDSIZE      ==> 256
RECEIVESIZE   ==> 256
ATI           ==> YES
TTI           ==> YES
LDCLIST       ==> LDC2
```

Needs LDC declaration in TCT :

```
LDC2  DFHTCT TYPE=LDC,LOCAL=INITIAL
      DFHTCT TYPE=LDC,LDC=BCHLU
      DFHTCT TYPE=LDC,LOCAL=FINAL
```

```
BIND SENT BY CICS :          010304B1 B0708000 00858580 00013118
                              40000092 0000E100 50000000 00000000
```

```
*****
```

7) 3790 SCSVRT

```
*****
```

```
S3790C  MODEENT LOGMODE=S3790C,  3790 WITH SCS
        TYPE=1,
        FMPROF=X'03',
        TSPROF=X'03',
        PRIPROT=X'B1',
        SECPROT=X'B0',
        COMPROT=X'3080',
        RUSIZES=X'8585',
        PSERVIC=X'010000000000000000000000'
```

TERMINAL definition

```
*****
AUTINSTNAME   ==> M3790C
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3790C
INSERVICE     ==> YES
```

TYPETERM definition

```
*****
TYPETERM      ==> T3790C
GROUP         ==> PDATD
DEVICE        ==> 3790
SESSIONTYPE   ==> SCSVRT
BRACKET       ==> YES
SENDSIZE      ==> 256
RECEIVESIZE   ==> 256
ATI           ==> YES
TTI           ==> YES
```

```
BIND SENT BY CICS :          010303B1 B0308000 00858580 00010000
```

Note that CEDA changes DEVICE=3790,
SESSIONTYPE=SCSPRT to DEVICE=SCSPRINT,
SESSIONTYPE=blanks, PRINTERTYPE=3284.

 8) 3767 INTERACTIVE (FLIP-FLOP) LU

S3767 MODEENT LOGMODE=S3767, 3767 INTERACTIVE
 TYPE=1,
 FMPROF=X'03',
 TSPROF=X'03',
 PRIPROT=X'B1',
 SECPROT=X'90',
 COMPROT=X'3080',
 PSERVIC=X'010000000000000000000000'

TERMINAL definition

 AUTINSTNAME ==> M3767
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TERMPRIORITY ==> 60
 TYPETERM ==> T3767
 INSERVICE ==> YES

TYPETERM definition

 TYPETERM ==> T3767
 GROUP ==> PDATD
 DEVICE ==> 3767
 VERTFORM ==> YES
 HORIZFORM ==> YES
 RELREQ ==> YES
 DISCREQ ==> YES
 IOAREALEN ==> 256
 AUTOPAGE ==> NO
 PAGESIZE ==> 12,80
 ATI ==> YES
 TTI ==> YES
 BRACKET ==> YES
 RECEIVESIZE ==> 256
 SENDSIZE ==> 256

BIND SENT BY CICS : 010303B1 90308000 00000080 00010000

 9) 3650 INTERPRETER LU
 (SESTYPE = USERPROG BRACKET = YES)

S3650A MODEENT LOGMODE=S3650A, 3650 SESTYPE=USERPROG
 BRACKET=YES
 TYPE=1,
 FMPROF=X'04',
 TSPROF=X'04',
 PRIPROT=X'31',
 SECPROT=X'30',
 COMPROT=X'6000'

TERMINAL definition

 AUTINSTNAME ==> M3650A
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TYPETERM ==> T3650A
 INSERVICE ==> YES

matching models and LOGON mode entries

```
TYPETERM definition
*****
TYPETERM      ==> T3650A
GROUP         ==> PDATD
DEVICE        ==> 3650
SESSIONTYPE   ==> USERPROG
ROUTEDMSGS    ==> SPECIFIC
FMHPARM       ==> YES
RELREQ        ==> YES
DISCREQ       ==> YES
BRACKET       ==> YES
RECEIVESIZE   ==> 256
IOAREALEN     ==> 256,256
ATI           ==> YES
TTI           ==> YES
AUTOCONNECT   ==> NO

BIND SENT BY CICS :          01040431 30600000 00000080 00000000

*****
10) 3650 HOST CONVERSATIONAL (3270) LU
*****
S3650B  MODEENT LOGMODE=S3650B,  3650 SESTYPE=3270
        TYPE=1,                  AND SESTYPE=3653
        FMPROF=X'04',
        TSPROF=X'03',
        PRIPROT=X'B1',
        SECPROT=X'90',
        COMPROT=X'6000'

TERMINAL definition
*****
AUTINSTNAME   ==> M3650B1
AUTINSTMODEL  ==> ONLY
GROUP         ==> PDATD
TYPETERM      ==> T3650B1
INSERVICE     ==> YES

TYPETERM definition
*****
TYPETERM      ==> T3650B1
GROUP         ==> PDATD
DEVICE        ==> 3650
OBFORMAT      ==> YES
SESSIONTYPE   ==> 3270
RELREQ        ==> YES
DISCREQ       ==> YES
IOAREALEN     ==> 256
BRACKET       ==> YES
RECEIVESIZE   ==> 240
ATI           ==> NO
TTI           ==> YES

BIND SENT BY CICS :          010403B1 90600000 00000080 00000000
```



```
*****
11) 3650 HOST CONVERSATIONAL (3653) LU
    (N.B. LOGMODE SAME AS HC (3270) LU)
*****
```

```
S3650B  MODEENT LOGMODE=S3650B,   3650 SESTYPE=3270
          TYPE=1,                   AND SESTYPE=3653
          FMPROF=X'04',
          TSPROF=X'03',
          PRIPROT=X'B1',
          SECPROT=X'90',
          COMPROT=X'6000'
```

TERMINAL definition

```
*****
AUTINSTNAME ==> M3650B2
AUTINSTMODEL ==> ONLY
GROUP       ==> PDATD
TYPETERM   ==> T3650B2
INSERVICE  ==> YES
```

TYPETERM definition

```
*****
TYPETERM   ==> T3650B2
GROUP      ==> PDATD
DEVICE     ==> 3650
SESSIONTYPE ==> 3653
RELREQ     ==> YES
DISCREQ   ==> NO
BRACKET    ==> YES
IOAREALEN ==> 256
RECEIVESIZE ==> 240
ROUTEDMSGS ==> NONE
ATI        ==> NO
TTI        ==> YES
```

```
BIND SENT BY CICS :          010403B1 90600000 00000080 00000000
```

```
*****
12) 3650 HOST COMMAND PROCESSOR LU
    (SESTYPE = USERPROG BRACKET = NO)
*****
```

```
S3650C  MODEENT LOGMODE=S3650C,   3650 SESTYPE=USERPROG
          TYPE=1,                   BRACKET=NO
          FMPROF=X'04',
          TSPROF=X'04',
          PRIPROT=X'B0',
          SECPROT=X'30',
          COMPROT=X'4000'
```

TERMINAL definition

```
*****
AUTINSTNAME ==> M3650C
AUTINSTMODEL ==> ONLY
GROUP       ==> PDATD
TYPETERM   ==> T3650C
INSERVICE  ==> YES
```

matching models and LOGON mode entries

TYPETERM definition

```
*****  
TYPETERM      ==> T3650C  
GROUP         ==> PDATD  
DEVICE        ==> 3650  
SESSIONTYPE   ==> USERPROG  
BRACKET       ==> NO  
RELREQ        ==> NO  
DISCREQ       ==> NO  
RECEIVESIZE   ==> 256  
IOAREALEN     ==> 256  
ATI           ==> YES  
TTI           ==> YES
```

```
BIND SENT BY CICS :          01040430 30400000 00000080 00000000
```

```
*****  
13) 8815 SCANMASTER (APPC SINGLE SESSION)
```

```
*****  
SIN62 MODEENT LOGMODE=SIN62,      8815 SCANMASTER.  
      TYPE=0,  
      FMPROF=X'13',  
      TSPROF=X'07',  
      PRIPROT=X'B0',  
      SECPROT=X'B0',  
      COMPROT=X'50B1',  
      PSNDPAC=X'00',  
      SRCVPAC=X'00',  
      SSNDPAC=X'00',  
      RUSIZES=X'8585',  
      PSERVIC=X'060200000000000000002C00'
```

TERMINAL definition

```
*****  
AUTINSTNAME   ==> MLU62  
AUTINSTMODEL  ==> ONLY  
GROUP         ==> PDATD  
TYPETERM      ==> SINLU62  
INSERVICE     ==> YES
```

TYPETERM definition

```
*****  
TYPETERM      ==> SINLU62  
GROUP         ==> PDATD  
DEVICE        ==> APPC  
RECEIVESIZE   ==> 2048  
SENDSIZE      ==> 2048  
ATI           ==> YES  
TTI           ==> YES
```

Note: There is no RDO keyword equivalent of the MACRO keyword 'FEATURE=SINGLE', because this is assumed with RDO DEFINE TYPETERM when DEVICE=APPC.

```
BIND SENT BY CICS :          001307B0 B050B100 00858580 00060200  
                              00000000 0000002C 00000800 00000000  
                              0000001D 00090240 40404040 40404009  
                              03006765 71D98A6C 300704C3 C9C3E2E6  
                              F1000000 00000000 00000000 00000000
```

 14) 3290 (SDLC)

S3290 MODEENT LOGMODE=S3290, 3290 SDLC
 TYPE=1,
 FMPROF=X'03',
 TSPROF=X'03',
 PRIPROT=X'B1',
 SECPROT=X'90',
 COMPROT=X'3080',
 RUSIZES=X'8787',
 PSERVIC=X'02800000000018503EA07F00'

TERMINAL definition

 AUTINSTNAME ==> M3290
 AUTINSTMODEL ==> ONLY
 GROUP ==> PDATD
 TYPETERM ==> T3290
 INSERVICE ==> YES

TYPETERM definition

 TYPETERM ==> T3290
 GROUP ==> PDATD
 DEVICE ==> LUTYPE2
 TERMMODEL ==> 2
 ALTSCREEN ==> 62,160
 DEFSCREEN ==> 24,80
 AUDIBLEALARM ==> YES
 UCTRAN ==> YES
 IOAREALEN ==> 2000,2000
 ERRLASTLINE ==> YES
 ERRINTENSIFY ==> YES
 USERAREALEN ==> 32
 ATI ==> YES
 TTI ==> YES
 LOGONMSG ==> YES
 ERRHILIGHT ==> BLINK
 RECEIVESIZE ==> 1024

BIND SENT BY CICS : 010303B1 90308000 00878780 00028000
 00000018 503EA07F 00000000 00000000

 15) 3601 WITH A 3604 ATTACHED

S3600 MODEENT LOGMODE=S3600, 3601
 TYPE=1,
 FMPROF=X'04',
 TSPROF=X'04',
 PRIPROT=X'B1',
 SECPROT=X'B0',
 COMPROT=X'7000',
 RUSIZES=X'0000'

definitions for CICS-supplied autoinstall models

TERMINAL definition

AUTINSTNAME ==> M3600
AUTINSTMODEL ==> ONLY
GROUP ==> PDATD
TERMPRIORITY ==> 50
TYPETERM ==> T3600
INSERVICE ==> YES

TYPETERM definition

TYPETERM ==> T3600
GROUP ==> PDATD
DEVICE ==> 3600
AUTOPAGE ==> NO
PAGESIZE ==> 6,40
RELREQ ==> YES
DISCREQ ==> NO
IOAREALEN ==> 256
SENDSIZE ==> 224
RECEIVESIZE ==> 256
USERAREALEN ==> 100
ATI ==> NO
TTI ==> YES
BRACKET ==> YES
LDCLIST ==> BMSLLDC1

Needs LDC declaration in TCT :

```
BMSLLDC1 DFHTCT TYPE=LDCLIST,  
          LDC=(DS,JP,PB=5,LP,MS)  
          DFHTCT TYPE=LDC,  
          LDC=(DS=1),  
          DVC=3604,  
          PGESIZE=(6,40),  
          PGESTAT=PAGE  
          DFHTCT TYPE=LDC,LDC=SYSTEM
```

BIND SENT BY CICS : 010404B1 B0700000 00000080 00000000

LOGON mode definitions for CICS-supplied autoinstall models

This section contains VTAM LOGON mode table definitions that match the CICS-supplied TYPETERM and model TERMINAL definitions for autoinstall. The first six entries are example definitions; that is, they are not supplied with VTAM.

```
DFHLU3  MODEENT LOGMODE=DFHLU3,  LU TYPE 3 PRINTER.  
        TYPE=1,  
        FMPROF=X'03',  
        TSPROF=X'03',  
        PRIPROT=X'B1',  
        SECPROT=X'B0',  
        COMPROT=X'3080',  
        RUSIZES=X'8585',  
        PSERVIC=X'038000000000000000000000200'
```

definitions for CICS-supplied autoinstall models

```

DFHSCSP  MODEENT LOGMODE=DFHSCSP, LU TYPE 1 SCS PRINTER
          TYPE=1,
          FMPROF=X'03',
          TSPROF=X'03',
          PRIPROT=X'B1',
          SECPROT=X'B0',
          COMPROT=X'7080',
          RUSIZES=X'8585',
          PSERVIC=X'010000010000000000000000'

DFHLU62T MODEENT LOGMODE=DFHLU62T, APPC SINGLE-SESSION
          TYPE=0,
          FMPROF=X'13',
          TSPROF=X'07',
          PRIPROT=X'B0',
          SECPROT=X'B0',
          COMPROT=X'50B1',
          RUSIZES=X'8888',
          PSERVIC=X'06020000000000000000002C00'

DFH3270  MODEENT LOGMODE=DFH3270, 3270
          TYPE=1,
          FMPROF=X'02',
          TSPROF=X'02',
          PRIPROT=X'71',
          SECPROT=X'40',
          COMPROT=X'2000',
          RUSIZES=X'0000'

DFH3270P MODEENT LOGMODE=DFH3270P, 3284/3286 BISYNC 3270P (QUERY)
          TYPE=1,
          FMPROF=X'02',
          TSPROF=X'02',
          PRIPROT=X'71',
          SECPROT=X'40',
          COMPROT=X'2000',
          RUSIZES=X'0000'

DFHLU2   MODEENT LOGMODE=DFHLU2, SNA LUTYPE2 3270
          TYPE=1,
          FMPROF=X'03',
          TSPROF=X'03',
          PRIPROT=X'B1',
          SECPROT=X'B0',
          COMPROT=X'3080',
          RUSIZES=X'85C7',
          PSERVIC=X'0280000000000000000000300'

```

The following entries are those LOGMODE definitions supplied by VTAM that match CICS-supplied TYPETERM definitions.

```

DFHLU0E2 MODEENT LOGMODE=NSX32702, LU0 model 2 queryable
          FMPROF=X'02',
          TSPROF=X'02',
          PRIPROT=X'71',
          SECPROT=X'40',
          COMPROT=X'2000',
          RUSIZES=X'0000',
          PSERVIC=X'008000000000185000007E00'

```

definitions for CICS-supplied autoinstall models

```
DFHLU0M2 MODEENT LOGMODE=D4B32782, LU0 model 2 nonqueryable
      FMPROF=X'02',
      TSPROF=X'02',
      PRIPROT=X'71',
      SECPROT=X'40',
      COMPROT=X'2000',
      RUSIZES=X'0000',
      PSERVIC=X'000000000000185000007E00'

DFHLU0M3 MODEENT LOGMODE=D4B32783, LU0 model 3 nonqueryable
      FMPROF=X'02',
      TSPROF=X'02',
      PRIPROT=X'71',
      SECPROT=X'40',
      COMPROT=X'2000',
      RUSIZES=X'0000',
      PSERVIC=X'000000000000185020507F00'

DFHLU0M4 MODEENT LOGMODE=D4B32784, LU0 model 4 nonqueryable
      FMPROF=X'02',
      TSPROF=X'02',
      PRIPROT=X'71',
      SECPROT=X'40',
      COMPROT=X'2000',
      RUSIZES=X'0000',
      PSERVIC=X'00000000000018502B507F00'

DFHLU0M5 MODEENT LOGMODE=D4B32785, LU0 model 5 nonqueryable
      FMPROF=X'02',
      TSPROF=X'02',
      PRIPROT=X'71',
      SECPROT=X'40',
      COMPROT=X'2000',
      RUSIZES=X'0000',
      PSERVIC=X'00000000000018501B847F00'

DFHLU2E2 MODEENT LOGMODE=SNX32702, LU2 model 2 queryable
      FMPROF=X'03',
      TSPROF=X'03',
      PRIPROT=X'B1',
      SECPROT=X'90',
      COMPROT=X'3080',
      RUSIZES=X'87F8',
      PSERVIC=X'028000000000185000007E00'

DFHLU2E3 MODEENT LOGMODE=SNX32703, LU2 model 3 queryable
      FMPROF=X'03',
      TSPROF=X'03',
      PRIPROT=X'B1',
      SECPROT=X'90',
      COMPROT=X'3080',
      RUSIZES=X'87F8',
      PSERVIC=X'028000000000185020507F00'
```

definitions for CICS-supplied autoinstall models

```
DFHLU2E4 MODEENT LOGMODE=SNX32704, LU2 model 4 queryable
    FMPROF=X'03',
    TSPROF=X'03',
    PRIPROT=X'B1',
    SECPROT=X'90',
    COMPROT=X'3080',
    RUSIZES=X'87F8',
    PSERVIC=X'02800000000018502B507F00'

DFHLU2M2 MODEENT LOGMODE=D4A32782, LU2 model 2 nonqueryable
    FMPROF=X'03',
    TSPROF=X'03',
    PRIPROT=X'B1',
    SECPROT=X'90',
    COMPROT=X'3080',
    RUSIZES=X'87C7',
    PSERVIC=X'020000000000185000007E00'

DFHLU2M3 MODEENT LOGMODE=D4A32783, LU2 model 3 nonqueryable
    FMPROF=X'03',
    TSPROF=X'03',
    PRIPROT=X'B1',
    SECPROT=X'90',
    COMPROT=X'3080',
    RUSIZES=X'87C7',
    PSERVIC=X'020000000000185020507F00'

DFHLU2M4 MODEENT LOGMODE=D4A32784, LU2 model 4 nonqueryable
    FMPROF=X'03',
    TSPROF=X'03',
    PRIPROT=X'B1',
    SECPROT=X'90',
    COMPROT=X'3080',
    RUSIZES=X'87C7',
    PSERVIC=X'02000000000018502B507F00'

DFHLU2M5 MODEENT LOGMODE=D4A32785, LU2 model 5 nonqueryable
    FMPROF=X'03',
    TSPROF=X'03',
    PRIPROT=X'B1',
    SECPROT=X'90',
    COMPROT=X'3080',
    RUSIZES=X'87C7',
    PSERVIC=X'02000000000018501B847F00'
```

definitions for CICS-supplied autoinstall models

Appendix B. Default actions of the node abnormal condition program

This appendix describes the default actions of the node abnormal condition program, DFHZNAC. The actions vary, depending on the terminal error code and system sense codes received from VTAM. In most cases, DFHZNAC issues messages and sets one or more “action flags” in the communication area passed to the node error program, DFHZNEP. DFHZNEP then has the opportunity to change the default actions (though not the messages) by setting or resetting flags. (Note, however, that in some circumstances, the actions actually taken can vary from the actions set, depending on the state of the node at the time of the error.)

For more information about DFHZNAC and DFHZNEP, see Chapter 9, “Writing a node error program” on page 395.

Default actions for terminal error codes

Terminal error codes from VTAM are put in a 1-byte field (TWAEC) of the communications area passed to DFHZNEP.

Table 136 shows the message issued and action flags set by DFHZNAC for each terminal error code.

For error codes with CICS messages associated with them, see the *CICS/ESA Messages and Codes* manual for descriptions of the corresponding error conditions.

The figures in the “**Action flags set**” column are translated into bit settings and explained in Table 139 on page 711.

Error code	Symbolic label	Message	Action flags set
X'10'	TCZSRCTU	DFHZC2405	18
X'11'	TCZSRCBF	DFHZC2403	2 5 18 24
X'13'	TCZSRCVH	DFHZC2416	18 24
X'14'	TCZLRCER	DFHZC2404	2 3 9 10 11 23 24
X'15'	TCZSRCPF	DFHZC2407	2 3 9 10 11 24
X'16'	TCZDMIT	DFHZC3492	None
X'18'	TCZLRCNR	DFHZC2404	2 3 9 10 11 23 24
X'19'	TCZSRCTS	DFHZC2406	9 10 11 18
X'1A'	TCZSRCVE	DFHZC2408	2 3 9 10 11 24
X'1D'	TCZSRCVI	DFHZC2417	2 24
X'1E'	TCZSRCV2	DFHZC2408	2 3 9 10 11 24
X'20'	TCZVTAMI	DFHZC2417	None
X'21'	TCZLUCF1	DFHZC4902	3 9 10 11 24
X'22'	TCZLUCF2	DFHZC4903	3 9 10 11 24

default actions of DFHZNAC

Table 136 (Page 2 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'23'	TCZFMSBE	DFHZC4904	3 9 10 11 24
X'24'	TCZFMSCS	DFHZC4905	3 9 10 11 24
X'25'	TCZFSMCR	DFHZC4906	3 9 10 11 24
X'26'	TCZSDLER	DFHZC4907	3 9 10 11 24
X'28'	TCZRVLER	DFHZC4909	3 9 10 11 24
X'29'	TCZRVLRB	DFHZC4910	3 9 10 11 24
X'2A'	TCZRLPEX	DFHZC4911	2 3 9 10 11 24
X'2B'	TCZRLPBD	DFHZC4912	2 3 9 10 11 24
X'2C'	TCZRLPDR	DFHZC4913	2 3 9 10 11 24
X'2D'	TCZRLPIL	DFHZC4914	2 3 9 10 11 24
X'2E'	TCZRLPEC	DFHZC4915	2 3 9 10 11 24
X'2F'	TCZRLPRR	DFHZC4916	2 3 9 10 11 24
X'30'	TCZRLPIF	DFHZC4917	2 3 9 10 11 24
X'31'	TCZRLPIR	DFHZC4918	2 3 9 10 11 24
X'32'	TCZRLXCL	DFHZC4922	20
X'33'	TCZIVIND	DFHZC4919	2 3 9 10 11 24
X'34'	TCZIVDAT	DFHZC4920	2 3 9 10 11 24
X'35'	TCZRTMT	DFHZC4930	2 3 9 10 11 24
X'36'	TCZXSBL	None	24
X'37'	TCZXSHRA	DFHZC3470	9 10 11 24 and sometimes 15
X'38'	TCZXSWAS	DFHZC6596	2 3 15 24
X'39'	TCZXSABN	DFHZC6595	2 3 5 24
X'3A'	TCZXSHR	DFHZC6594	24
X'3B'	TCZXSBC	DFHZC6593	None
X'3C'	TCZXUVAR	DFHZC3488	2 3 9 10 11 24
X'3D'	TCZXMSG	None	None
X'3E'	TCZXERR	DFHZC6591	9 10 11 15 24
X'3F'	TCZXRST	DFHZC6590	None
X'40'	TCZINCPY	DFHZC2489	3 9 11
X'41'	TCZTOLRQ	DFHZC2490	2 3 9 10 11 15 24
X'42'	TCZUNPRT	DFHZC2497 ¹	None
X'43'	TCZCPYNS	DFHZC2434	3 11
X'44'	TCZSRCDE	DFHZC2456	2 3 9 10 11 24
X'45'	TCZCHMX	DFHZC3400	3 10 11 22
X'46'	TCZOCIR	DFHZC3402	3 9 10 11
X'47'	TCZGMMS	None ²	13

#

Table 136 (Page 3 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'48'	TCZOPSIN	DFHZC3461	None
X'49'	TCZCLSIN	DFHZC3462	None
X'4A'	TCZOPACB	DFHZC3463	None
X'4B'	TCZICPUT	DFHZC2498	None
X'4C'	TCZDSPCL	DFHZC3481	2 3 9 10 11 24
X'4D'	TCZSL SRL	DFHZC3473	None
X'4E'	TCZUNBFE	DFHZC3479	2 3 9 10 11 24
X'4F'	TCZCNOS0	None	None
X'50'	TCZSDRE3	DFHZC3417	3 9 10 11 24
X'51'	TCZBDPRI	DFHZC3418	3 9 10 11 24
X'52'	TCZBDUAC	DFHZC3419	2 3 5
X'53'	TCZBDTOS	DFHZC3420	20
X'54'	TCZUNBIS	DFHZC3434	2 3 9 10 11 24
X'55'	TCZEMWBK	DFHZC3440	None
X'56'	TCZXRFVS	DFHZC6598	None
X'57'	TCZRELIS	DFHZC3464	20
X'58'	TCZERMGR	DFHZC3433	None
X'59'	TCZROCT	DFHZC2443	2 3 9 10 11 24
X'5A'	TCZSBIRV	DFHZC3421	20
X'5B'	TCZNSP01	DFHZC3422	2 3 9 10 11 24
X'5C'	TCZNSP02	DFHZC3424	9 10 11 15 24
X'5D'	TCZPRDTO	DFHZC0101	None
X'5E'	TCZBRUAC	DFHZC3454	2 3 5 18 24
X'5F'	TCZBDSQP	DFHZC3455	2 3 5 18 24
X'60'	TCZUNCMD	DFHZC2421	2 3 9 10 11 24
X'62'	TCZVTAMQ	None ³	24
X'63'	TCZVTAMO	DFHZC3441	None
X'64'	TCZVTAMA	DFHZC3443	None
X'65'	TCZINVR	DFHZC2448	2 3 10 11 22 23 24
X'66'	TCZSIGR	DFHZC3452	None
X'67'	TCZVTAMK	DFHZC3442	None
X'69'	TCZSEXOS	DFHZC3466	20 23
X'6A'	TCZTIOAE	DFHZC3444	1 2 3 9 10 11 24
X'6B'	TCZNOTNA	DFHZC3495	24
X'6C'	TCZPSAF	DFHZC0155	3 6 9 10 11 24
X'6D'	TCZPSAR	DFHZC0156	None
X'6F'	TCZSDAS	DFHZC2351	2 3

#

default actions of DFHZNAC

Table 136 (Page 4 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'70'	TCZCLRRV	DFHZA3468	9 10 11 15 24
X'71'	TCZPSLE	DFHZA0147	3 6 9 10 11 24
X'72'	TCZPSVF	DFHZA0148	9 10 11 24
X'73'	TCZSDSE4	DFHZA2437	3 9 11
X'74'	TCZSDSE5	DFHZA2423	3 9 10 11 24
X'75'	TCZSESE1	DFHZA2424	3 9 10 11 15 24
X'76'	TCZLGNA	DFHZA2487	3
X'77'	TCZDMRY	DFHZA2488	None
X'78'	TCZSDRE2	DFHZA2430	3 9 11 22
X'79'	TCZPSRAF	DFHZA0145	3 6 9 10 11 24
X'7A'	TCZPSRAC	DFHZA0144	11
X'7D'	TCZRABUS	DFHZA4949	2 3 9 10 11 24
X'80'	TCZSRCSP	DFHZA2414	None
X'81'	TCZSSXNR	DFHZA2432	None
X'82'	TCZSSXUC	DFHZA2419	2 3 9 10 11 23 24
X'83'	TCZSSXAR	DFHZA2450	None
X'84'	TCZSSXIB	DFHZA2446	2 3 9 10 11 23 24
X'85'	TCZUNEGR	DFHZA3409	2 3 9 10 11 23 24
X'88'	TCZLEXCI	DFHZA2467	2 3 9 10 11 23 24
X'89'	TCZLEXUS	DFHZA2468	2 3 9 10 11 24
X'8A'	TCZLUSRR	DFHZA4937	2 3 5 24
X'8B'	TCZLUSRF	DFHZA4938	2 3 5 24
X'8C'	TCZLUPUN	DFHZA4939	2 3 5 24
X'8D'	TCZLUPLK	DFHZA4941	2 3 5 24
X'8E'	TCZLUPEX	DFHZA4942	2 3 5 24
X'8F'	TCZLUSKN	DFHZA4940	2 3 5 24
X'90'	TCZLGCER	DFHZA2422	1 2 3 6 9 10 11 23 24
X'91'	TCZRSTLE	DFHZA2429	3 10 11
X'92'	TCZSDSE6	DFHZA2428	3 9 11
X'93'	TCZRACET	DFHZA2455	2 3 9 10 11
X'94'	TCZRACES	DFHZA2426	2 3 9 10 11 22
X'95'	TCZSDSE8	DFHZA2445	3 9 11
X'96'	TCZRVSZ1	DFHZA2435	3 10 11 24
X'97'	TCZRVSZ3	DFHZA2436	3 10 11
X'98'	TCZACT01	DFHZA2439	2 18
X'99'	TCZSDSE7	DFHZA2459	3 9 11
X'9A'	TCZDOMCF	DFHZA2447	3 9 10 11 23

Table 136 (Page 5 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'9B'	TCZRACNL	DFHZC2486	3
X'9D'	TCZRSPER	DFHZC3465	1 2 3 9 10 11 23
X'9E'	TCZDEVND	DFHZC3472	None
X'A0'	TCZNOISC	DFHZC3480	23 24
X'A1'	TCZRVSZ2	DFHZC2438	3 10 11
X'A2'	TCZPRGE	DFHZC4945	3 9 10 11 24
X'A3'	TCZBKTSE	DFHZC2444	2 3 9 10 11 24
X'A7'	TCZBOEB	DFHZC2449	2 3 11 18 22 24
X'A8'	TCZFMHLE	DFHZC2471	2 3 4 10 11 22 24
X'A9'	TCZRACRF	DFHZC2472	11
X'AA'	TCZSDSE9	DFHZC2473	3 9 11
X'AB'	TCZLUERR	DFHZC3470	9 10 11 24
X'AC'	TCZVRDAC	DFHZC3474	9 10 11 24
X'AD'	TCZNRLUF	DFHZC3475	9 10 11 24
X'AE'	TCZRCLUF	DFHZC3476	9 10 11 24
X'AF'	TCZCLEAN	DFHZC3477	9 10 11 24
X'BA'	TCZRACSF	DFHZC2114	11
X'B0'	TCZEXRO	DFHZC3491	15 24
X'B1'	TCZRPLAC	DFHZC2401	2 3 9 10 11 23 24
X'B2'	TCZSDAUC	DFHZC2425	3 9 10 11 15 24
X'B3'	TCZBDBND	DFHZC4929	2 3 5 24
X'B4'	TCZRSNE	DFHZC2402	3 11
X'B5'	TCZSAXUC	DFHZC2420	2 3 9 10 11 23 24
X'B6'	TCZNSEED	DFHZC4924	2 3 5 24
X'B7'	TCZASINC	DFHZC4925	2 3 5 24
X'B8'	TCZEVBAD	DFHZC4926	2 3 5 24
X'B9'	TCZFMH12	DFHZC4927	2 3 5 24
X'BB'	TCZSEXUC	DFHZC2418	2 3 9 10 11 23 24
X'BC'	TCZINIIR	DFHZC3410	2 3 9 10 11
X'BD'	TCZDESGM	DFHZC4928	24
X'BE'	TCZBFAIL	DFHZC4944	2 3 5 24
X'BF'	TCZCPFAL	DFHZC3490	24
X'C0'	TCZDWEGF	DFHZC3499	None
X'C1'	TCZSRCAT	DFHZC2400	2 3 9 10 11 23 24
X'C2'	TCZLUINP	DFHZC3486	24
X'C3'	TCZCPFAL	DFHZC3490	24
X'C5'	TCZSRCNA	DFHZC2427	2

#

default actions of DFHZNAC

Table 136 (Page 6 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'C6'	TCZPASSD	DFHZA3484	None
X'C7'	TCZPSPRE	DFHZA3485	24
X'C8'	TCZLUIH	DFHZA3489	18 24
X'C9'	TCZNPSAU	DFHZA3487	24
X'CB'	TCZSRCTC	DFHZA2431	2 3 9 10 11
X'CC'	TCZSRCCI	DFHZA2451	2 3 9 10 11
X'CD'	TCZSRCCX	DFHZA2454	2 3 9 10 11
X'CE'	TCZVHOLD	DFHZA3469	9 10 11 24
X'CF'	TCZVRNOP	DFHZA3471	9 10 11 24
X'D0'	TCZTXCS	DFHZA2409	2 3 9 10 11 15 24
X'D1'	TCZTXCU	DFHZA2410	2 3 9 10 11 18 24
X'D3'	TCZDMPD	DFHZA2463	None
X'D4'	TCZCXRR	DFHZA2453	1 2 3 9 10
X'D5'	TCZCXE2	DFHZA2452	3 9 10 11 18 24
X'D6'	TCZSXC2	DFHZA2441	None
X'D7'	TCZSXC1	DFHZA2440	None
X'D8'	TCZRNCH	DFHZA2457	2 3 9 10 11 24
X'D9'	TCZYX43	DFHZA2469	2 3 9 10 11
X'DA'	TCZSXC3	DFHZA2470	9 10 11 24
X'DB'	TCZPIPL	DFHZA2117	9 10 11 23 24
X'DC'	TCZPX1	DFHZA2442	None
X'DD'	TCZPX2	DFHZA2458	None
X'DF'	TCZDMGF	DFHZA3482	None
X'E0'	TCZDMSN	DFHZA2411	None
X'E1'	TCZDMRA	DFHZA2412	None
X'E2'	TCZDMCL	DFHZA2413	2
X'E3'	TCZCNCL	DFHZA2485	3 9 10 11
X'E4'	TCZAIER	DFHZA2433	None
X'E6'	TCZDMLG	DFHZA2404	None
X'E8'	TCZDMSLE	DFHZA3416	2 3
X'E9'	TCZSTIND	DFHZA2102	3
X'EA'	TCZSTLER	DFHZA3432	2 3
X'EB'	TCZSTRMH	DFHZA3428	3
X'EC'	TCZSTRMM	DFHZA3429	2 3
X'ED'	TCZSTON	DFHZA3430	2 3
X'EF'	TCZSTIN	DFHZA3431	2 3
X'F1'	TCZBDMOD	DFHZA4931	18 24

Table 136 (Page 7 of 7). Messages issued and flags set by DFHZNAC for specific error codes

Error code	Symbolic label	Message	Action flags set
X'F2'	TCZEXRVT	DFHZA2469	2 3 9 10 11
X'F3'	TCZICTYP	DFHZA4932	2 3 24
X'F4'	TCZIDBA	DFHZA4933	2 3 24
X'F5'	TCZISYNL	DFHZA4934	2 3 24
X'F6'	TCZIUOW	DFHZA4935	2 3 24
X'F7'	TCZIFMHL	DFHZA4936	2 3 24
X'F8'	TCZFSMRB	DFHZA4943	3 9 10 11 24
X'F9'	TCZINVAT	DFHZA4946	2 3 24
X'FA'	TCZLUSEC	DFHZA4947	2 3 24
X'FB'	TCZPSUNB	DFHZA0125	None
X'FC'	TCZPSOPN	DFHZA0131	None
X'FD'	TCZPSRC	DFHZA0146	None
X'FE'	TCZPSRF	DFHZA0150	3 6 9 10 11 15 24
X'FF'	TCZPSPE	DFHZA0149	None

Notes:

1. See message DFHZA2497 or DFHZA3493, depending on the device type.
2. "Good morning" message to be sent.
3. Cancel task, and close VTAM session owing to quick close or abend.

CICS messages associated with VTAM errors

Table 137 (Page 1 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZA0101	TCZPRDIO	X'5D'	None
DFHZA0125	TCZPSUNB	X'FB'	None
DFHZA0131	TCZPSOPN	X'FC'	None
DFHZA0144	TCZPSRAC	X'7A'	11
DFHZA0145	TCZPSRAF	X'79'	3 6 9 10 11 24
DFHZA0146	TCZPSRC	X'FD'	None
DFHZA0147	TCZPSLE	X'71'	3 6 9 10 11 24
DFHZA0148	TCZPSVF	X'72'	9 10 11 24
DFHZA0149	TCZPSPE	X'FF'	None
DFHZA0150	TCZPSRF	X'FE'	3 6 9 10 11 15 24
DFHZA0155	TCZPSAF	X'6C'	3 6 9 10 11 24
DFHZA0156	TCZPSAR	X'6D'	None
DFHZA2102	TCZSTIND	X'E9'	3
DFHZA2114	TCZRACSF	X'BA'	11

Table 137 (Page 2 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZC2117	TCZPIPL	X'DB'	9 10 11 23 24
DFHZC2351	TCZSDAS	X'6F'	2 3
DFHZC2400	TCZSRCAT	X'C1'	2 3 9 10 11 23 24
DFHZC2401	TCZRPLAC	X'B1'	2 3 9 10 11 23 24
DFHZC2402	TCZRSNE	X'B4'	3 11
DFHZC2403	TCZSRCBF	X'11'	2 5 18 24
DFHZC2404	TCZLRCER	X'14'	2 3 9 10 11 23 24
DFHZC2404	TCZLRCNR	X'18'	2 3 9 10 11 23 24
DFHZC2404	TCZDMLG	X'E6'	None
DFHZC2405	TCZSRCTU	X'10'	18
DFHZC2406	TCZSRCTS	X'19'	9 10 11 18
DFHZC2407	TCZSRCPF	X'15'	2 3 9 10 11 24
DFHZC2408	TCZSRCVE	X'1A'	2 3 9 10 11 24
DFHZC2408	TCZSRCV2	X'1E'	2 3 9 10 11 24
DFHZC2409	TCZTXCS	X'D0'	2 3 9 10 11 15 24
DFHZC2410	TCZTXCU	X'D1'	2 3 9 10 11 18 24
DFHZC2411	TCZDMSN	X'E0'	None
DFHZC2412	TCZDMRA	X'E1'	None
DFHZC2413	TCZDMCL	X'E2'	2
DFHZC2414	TCZSRCSP	X'80'	None
DFHZC2416	TCZSRCVH	X'13'	18 24
DFHZC2417	TCZSRCVI	X'1D'	2 24
DFHZC2417	TCZVTAMI	X'20'	None
DFHZC2418	TCZSEXUC	X'BB'	2 3 9 10 11 23 24
DFHZC2419	TCZSSXUC	X'82'	2 3 9 10 11 23 24
DFHZC2420	TCZSAXUC	X'B5'	2 3 9 10 11 23 24
DFHZC2421	TCZUNCMD	X'60'	2 3 9 10 11 24
DFHZC2422	TCZLGCER	X'90'	1 2 3 6 9 10 11 23 24
DFHZC2423	TCZSDSE5	X'74'	3 9 10 11 24
DFHZC2424	TCZSESE1	X'75'	3 9 10 11 15 24
DFHZC2425	TCZSDAUC	X'B2'	3 9 10 11 15 24
DFHZC2426	TCZRACES	X'94'	2 3 9 10 11 22
DFHZC2427	TCZSRCNA	X'C5'	2
DFHZC2428	TCZSDSE6	X'92'	3 9 11
DFHZC2429	TCZRSTLE	X'91'	3 10 11
DFHZC2430	TCZSDRE2	X'78'	3 9 11 22
DFHZC2431	TCZSRCTC	X'CB'	2 3 9 10 11
DFHZC2432	TCZSSXNR	X'81'	None

Table 137 (Page 3 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZC2433	TCZAIER	X'E4'	None
DFHZC2434	TCZCPYNS	X'43'	3 11
DFHZC2435	TCZRVSZ1	X'96'	3 10 11 24
DFHZC2436	TCZRVSZ3	X'97'	3 10 11
DFHZC2437	TCZSDSE4	X'73'	3 9 11
DFHZC2438	TCZRVSZ2	X'A1'	3 10 11
DFHZC2439	TCZACT01	X'98'	2 18
DFHZC2440	TCZSXC1	X'D7'	None
DFHZC2441	TCZSXC2	X'D6'	None
DFHZC2442	TCZPX1	X'DC'	None
DFHZC2443	TCZROCT	X'59'	2 3 9 10 11 24
DFHZC2444	TCZBKTSE	X'A3'	2 3 9 10 11 24
DFHZC2445	TCZSDSE8	X'95'	3 9 11
DFHZC2446	TCZSSXIB	X'84'	2 3 9 10 11 23 24
DFHZC2447	TCZDOMCF	X'9A'	3 9 10 11 23
DFHZC2448	TCZINVRR	X'65'	2 3 10 11 22 23 24
DFHZC2449	TCZBOEB	X'A7'	2 3 11 18 22 24
DFHZC2450	TCZSSXAR	X'83'	None
DFHZC2451	TCZSRCCI	X'CC'	2 3 9 10 11
DFHZC2452	TCZCXE2	X'D5'	3 9 10 11 18 24
DFHZC2453	TCZCXRR	X'D4'	1 2 3 9 10
DFHZC2454	TCZSRCCX	X'CD'	2 3 9 10 11
DFHZC2455	TCZRACET	X'93'	2 3 9 10 11
DFHZC2456	TCZSRCDE	X'44'	2 3 9 10 11 24
DFHZC2457	TCZRNCH	X'D8'	2 3 9 10 11 24
DFHZC2458	TCZPX2	X'DD'	None
DFHZC2459	TCZSDSE7	X'99'	3 9 11
DFHZC2463	TCZDMPD	X'D3'	None
DFHZC2467	TCZLEXCI	X'88'	2 3 9 10 11 23 24
DFHZC2468	TCZLEXUS	X'89'	2 3 9 10 11 24
DFHZC2469	TCZYX43	X'D9'	2 3 9 10 11
DFHZC2469	TCZEXRVT	X'F2'	2 3 9 10 11
DFHZC2470	TCZSXC3	X'DA'	9 10 11 24
DFHZC2471	TCZFMHLE	X'A8'	2 3 4 10 11 22 24
DFHZC2472	TCZRACRF	X'A9'	11
DFHZC2473	TCZSDSE9	X'AA'	3 9 11
DFHZC2485	TCZCNCL	X'E3'	3 9 10 11
DFHZC2486	TCZRACNL	X'9B'	3

Table 137 (Page 4 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZC2487	TCZLGNA	X'76'	3
DFHZC2488	TCZDMRY	X'77'	None
DFHZC2489	TCZINCPY	X'40'	3 9 11
DFHZC2490	TCZTOLRQ	X'41'	2 3 9 10 11 15 24
DFHZC2497	TCZUNPRT	X'42'	None
DFHZC2498	TCZICPUT	X'4B'	None
DFHZC3400	TCZCHMX	X'45'	3 10 11 22
DFHZC3402	TCZOCIR	X'46'	3 9 10 11
DFHZC3409	TCZUNEGR	X'85'	2 3 9 10 11 23 24
DFHZC3410	TCZINIIR	X'BC'	2 3 9 10 11
DFHZC3416	TCZDMSLE	X'E8'	2 3
DFHZC3417	TCZSDRE3	X'50'	3 9 10 11 24
DFHZC3418	TCZBDPRI	X'51'	3 9 10 11 24
DFHZC3419	TCZBDUAC	X'52'	2 3 5
DFHZC3420	TCZBDTOS	X'53'	20
DFHZC3421	TCZSBIRV	X'5A'	20
DFHZC3422	TCZNSP01	X'5B'	2 3 9 10 11 24
DFHZC3424	TCZNSP02	X'5C'	9 10 11 15 24
DFHZC3428	TCZSTRMH	X'EB'	3
DFHZC3429	TCZSTRMM	X'EC'	2 3
DFHZC3430	TCZSTON	X'ED'	2 3
DFHZC3431	TCZSTIN	X'EF'	2 3
DFHZC3432	TCZSTLER	X'EA'	2 3
DFHZC3433	TCZERMGR	X'58'	None
DFHZC3434	TCZUNBIS	X'54'	2 3 9 10 11 24
DFHZC3440	TCZEMWBK	X'55'	None
DFHZC3441	TCZVTAMO	X'63'	None
DFHZC3442	TCZVTAMK	X'67'	None
DFHZC3443	TCZVTAMA	X'64'	None
DFHZC3444	TCZTIOAE	X'6A'	1 2 3 9 10 11 24
DFHZC3452	TCZSIGR	X'66'	None
DFHZC3454	TCZBRUAC	X'5E'	2 3 5 18 24
DFHZC3455	TCZBDSQP	X'5F'	2 3 5 18 24
DFHZC3461	TCZOPSIN	X'48'	None
DFHZC3462	TCZCLSIN	X'49'	None
DFHZC3463	TCZOPACB	X'4A'	None
DFHZC3464	TCZRELIS	X'57'	20
DFHZC3465	TCZRSPER	X'9D'	1 2 3 9 10 11 23

Table 137 (Page 5 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZC3466	TCZSEXOS	X'69'	20 23
DFHZC3468	TCZCLRRV	X'70'	9 10 11 15 24
DFHZC3469	TCZVHOLD	X'CE'	9 10 11 24
DFHZC3470	TCZXSHRA	X'37'	9 10 11 24 and sometimes 15
DFHZC3470	TCZLUERR	X'AB'	9 10 11 24
DFHZC3471	TCZVRNOP	X'CF'	9 10 11 24
DFHZC3472	TCZDEVND	X'9E'	None
DFHZC3473	TCZSLSRL	X'4D'	None
DFHZC3474	TCZVRDAC	X'AC'	9 10 11 24
DFHZC3475	TCZNRLUF	X'AD'	9 10 11 24
DFHZC3476	TCZRCLUF	X'AE'	9 10 11 24
DFHZC3477	TCZCLEAN	X'AF'	9 10 11 24
DFHZC3479	TCZUNBFE	X'4E'	2 3 9 10 11 24
DFHZC3480	TCZNOISC	X'A0'	23 24
DFHZC3481	TCZDSPCL	X'4C'	2 3 9 10 11 24
DFHZC3482	TCZDMGF	X'DF'	None
DFHZC3484	TCZPASSD	X'C6'	None
DFHZC3485	TCZPSPRE	X'C7'	24
DFHZC3486	TCZLUINP	X'C2'	24
DFHZC3487	TCZNPSAU	X'C9'	24
DFHZC3488	TCZXUVAR	X'3C'	2 3 9 10 11 24
DFHZC3489	TCZLUINH	X'C8'	18 24
DFHZC3490	TCZCPFAL	X'C3'	24
DFHZC3491	TCZEXRO	X'B0'	15 24
DFHZC3492	TCZDMIT	X'16'	None
DFHZC3495	TCZNOTNA	X'6B'	24
DFHZC3499	TCZDWEGF	X'C0'	None
DFHZC4902	TCZLUCF1	X'21'	3 9 10 11 24
DFHZC4903	TCZLUCF2	X'22'	3 9 10 11 24
DFHZC4904	TCZFSMBE	X'23'	3 10 11 9 24
DFHZC4905	TCZFSMCS	X'24'	3 10 11 9 24
DFHZC4906	TCZFSMCR	X'25'	3 10 11 9 24
DFHZC4907	TCZSDLER	X'26'	3 10 11 9 24
DFHZC4909	TCZRVLER	X'28'	3 10 11 9 24
DFHZC4910	TCZRVLRB	X'29'	3 10 11 9 24
DFHZC4911	TCZRLPEX	X'2A'	2 3 9 10 11 24
DFHZC4912	TCZRLPBD	X'2B'	2 3 9 10 11 24
DFHZC4913	TCZRLPDR	X'2C'	2 3 9 10 11 24

<i>Table 137 (Page 6 of 7). CICS messages associated with VTAM errors</i>			
Message	Symbolic label	Error code	Action flags set
DFHZC4914	TCZRLPIL	X'2D'	2 3 9 10 11 24
DFHZC4915	TCZRLPEC	X'2E'	2 3 9 10 11 24
DFHZC4916	TCZRLPRR	X'2F'	2 3 9 10 11 24
DFHZC4917	TCZRLPIF	X'30'	2 3 9 10 11 24
DFHZC4918	TCZRLPIR	X'31'	2 3 9 10 11 24
DFHZC4919	TCZIVIND	X'33'	2 3 9 10 11 24
DFHZC4920	TCZIVDAT	X'34'	2 3 9 10 11 24
DFHZC4922	TCZRLXCL	X'32'	20
DFHZC4924	TCZNSEED	X'B6'	2 3 5 24
DFHZC4925	TCZASINC	X'B7'	2 3 5 24
DFHZC4926	TCZEVBAD	X'B8'	2 3 5 24
DFHZC4927	TCZFMH12	X'B9'	2 3 5 24
DFHZC4928	TCZDESGM	X'BD'	24
DFHZC4929	TCZBDBND	X'B3'	2 3 5 24
DFHZC4930	TCZRTMT	X'35'	2 3 9 10 11 24
DFHZC4931	TCZBDMOD	X'F1'	18 24
DFHZC4932	TCZICTYP	X'F3'	2 3 24
DFHZC4933	TCZIDBA	X'F4'	2 3 24
DFHZC4934	TCZISYNL	X'F5'	2 3 24
DFHZC4935	TCZIUOW	X'F6'	2 3 24
DFHZC4936	TCZIFMHL	X'F7'	2 3 24
DFHZC4937	TCZLUSRR	X'8A'	2 3 5 24
DFHZC4938	TCZLUSRF	X'8B'	2 3 5 24
DFHZC4939	TCZLUPUN	X'8C'	2 3 5 24
DFHZC4940	TCZLUSKN	X'8F'	2 3 5 24
DFHZC4941	TCZLUPLK	X'8D'	2 3 5 24
DFHZC4942	TCZLUPEX	X'8E'	2 3 5 24
DFHZC4943	TCZFMRB	X'F8'	3 10 11 9 24
DFHZC4944	TCZBFAIL	X'BE'	2 3 5 24
DFHZC4945	TCZPRGE	X'A2'	3 9 10 11 24
DFHZC4946	TCZINVAT	X'F9'	2 3 24
DFHZC4947	TCZLUSEC	X'FA'	2 3 24
DFHZC4949	TCZRABUS	X'7D'	2 3 9 10 11 24
DFHZC6590	TCZXRST	X'3F'	None
DFHZC6591	TCZXERR	X'3E'	9 10 11 15 24
DFHZC6593	TCZXSBC	X'3B'	None
DFHZC6594	TCZXSHR	X'3A'	24
DFHZC6595	TCZXSABN	X'39'	2 3 5 24

#

Table 137 (Page 7 of 7). CICS messages associated with VTAM errors

Message	Symbolic label	Error code	Action flags set
DFHZC6596	TCZXSWAS	X'38'	2 3 15 24
DFHZC6598	TCZXRFVS	X'56'	None

Default actions for system sense codes

Table 138 shows the message issued and action flags set by DFHZNAC for each inbound system sense code received. See the *CICS/ESA Messages and Codes* manual for a description of the conditions that correspond to the system sense codes. The figures in the “**Action flags set**” column are translated into bit settings and explained in Table 139 on page 711.

Table 138 (Page 1 of 2). Messages issued and flags set by DFHZNAC for specific sense codes

Sense code	Message	Action flags set
X'0001' 1	DFHZC3401	2
X'0002' 1	DFHZC3415	2, 3, 10, 11
X'0003' 1	DFHZC3449	None
X'0004' 1	DFHZC3450	None
X'0007' 1	DFHZC3451	None 2
X'00FF'	DFHZC3446	2, 3, 9, 10, 11, 23, 24
X'0801'	DFHZC2476	3, 9, 10, 11
X'0802'	DFHZC2461	None
X'0806'	DFHZC3426	None
X'0807'	DFHZC3411	None
X'080B'	DFHZC2462	2, 3, 9, 10, 11, 15, 24
X'080E'	DFHZC3448	23
X'080F'	DFHZC3436	9, 10, 11
X'0811'	DFHZC2464	9, 10, 11
X'0812'	DFHZC2465	2, 3
X'0814'	DFHZC3447	2, 3, 9, 10, 11, 24
X'081B'	DFHZC2483	2, 3 3
X'081C'	DFHZC2466	2, 3, 9, 10, 11
X'0824'	DFHZC2475	3, 9, 10, 11
X'0825'	DFHZC2484	2, 3, 9, 10, 11
X'0826'	DFHZC3423	2, 3, 9, 10, 11
X'0827'	DFHZC2480	3
X'0829'	DFHZC3407	1, 2, 3, 10, 11, 24
X'082A'	None 4	9
X'082B'	DFHZC3408	2, 3, 10, 11, 13

default actions for system sense codes

<i>Table 138 (Page 2 of 2). Messages issued and flags set by DFHZNAC for specific sense codes</i>		
Sense code	Message	Action flags set
X'082D'	DFHZC3413	None
X'082E'	DFHZC3412	None
X'082F'	DFHZC3414	2, 3, 9, 10, 11
X'0831'	DFHZC3438	None
X'0833'	DFHZC3427	None
X'0847'	DFHZC3439	None
X'084A'	None ⁵	None
X'084C'	DFHZC3467	9, 10, 11
X'0860'	DFHZC3459	None
X'0863'	DFHZC3460	9, 10, 11
X'0864'	DFHZC2475	3, 9, 10, 11
X'0865'	DFHZC2465	3, 9, 10, 11
X'0866'	DFHZC2475	3, 9, 10, 11
X'0867'	None ⁶	9, 10, 11
X'0868'	DFHZC3456	2, 9, 10, 11
X'0869'	DFHZC3457	2, 9, 10, 11
X'08FF'	DFHZC3447	2, 3, 9, 10, 11, 24
X'1000'	DFHZC3494	2, 3, 9, 10, 11
X'1001'	DFHZC2481	2, 3, 9, 10, 11, 14
X'1002'	DFHZC2481	2, 3, 9, 10, 11, 14
X'1003'	DFHZC2479	2, 3, 9, 10, 11, 14
X'1005'	DFHZC3406	2, 3, 4, 9, 10, 11, 14
X'1008'	DFHZC2478	None
X'1009'	DFHZC3458	2, 9, 10, 11
X'10FF'	DFHZC3446	2, 3, 9, 10, 11, 23, 24
X'2003'	DFHZC3405	2, 3, 9, 10, 11, 15, 24
X'20FF'	DFHZC3445	2, 3, 9, 10, 11, 23, 24
X'400B'	DFHZC2477	1, 3, 11
X'40FF'	DFHZC3453	2, 3, 9, 10, 11, 23, 24
X'8000'	DFHZC3435	2, 3, 9, 10, 11, 18, 24
X'8005'	DFHZC3435	2, 3, 9, 10, 11, 18, 24
X'80FF'	DFHZC3435	2, 3, 9, 10, 11, 18, 23, 24
X'FFFF'	DFHZC2460	2, 3, 9, 10, 11, 23, 24

Notes:

1. The system sense code is in the form of an LUSTATUS command code.
2. No action flags are set if a task is attached or if outstanding operations are to complete. Otherwise, flag 21 is set.
3. Action flags 2 and 3 are set for negative response received for a SEND that requested a definite response.
4. Presentation space error.
5. Presentation error on read. Display buffer alteration, due to operator intervention, detected on a READ command to a compatibility-mode logical unit.
6. Function abend received from a device. A negative response to a chain was sent, but purged.

Action flag settings and meanings

Table 139 shows the “action flags” that can be set by DFHZNAC in the communication area passed to DFHZNEP. The flags set by DFHZNAC represent the default actions that will be taken if the settings are not changed by DFHZNEP.

The figures in the “**Flag**” column refer to those in columns 3 of Table 136 on page 697 and Table 138 on page 709.

Table 139. Meanings of action flags set by DFHZNAC

Flag	Field	Bit mask	Hex bit setting	Action
1	TWAOPT1	1... ..	X'80'	Print action flags
2		.1.. ..	X'40'	Print VTAM RPL
3		..1.	X'20'	Print TCTTE
4		...1	X'10'	Print TIOA
5	 1...	X'08'	Print BIND area
6	1..	X'04'	System dump if no task attached
9	TWAOPT2	1... ..	X'80'	Abort any send for this terminal
10		.1.. ..	X'40'	Abort any receive for this terminal
11		..1.	X'20'	Abend any task attached to TCTTE
12		...1	X'10'	Cancel any task attached to TCTTE
13	 1...	X'08'	Good Morning message to be sent
141..	X'04'	Purge any BMS pages for this TCTTE	
151.	X'02'	SIMLOGON required	
17	TWAOPT3	1... ..	X'80'	Set INTLOG now allowed
18		.1.. ..	X'40'	Set no internal general logons
20		...1	X'10'	Normal CLSDST (no reset allowed)
21	 1...	X'08'	Normal CLSDST (reset allowed)
221..	X'04'	Send negative response	
231.	X'02'	AOS - keep node out of service	
241	X'01'	CLSDST node	

#

action flag settings and meanings

Appendix C. Transient data write-to-terminal program (DFH\$TDWT)

DFH\$TDWT is a sample program that sends transient data messages to a terminal or printer. You can use it to send messages from a single transient data queue, or from several queues, to one terminal.

In the destination control table (DCT), you can specify that particular categories of message (for example, those from the abnormal condition program (DFHACP), and signon and sign-off messages) should be sent to destinations defined as INDIRECT. If these INDIRECT destinations are defined (by means of the INDDDEST operand) so that they refer to the same intrapartition destination with a transaction identifier and a trigger level of 1, the receipt of a single message in any of the specified categories causes the transaction to be started. The program thus invoked displays or prints the message. The transaction that invokes the DFH\$TDWT sample program is TDWT.

To use the sample program, your CICS system must include automatic transaction initiation and an intrapartition transient data set. The source code for the DFH\$TDWT sample program is provided in CICS410SDFHSAMP, and the object code is provided in CICS410SDFHLOAD.

For detailed information about defining transient data destinations in the DCT, see the *CICS/ESA Resource Definition Guide*.

Resource definitions required

To use the DFH\$TDWT sample program as supplied, you need the following resource definitions installed on your CICS region:

- A CSD program definition for the DFH\$TDWT program
- A CSD transaction definition for the TDWT transaction
- A CSD terminal definition for the L86P terminal
- A DCT entry for the intrapartition destination L86P
- DCT entries for the intrapartition destinations indirect to the destination L86P.

The CSD resource definitions for the program and transaction are provided in the DFH\$UTILS group, and installed in the CSD when you install CICS with the default group list DFHLIST (or your own group list after adding those resource definitions to it).

The DCT entry for the intrapartition destination L86P is provided in the sample DCT, DFHDCT2\$.

However, you must define the other resources:

- Add to the CSD, a terminal definition for the L86P terminal and install it in your CICS region.
- Add to the DCT, entries for the intrapartition destinations that are indirect to the destination L86P and install the DCT in your CICS region.

transient data write-to-terminal program

For example:

```
DFHDCT TYPE=INDIRECT,DESTID=CSMT,INDDEST=L86P  
DFHDCT TYPE=INDIRECT,DESTID=CSTL,INDDEST=L86P  
DFHDCT TYPE=INDIRECT,DESTID=CSML,INDDEST=L86P
```

Appendix D. Uppercase translation of national characters

Uppercase translation in CICS can be done either by using the UCTRAN option on the PROFILE and TYPETERM definitions, or by using the EXEC CICS SET TERMINAL(termid) UCTRANST command.

However, some languages have characters which are not part of the set of EBCDIC characters translated by UCTRAN, and so are never translated to uppercase, regardless of what you have specified on your resource definitions. To translate these national characters, you have two options:

- Use the XZCIN exit
- Modify the translation table in DFHTCTDY.

Whichever method you use, the *Character Data Representation Architecture Level 1 - Registry* manual, SC09-1391-00, is a useful reference for information on code pages.

Using the XZCIN exit

XZCIN is described on page 121. To use it for uppercase translation, you must supply your own translation routine, which is then invoked when terminal input occurs.

Using DFHTCTDY

CICS provides the source for DFHTCTDY in CICS410.SDFHSAMP; you can use this to modify the translation table to translate your national characters. Figure 84 on page 716 shows a suggested way to code the assembler source statements used to generate DFHTCTDY.

uppercase translation of national characters

```
MACRO
NATLANG
DFHUCTRT CSECT          Resume UCTRAN table CSECT
.*
.* This example translates lowercase 'a' ( EBCDIC X'81') to
.* uppercase 'A' (EBCDIC X'C1') for a US code page.
.*
      ORG TCZUCTAB+X'81'          Reset the counter to the
                                character to be translated.
      DC  X'C1'                  Declare the replacement
                                character as a constant.
.*
.* Repeat the above two statements for each extra character you wish
.* to be translated.
.*
      ORG ,                      Reset the location counter
&SYSLOC LOCTR                Resume previous location counter
MEND                          End of macro definition
DFHTCT TYPE=INITIAL,SUFFIX=DY, *
      MIGRATE=COMPLETE,         *
      ACCMETH=(VTAM),          *
      DUMMY=DUMMY
      NATLANG                    Execute NATLANG
DFHTCT TYPE=FINAL
END DFHTCTBA
```

Figure 84. Suggested coding for national language character translation

If you use this method, you must reassemble your modified copy of DFHTCTDY, keeping the 'DY' suffix. The 'DY' suffix is necessary because, even if you use RDO for all your terminals and have TCT=NO specified in your SIT or its overrides, CICS uses DFHTCTDY by default to create control blocks for RDO-defined and autoinstalled terminals.

If you specify a suffix other than 'DY' on the SIT TCT parameter, you must add
your translation code to the TCT you are using.

Appendix E. The example program for the XTSEREQ global user exit, DFH\$XTSE

This appendix lists the example global user exit program, DFH\$XTSE. The example shows you how to:

- Use EXEC CICS commands in a global user exit program
- Use EXEC CICS commands and XPI calls in the same exit program
- Modify the command parameter list in EXEC interface exits such as XTSEREQ and XICEREQ
- Modify Temporary Storage (TS) requests.

```

*****
*                                                                 *
* MODULE NAME = DFH$XTSE                                         *
*                                                                 *
* FUNCTION =                                                     *
*   Example global user exit program to run at the XTSEREQ and   *
*   XTSEREQC exits.                                             *
*                                                                 *
* DESCRIPTION =                                                 *
*   The program gives examples of:                               *
*   1) Coding Exec Interface global user exits, showing how to  *
*       modify and add parameters to the Command Parameter List. *
*   2) Issuing a mixture of EXEC CICS API and XPI calls within  *
*       the same global user exit program.                       *
*   3) Modifying Temporary Storage requests, by renaming the queue *
*       name and allowing the SYSID to be added so that the request *
*       is routed to a queue-owning region (QOR).                *
*                                                                 *
* ----- *
*   NOTE that this program is only intended to DEMONSTRATE the use *
*   of the TS request user exit XTSEREQ, and to show the sort of *
*   information which can be obtained from the exit parameter list. *
*   IT SHOULD BE TAILORED BEFORE BEING USED IN A PRODUCTION ENVIRONMENT. *
* ----- *
*                                                                 *
* NOTES =                                                       *
*   The important notes to remember when coding similar global user *
*   exits are:                                                  *
*                                                                 *
*   1) If the exit program modifies the Command Parameter List, you *
*       MUST ensure that the storage used for additional fields such *
*       as the SYSID is non-volatile. Here are examples of storage *
*       that is safe:                                           *
*       a) Shared storage obtained by GETMAIN. This should be *
*           obtained in the Request exit, and freed in the Request *
*           Complete exit.. The shared storage address can be passed *
*           using the 4-byte token in the DFHUEPAR parameter list. *
*       b) Shared global work area storage.                      *
*       c) Storage obtained by using the LOAD HOLD option.      *
*       d) TCTUA or CWA storage.                                 *

```

Figure 85 (Part 1 of 16). Example exit program for the XTSEREQ exit

example XTSERREQ global user exit program

```

*
*      It is not safe to use the following storage:
*      Program storage (DFHEISTG) since this is freed as soon
*      as the exit program returns control to CICS.
*
* 2) When adding or removing a field in the command parameter list,
* you must remember:
*   a) To set/clear the field's existence bit in the EID
*   b) To set/clear the appropriate address in the Addr_List
*   c) To set the hi-order bit in the LAST address in the
*       Addr_List.
*
* 3) If you are planning to use the CICS API in the exit, you
* must:
*   a) Use the DFHEIENT macro to control module entry.
*   b) Use the DFHEIRET macro to return control to CICS. However,
*       the exit return code MUST be set in Register 15.
*   c) Issue an ADDRESS EIB command before issuing any EXEC CICS
*       commands.
*
* 4) If you are planning to use the API and XPI in the same
* global user exit program, take care to ensure that Register
* 13 points to the kernel stack entry (UEPSTACK) for XPI calls,
* and is restored for API calls if necessary.
*
*
*
*****
      EJECT ,
*-----*
*
* Copybook and DSECTS required by the exit program
*
*-----*
      DFHUEXIT TYPE=EP,ID=(XTSEREQ,XTSEREQC)
      DFHUEXIT TYPE=XPIENV      Exit programming interface (XPI)
      COPY  DFHTRPTY           Trace XPI definitions
      COPY  DFHTSUED           Command Level Plist definitions
*
*-----*
* The following DSECT maps the shared storage obtained by the
* EXEC CICS GETMAIN API call. This storage is used to store the
* modified SYSID and/or TS QNAME that is passed to CICS on return
* from the exit program.
*-----*
      SHARED_STORAGE      DSECT
      SHARED_EYECATCHER  DS CL16
      SHARED_NAME        DS CL8
      SHARED_SYSID       DS CL4
*

```

Figure 85 (Part 2 of 16). Example exit program for the XTSERREQ exit

```

*-----*
* The TS Routing table is made up of a set of entries. Each entry *
* can be mapped by the TABLE_ENTRY DSECT *
*-----*
TABLE_ENTRY DSECT
ENTRY_NAME DS CL8
NEW_NAME DS CL8
NEW_SYSID DS CL4
ENTRY_ACTION DS XL1
FILLER DS CL3
*
*-----*
* The following definitions are for program working storage. *
*-----*
DFHEISTG DSECT
RETCODE DS XL4 Program Return Code
TR_ERROR_N DS X Error Number for Trace Entry
RESP DS X API Response
EJECT ,
*****
* PROGRAM REGISTER USAGE : *
* R0 - Work Register *
* R1 - Points to DFHUEPAR plist on entry *
* Work Register *
* R2 - DFHUEPAR parameter List *
* R3 - Code Base Register *
* R4 - <unused> *
* R5 - <unused> *
* R6 - Subroutine Linkage Register *
* R7 - Address of TS Queue Name from Command Plist *
* R8 - Command Parameter list UEPCPLS *
* R9 - Address of Table_Entry in TS_Routing_Table *
* R10- <unused> *
* R11- EIB Register *
* R12- Work Register *
* R13- DFHEISTG for API calls *
* Kernel Stack for XPI calls *
* R14- Work Register *
* R15- Work Register *
*****
EJECT ,
*****
* DFH$XTSE - Main Routine *
* This is the entry point for the exit program. Control is passed *
* to the TS_REQUEST or TS_REQUEST_COMPLETE routines depending *
* on whether the exit was invoked at the XTSERREQ or XTSEREQC exit *
* points *
*

```

Figure 85 (Part 3 of 16). Example exit program for the XTSERREQ exit

example XTSEREQ global user exit program

```

* Registers:
* R1 = UEPAR plist (set on entry)
*   = Work register
* R2 = UEPAR plist
* R3 = Program base register (set by DFHEIENT)
* R6 = Linkage register
* R11= EIB register
* R13= EISTG register (set by DFHEIENT)
* R15= Work register
*       User Exit Return Code
*
* Logic:
* DFH$XTSE:
*   Exec Interface Entry
*   Address DFHUEPAR plist
*   Set OK Return Code
*   Address the EIB
*   Trace entry
*   Select Exitid
*     When(XTSEREQ) then call TS_Request
*     When(XTSEREQC) then call TS_Request_Complete
*     Otherwise call Error(Invalid_Exit)
*   End Select
*   Trace exit
*   Set Exit return code
*   Return
*****
DFH$XTSE DFHEIENT
DFH$XTSE AMODE 31
DFH$XTSE RMODE ANY
          LR   R2,R1           DFHUEPAR plist provided by caller
          USING DFHUEPAR,R2    Use R2 to address UEPAR PLIST
*
          LA   R15,UERCNORM    Set OK Response
          ST   R15,RETCODE     in working storage
*
          EXEC CICS ADDRESS EIB(R11)
          USING DFHEIBLK,R11
*
          BAL  R6,TRACE_ENTRY   Trace program entry
*
          L    R1,UEPEXN        Address of the 1 byte Exit Id
          CLI  0(R1),XTSEREQ    Is this XTSEREQ exit?
          BE   TS_REQUEST       ..Yes Branch to routine
          CLI  0(R1),XTSEREQC   Is this XTSEREQC exit?
          BE   TS_REQUEST_COMPLETE .. Yes Branch to routine
          B    ERROR1           Otherwise Branch to error routine
*
RETURN    DS   0H              Return point
          BAL  R6,TRACE_EXIT    Trace program exit
*
          L    R15,RETCODE      Fetch return code
          DFHEIRET RCREG=15     Return to CICS
          EJECT ,

```

Figure 85 (Part 4 of 16). Example exit program for the XTSEREQ exit


```

*-----*
* TS_REQUEST - Invoked at XTSERREQ exit point *
* Determine the TS Queue Name and scan the TS_Routing_Table for *
* a match. If an entry exists in the table, then check the action *
* field and call the ROUTE_REQUEST or LOCAL_REQUEST routines. *
* *
* The TS_Routing_Table is made up of entries with the following *
* structure: *
* *
* TABLE_ENTRY: *
* ----- *
* | Entry_Name | New_Name | QOR_Sysid | Action | *filler* | *
* | Char 8 | Char 8 | Char 4 | Bin 1 | Char 3 | *
* ----- *
* Last Entry is indicated by special TS_Queue Name *
* *
* Registers: *
* R1 = Work register *
* R7 = Set to the TS Queue Name *
* R8 = Command Parameter List (CLPS) *
* R9 = Points to the next entry in the TS_Routing_Table *
* R15= Work register *
* *
* Logic: *
* TS_Request: *
* If called recursively then *
* call Error(Recursive_Call1) *
* Else *
* If the Command GROUP code is not a TS request then *
* call Error(Invalid_Group_Code1) *
* Else *
* Clear the UEPTQTOK *
* Address the Command Plist UEPCPLS *
* Fetch tsq_name *
* Fetch start of TS_Routing_Table *
* Check_Next_Entry: *
* Get the next table entry *
* Select (entry_name) *
* When (last_entry) call Entry_Not_Found *
* When (tsq_name) *
* Select (entry_action) *
* When (Route) call Route_Request *
* When (Local) call Local_Request *
* Otherwise call Error(Invalid_Table_Action) *
* End Select *
* Otherwise *
* Goto Check_Next_Entry *
* End Select *
* End If *
* End If *
* Return *

```

Figure 85 (Part 5 of 16). Example exit program for the XTSERREQ exit

example XTSEREQ global user exit program

```

*-----*
TS_REQUEST DS 0H
*      Check for possible recursion
      L   R1,UEPRECUR      Address of recursive count
      LH  R1,0(R1)         Fetch count
      LTR R1,R1             Has exit been invoked recursively?
      BNZ ERROR2           ..Yes Branch to error routine
*
*      Extract pointer to the EID and TS queue name from CLPS
      L   R8,UEPCLPS       Fetch address of Command Plist
      USING TS_ADDR_LIST,R8 Use R8 to address CLPS
      L   R1,TS_ADDR0      Address the EID..
      L   R7,TS_ADDR1      Fetch address of TS QUEUE
      DROP R8              Drop addressability to CLPS
*
*      Check that the Command GROUP code corresponds to a TS request
      USING TS_EID,R1      ..with Register 1
      CLI TS_GROUP,TS_TEMPSTOR_GROUP Is this a TS request?
      BNE ERROR3           ..No Branch to error routine
      DROP R1              Drop addressability to EID
*
*      Clear the TS Request token
      L   R1,UEPTQOK       Fetch address of token
      XC  0(4,R1),0(R1)    Clear Token for XTSEREQC
*
*
*-----*
* Start scan of TS_Routing Table *
*-----*
      LA  R9,TS_ROUTING_TABLE Fetch address of routing table
      USING TABLE_ENTRY,R9   Address entries from R9
*
CHECK_NEXT_ENTRY DS 0H
      CLC ENTRY_NAME,ENTRY_NAME_LAST Is this the last entry
      BE  ENTRY_NOT_FOUND ..Yes Take default routing action
      CLC ENTRY_NAME,0(R7) Is this the wanted TS queue name?
      BE  ENTRY_FOUND ..Yes Check for the action required
      LA  R9,24(R9) Point to next entry
      B   CHECK_NEXT_ENTRY Start search again
*
ENTRY_FOUND DS 0H
      CLI ENTRY_ACTION,ROUTE Is the action to route request?
      BE  ROUTE_REQUEST ..Yes Branch to Route routine
      CLI ENTRY_ACTION,LOCAL Is the action to rename queue?
      BE  LOCAL_REQUEST ..Yes Branch to Local routine
      B   ERROR4 Otherwise Branch to error routine
      DROP R9 Drop addressability to Entry
      EJECT ,
*

```

Figure 85 (Part 6 of 16). Example exit program for the XTSEREQ exit

```

*-----*
* TS_REQUEST_COMPLETE - Invoked at XTSEREQC exit point *
*   Free any shared storage that was acquired during previous *
*   invocation at XTSEREQ *
* *
* Registers: *
*   R1 = Work register *
*   R6 = Linkage register *
*   R8 = Command Parameter List (CLPS) *
* *
* Logic: *
*   TS_Request_Complete: *
*   If called recursively then *
*     call Error(Recursive_Call2) *
*   Else *
*     If the Command GROUP code is not a TS request then *
*       call Error(Invalid_Group_Code2) *
*     Else *
*       If UEPTQTOK->token = 0 then Call Freemain_Shared_Plist *
*     End If *
*   End If *
*   Return *
*-----*
TS_REQUEST_COMPLETE DS 0H
*   Check for possible recursion
*   L   R1,UEPRECUR      Address of recursive count
*   LH  R1,0(R1)        Fetch count
*   LTR R1,R1           Has exit been invoked recursively?
*   BNZ ERROR5         ..Yes Branch to error routine
*
*   Check that the Command GROUP code corresponds to a TS request
*   L   R8,UEPCLPS      Fetch address of Command Plist
*   USING TS_ADDR_LIST,R8 Use R8 to address CLPS
*   L   R1,TS_ADDR0     Address the EID..
*   USING TS_EID,R1     ..with Register 1
*   CLI TS_GROUP,TS_TEMPSTOR_GROUP Is this a TS request?
*   BNE ERROR6         ..No Branch to error routine
*   DROP R1             Drop addressability to EID
*   DROP R8            Drop addressability to CLPS
*
*   L   R1,UEPTQTOK    Fetch address of Token
*   L   R1,0(R1)       Fetch actual token
*   LTR R1,R1          Did XTSEREQ GETMAIN any storage?
*   BZ  RETURN         ..No Return to caller
*   BAL R6,FREEMAIN_SHARED ..Yes Issue FREEMAIN
*   B   RETURN         Return to caller
*   EJECT ,
*

```

Figure 85 (Part 7 of 16). Example exit program for the XTSEREQ exit

example XTSEREQ global user exit program

```

=====
* LOCAL_REQUEST: Process Local TS Queues
* An entry has been found in the TS_Routing Table for this TS
* Queue Name. If required, rename the TS Queue Name, but do not
* modify the SYSID.
*
* Registers:
* R1 = Work register
* R6 = Link Register
* R7 = Address of current Queue name (Set on entry)
* R8 = Command Parameter List (CLPS)
* R9 = Address of table entry (Set on entry)
* R12= Work register (Shared_storage)
*
* Logic:
* Local_Request:
* If entry_name ^= new_name then
* Call Getmain_Shared
* Copy new_name into shared storage
* Address the command plist
* Update ADDR1 to point to address of the new TS QUEUE name
* Set the Hi-order bit if last address in CLPS
* End If
* Return
=====
LOCAL_REQUEST DS 0H
        USING TABLE_ENTRY,R9      R9 points to the table entry
        CLC NEW_NAME,0(R7)         Is the new_name=current_queue name?
        BE RETURN                  ..Yes Return

*
* Obtain Shared storage to hold the new queue name
        BAL R6,GETMAIN_SHARED      GETMAIN SHARED storage
        L R12,UEPTQOK              Fetch address of token
        L R12,0(R12)               Fetch shared storage pointer
        USING SHARED_STORAGE,R12   Address using R12
        MVC SHARED_NAME,NEW_NAME   Copy QNAME into shared storage

*
* Update the Queue Name in CLPS
        L R8,UEPCLPS               Address the CLPS.
        USING TS_ADDR_LIST,R8      ..with Register 8
        LA R1,SHARED_NAME          Fetch address of the new QNAME
        TM TS_ADDR1,X'80'          Is the hi-order bit on?
        BZ LOCAL1                  ..No continue
        O R1,=X'80000000'          Indicate ADDR1 is last parameter
LOCAL1  DS 0H
        ST R1,TS_ADDR1             Store address in TS_ADDR1
        B RETURN                   Return
        DROP R8                    Drop TS_ADDR_LIST
        DROP R12                   Drop SHARED_STORAGE
        DROP R9                    Drop addressability to Entry
        EJECT ,
*

```

Figure 85 (Part 8 of 16). Example exit program for the XTSEREQ exit

example XTSERREQ global user exit program

```

=====
* ROUTE_REQUEST: Ship request to remote system
* An entry has been found in the TS_Routing Table for this TS
* Queue Name. The request is modified by adding a SYSID to the
* command and renaming the queue if required.
*
* Registers:
* R1 = Work register
* R6 = Link Register
* R7 = Address of current Queue name (Set on entry)
* R8 = Command Parameter List (CLPS)
* R9 = Address of table entry (Set on entry)
* R12= Work register (Shared_storage)
*
* Logic:
* Route_Request:
* Call Getmain_Shared
* If entry_name ^= new_name then
* Copy new_name into shared storage
* Address the command plist
* Update ADDR1 to point to address of the new TS QUEUE name
* End If
* Copy new_sysid into shared storage
* Address the command plist
* Update ADDR7 to point to the address of the new SYSID
* Set the SYSID existence bit in the EID
* Set the Hi-order bit in last address in CLPS
* Return
=====
ROUTE_REQUEST DS 0H
        BAL  R6,GETMAIN_SHARED  GETMAIN SHARED storage
        L    R12,UEPTQOK        Fetch address of token
        L    R12,0(R12)         Fetch Shared storage address
        USING SHARED_STORAGE,R12 Address using R12

*
* Update the Queue Name in CLPS
        USING TABLE_ENTRY,R9    R9 points to the table entry
        CLC  NEW_NAME,0(R7)      Is the new_name=current_queue name?
        BE   ROUTE1              ..Yes No need to update Queue Name
        MVC  SHARED_NAME,NEW_NAME Copy QNAME into shared storage
        L    R8,UEPCLPS          Address the CLPS..
        USING TS_ADDR_LIST,R8    ..with Register 8
        LA  R1,SHARED_NAME      Fetch address of the new QNAME
        ST  R1,TS_ADDR1         Store address in TS_ADDR1
        DROP R8                  Drop TS_ADDR_LIST
*

```

Figure 85 (Part 9 of 16). Example exit program for the XTSERREQ exit

example XTSEREQ global user exit program

```

*      Update the Sysid in CLPS
ROUTE1 DS    0H
MVC    SHARED_SYSID,NEW_SYSID Copy SYSID into shared storage
L      R8,UEPCLPS           Address the CLPS..
USING  TS_ADDR_LIST,R8     ..with Register 8
L      R1,TS_ADDR0         Address the EID..
USING  TS_EID,R1           ..with Register 1
OI     TS_BITS1,TS_SYSID_V Indicate SYSID now present in CLPS
DROP   R1                  Drop addressability to EID
LA     R1,SHARED_SYSID     Fetch address of the new SYSID
ST     R1,TS_ADDR7         Store address in TS_ADDR7
OI     TS_ADDR7,X'80'      Indicate SYSID is end of plist

*
*      Clear hi-order bits in ARGs 1 to 5
NI     TS_ADDR1,X'7F'      Indicate not last parameter in CLPS
NI     TS_ADDR2,X'7F'      Indicate not last parameter in CLPS
NI     TS_ADDR3,X'7F'      Indicate not last parameter in CLPS
NI     TS_ADDR4,X'7F'      Indicate not last parameter in CLPS
NI     TS_ADDR5,X'7F'      Indicate not last parameter in CLPS
B      RETURN              Return
DROP   R8                  Drop TS_ADDR_LIST
DROP   R12                 Drop SHARED_STORAGE
DROP   R9                  Drop addressability to Entry
EJECT ,

*
*=====*
* ENTRY_NOT_FOUND - No entry was found in the TS_Routing_Table *
* No entry found in Routing Table for this TS Queue Name. In the *
* sample program, all such requests are routed. *
* * *
* Registers: *
* R1 = Work register *
* R6 = Link Register *
* R8 = Command Parameter List (CLPS) *
* R12= Work register (Shared_storage) *
* *

```

Figure 85 (Part 10 of 16). Example exit program for the XTSEREQ exit

```

* Logic:
*   Entry_Not_Found:
*     Call Getmain_Shared
*     Copy default_sysid into shared storage
*     Address the command plist
*     Update ADDR7 to point to the address of the default SYSID
*     Set the SYSID existence bit in the EID
*     Set the Hi-order bit in last address in CLPS
*   Return
*=====*
ENTRY_NOT_FOUND DS 0H
      BAL  R6,GETMAIN_SHARED  GETMAIN SHARED storage
      L    R12,UEPTQTOK      Fetch address of token
      L    R12,0(R12)        Fetch shared storage address
      USING SHARED_STORAGE,R12 Address using R12
*
*   Update the Sysid in CLPS
MVC  SHARED_SYSID,DEFAULT_SYSID Copy SYSID to shared storage
L    R8,UEPCLPS             Address the CLPS..
USING TS_ADDR_LIST,R8      ..with Register 8
L    R1,TS_ADDR0            Address the EID..
USING TS_EID,R1            ..with Register 1
OI   TS_BITS1,TS_SYSID_V   Indicate SYSID now present in CLPS
DROP R1                     Drop addressability to EID
LA   R1,SHARED_SYSID       Fetch address of the new SYSID
ST   R1,TS_ADDR7           Store address in TS_ADDR7
OI   TS_ADDR7,X'80'        Indicate SYSID is end of plist
*
*   Clear hi-order bits in ARGs 1 to 5
+
+
+
+
+
NI   TS_ADDR1,X'7F'        Indicate not last parameter in CLPS
NI   TS_ADDR2,X'7F'        Indicate not last parameter in CLPS
NI   TS_ADDR3,X'7F'        Indicate not last parameter in CLPS
NI   TS_ADDR4,X'7F'        Indicate not last parameter in CLPS
NI   TS_ADDR5,X'7F'        Indicate not last parameter in CLPS
B    RETURN                Return
DROP R8                     Drop TS_ADDR_LIST
DROP R12                     Drop SHARED_STORAGE
EJECT ,
*

```

Figure 85 (Part 11 of 16). Example exit program for the XTSEREQ exit

example XTSEREQ global user exit program

```

=====
* GETMAIN_SHARED - Obtain Shared storage *
* We cannot use transaction storage to pass information in the *
* Command Parameter List since this is volatile and will be *
* released when the exit program returns to CICS. *
* We must obtain SHARED storage here, and free it at the *
* TS request complete exit XTSEREQC *
* *
* Registers: *
* R0 = Used by EXEC CICS call *
* R1 = Used by EXEC CICS call *
* Work Register *
* R6 = Link Register - Return Address *
* R11= EIB register (set on entry) *
* R12= Work register *
* R14= Used by EXEC CICS call *
* R15= Used by EXEC CICS call *
* *
* Logic: *
* Getmain_Shared: *
* EXEC CICS GETMAIN LENGTH(32) SET(UEPTQTOK) SHARED RESP(resp) *
* If resp = OK then *
* Call Error(Getmain_Failed) *
* Else *
* Address shared storage *
* Set eyecatcher 'XTSEREQ Storage' *
* End If *
* Return *
=====
GETMAIN_SHARED DS 0H
    L    R12,UEPTQTOK          Fetch address of token
    L    R12,0(R12)           Fetch shared storage anchor
    LTR  R12,R12              Is the storage already present?
    BNZR R6                   ..Yes Return
    EXEC CICS GETMAIN LENGTH(32) SET(R12) SHARED X
    INITIMG(X'00') RESP(RESP)
    CLC  RESP,DFHRESP(NORMAL) GETMAIN worked OK?
    BNE  ERROR7              ..No Goto Error routine
    L    R1,UEPTQTOK         Fetch address of token
    ST   R12,0(R1)          Save address of storage
    USING SHARED_STORAGE,R12
    MVC  SHARED_EYECATCHER,EYE_CATCHER Set Eyecatcher
    DROP R12                Drop R12
    BR   R6                  Return to caller
    EJECT ,
*

```

Figure 85 (Part 12 of 16). Example exit program for the XTSEREQ exit


```

*-----*
* FREEMAIN_SHARED - Free shared storage *
* Free the shared storage associated with this command. *
* *
* Registers: *
* R0 = Used by EXEC CICS call *
* R1 = Used by EXEC CICS call *
* R6 = Link Register - Return Address *
* R11= EIB register (set on entry) *
* R12= Work register *
* R14= Used by EXEC CICS call *
* R15= Used by EXEC CICS call *
* *
* Logic: *
* Freemain_Shared: *
* Address shared storage *
* If eyecatcher ^= 'XTSEREQ Storage' then *
* Call Error(Freemain_Logic_Error) *
* Else *
* EXEC CICS FREEMAIN DATAPOINTER(UEPTQTOK) RESP(resp) *
* If resp ^= OK then *
* Call Error(Freemain_Failed) *
* End If *
* End If *
* Return *
*-----*
FREEMAIN_SHARED DS 0H
    L    R12,UEPTQTOK           Fetch token address
    L    R12,0(R12)            Address shared storage address
    USING SHARED_STORAGE,R12    ..Using R12
    CLC  SHARED_EYECATCHER,EYE_CATCHER Is this our storage?
    BNE  ERROR8                ..No Goto Error routine
    DROP R12                    Drop R12
    EXEC CICS FREEMAIN DATAPOINTER(R12) RESP(RESP)
    CLC  RESP,DFHRESP(NORMAL)   FREEMAIN worked OK?
    BNE  ERROR9                ..No Goto Error routine
    L    R12,UEPTQTOK           Fetch token address
    XC   0(4,R12),0(R12)        Clear token address
    BR   R6                     Return to caller
    EJECT ,
*
*-----*
* Trace Routines *
* Issue a Trace XPI call *
* *
* Registers: *
* R0 = Used by XPI call *
* R1 = DFHTRPT plist *
* R6 = Link Register - Return Address *
* R12= Work register *
* R13= EISTG register (set by DFHEIENT) *
* Kernel Stack entry *
* R14= Used by XPI call *
* R15= Used by XPI call *
*-----*

```

Figure 85 (Part 13 of 16). Example exit program for the XTSEREQ exit

example XTSEREQ global user exit program

```

        USING DFHTRPT_ARG,R1
TRACE_ENTRY DS 0H
        L    R1,UEPXSTOR          Prepare for XPI call
        DFHTRPTX CLEAR,
                                           X
                POINT_ID(TR_ENTRY)
        B    ISSUE_TRACE
TRACE_EXIT DS 0H
        L    R1,UEPXSTOR          Prepare for XPI call
        DFHTRPTX CLEAR,
                                           X
                POINT_ID(TR_EXIT)
        B    ISSUE_TRACE
TRACE_ERROR DS 0H
        L    R1,UEPXSTOR          Prepare for XPI call
        DFHTRPTX CLEAR,
                                           X
                POINT_ID(TR_ERROR),
                DATA1(TR_ERROR_N,1)
        BAL  R6,ISSUE_TRACE
        B    RETURN
*
*-----*
* Issue the Trace XPI call
*-----*
ISSUE_TRACE DS 0H
        L    R8,UEPTRACE          Address of trace flag
        TM   0(R8),UEPTRON        Is trace on?
        BZ   NO_TRACE             No - do not issue trace then
        LR   R12,R13              Save R13 round XPI call
        L    R13,UEPSTACK
        DFHTRPTX CALL,
                                           X
                IN,
                                           X
                FUNCTION(TRACE_PUT),
                                           X
                POINT_ID(*),
                                           X
                OUT,
                                           X
                RESPONSE(*),
                                           X
                REASON(*)
        LR   R13,R12              Restore R13 (DFHEISTG)
NO_TRACE DS 0H
        BR   R6                   Return to caller
        DROP R1
*
*-----*
* ERRORn
* Error has occurred during processing
* Issue a trace point and return to the CICS
*-----*
ERROR1 DS 0H
        MVI  TR_ERROR_N,1
        B    TRACE_ERROR
ERROR2 DS 0H
        MVI  TR_ERROR_N,2
        B    TRACE_ERROR
ERROR3 DS 0H
        MVI  TR_ERROR_N,3
        B    TRACE_ERROR

```

Figure 85 (Part 14 of 16). Example exit program for the XTSEREQ exit

```

ERROR4 DS 0H
      MVI TR_ERROR_N,4
      B TRACE_ERROR
ERROR5 DS 0H
      MVI TR_ERROR_N,5
      B TRACE_ERROR
ERROR6 DS 0H
      MVI TR_ERROR_N,6
      B TRACE_ERROR
ERROR7 DS 0H
      MVI TR_ERROR_N,7
      B TRACE_ERROR
ERROR8 DS 0H
      MVI TR_ERROR_N,7
      B TRACE_ERROR
ERROR9 DS 0H
      MVI TR_ERROR_N,7
      B TRACE_ERROR
      EJECT ,
      DROP R2 Drop DFHUEPAR
      DROP R11 Drop EIB
      LTORG ,
*****
* CONSTANTS *
*****
      DS 0D
EYE_CATCHER DC CL16'XTSEREQ Storage '
DEFAULT_SYSID DC CL4'MQ1 '
LOCAL EQU X'01'
ROUTE EQU X'02'
*
* Trace point ids
TR_ENTRY DC XL2'120'
TR_EXIT DC XL2'121'
TR_ERROR DC XL2'122'
*
*-----*
* TABLE_ENTRY: *
*-----*
* | Entry_Name | New_Name | QOR_Sysid | Action | *filler* | *
* | Char 8 | Char 8 | Char 4 | Bin 1 | Char 3 | *
*-----*
* Last Entry is indicated by special TS_Queue Name *
*-----*

```

Figure 85 (Part 15 of 16). Example exit program for the XTSEREQ exit

example XTSERREQ global user exit program

```

TS_ROUTING_TABLE DS 0D
ENTRY_NAME_1     DC CL8'AAAAAAA'      Rename Queue AAAAAAA as
NEW_NAME_1       DC CL8'BBBBBBBB'     BBBBBBBBBB
QOR_SYSID_1      DC CL4' '
ACTION_1         DC XL1'01'           Local request
FILLER_1         DC CL3' '
ENTRY_NAME_2     DC CL8'A1'           '      Rename Queue A1 as
NEW_NAME_2       DC CL8'B1'           '      B1
QOR_SYSID_2      DC CL4' '
ACTION_2         DC XL1'01'           Local request
FILLER_2         DC CL3' '
ENTRY_NAME_3     DC CL8'A2'           '      Rename Queue A2 as
NEW_NAME_3       DC CL8'B2'           '      B2
QOR_SYSID_3      DC CL4' '
ACTION_3         DC XL1'01'           Local request
FILLER_3         DC CL3' '
ENTRY_NAME_4     DC CL8'RRRRRRR'      '      Rename Queue RRRRRRRR as
NEW_NAME_4       DC CL8'REMOTE'      '      REMOTE and ship request
QOR_SYSID_4      DC CL4'MQ1'         '      to System MQ1
ACTION_4         DC XL1'02'
FILLER_4         DC CL3' '
ENTRY_NAME_5     DC CL8'R1'           '      Don't rename Queue R1, but
NEW_NAME_5       DC CL8'R1'           '      ship request to System MQ1
QOR_SYSID_5      DC CL4'MQ1'
ACTION_5         DC XL1'02'
FILLER_5         DC CL3' '
ENTRY_NAME_LAST  DC XL8'FFFFFFFFFFFF'
NEW_NAME_LAST    DC CL8' '
QOR_SYSID_LAST   DC CL4' '
ACTION_LAST      DC XL1'00'
FILLER_LAST      DC CL3' '
                END DFH$XTSE

```

Figure 85 (Part 16 of 16). Example exit program for the XTSERREQ exit

Index

Special Characters

- “good night” transaction
 - customizing the sample program 639
 - overview 635
 - sample program, DFH0GNIT 637

Numerics

- 3270 information display system
 - error processors (optional) 413
 - TCAM 606
 - unavailable printer
 - DFHZNEP 422

A

- abends
 - abend and restart, TCAM 608
 - transaction bit 386
- abnormal conditions
 - in terminal error programs 384
 - sample node error program 411
 - sample terminal error program 363
 - user-written node error programs 421
- abort write bit 386
- ACF/VTAM
 - application routing failure 410
 - automatic installation 431
 - CINIT request unit 434
 - CLSDST PASS function 410
 - default DFHZNEP 396
 - DFHZNAC logging facility 410
 - entries in LOGON mode table 669
 - error-handling 396
 - DFHZNAC/DFHZNEP interface 396
 - DFHZNAC/DFHZNEP interface action flags 398
 - generic resources 429
 - node error program 429
 - ISTINCLM values 674
 - node error program (DFHZNEP) 411
 - PSERVIC values 679
 - session failures
 - user-written NEPs 423
 - transaction-class error-handling routine 403
 - VTAM LOGON mode table 432
- ACQUIRE PROGRAM function of the XPI 271
- action flag names, DFHTEP 369
- adapter, task-related user exits 201
- ADD SUSPEND function of the XPI 251
- addressing mode implications 217
- ADYN, dynamic allocation transaction 614

- AIEXIT, system initialization parameter 432, 452
- AILDELAY, system initialization parameter 121, 581
- AIRDELAY, system initialization parameter 428, 581
- allocate queues
 - controlling the length of
 - using the XISCONA global user exit 77
 - using the XZIQUE global user exit 188
- APPC connections, automatic installation of 451
- assembling and link-editing a user-replaceable program 344
- autoinstall
 - See automatic installation of APPC connections
 - See automatic installation of programs
 - See automatic installation of shipped terminals
 - See automatic installation of terminals
- autoinstall user-replaceable programs
 - for APPC connections (DFHZATDY) 451
 - for programs (DFHPDADX) 467
 - for shipped terminals (DFHZATDX) 459
 - for shipped terminals (DFHZATDY) 459
 - for terminals (DFHZATDX) 444
 - link-edit statements 347
- automatic installation of APPC connections
 - benefits of 452
 - control program
 - at delete 456
 - parameter list at install 453
 - purpose of 453
 - introduction 451
 - model definitions 452
 - parallel-session 452
 - recovery and restart 453
 - requirements for 452
 - single-session
 - initiated by BIND 452
 - initiated by CINIT 451
 - supplied resource definitions 457
 - templates 452
 - the sample program 456
 - default actions 456
- automatic installation of programs
 - benefits of 469
 - control program
 - parameter list at install 470
 - testing 475
 - installation of mapsets 468
 - introduction 467
 - model definitions 468
 - requirements for 469
 - supplied resource definitions 474
 - system autoinstall 468
 - the sample programs
 - customizing 473

- automatic installation of programs (*continued*)
 - the sample programs (*continued*)
 - DFHPGADX 472
 - DFHPGAHX 472
 - DFHPGALX 472
 - DFHPGAOX 472

- automatic installation of shipped terminals
 - control program
 - parameter list at delete 464
 - parameter list at install 462
 - introduction 459

- automatic installation of terminals
 - control program
 - action at delete 441
 - action at install 433
 - action on return 440
 - information returned to CICS 437
 - link-edit statements 347
 - naming 442
 - testing and debugging 442
 - parameter list at logon 434
 - the sample programs
 - customizing 446
 - DFHZATDX 444
 - DFHZCTDX 444
 - DFHZDTEX 444
 - DFHZPTDX 444

- VTAM LOGON mode table 432

- automatic journal archiving 537

B

- Basic Mapping Support (BMS)

- global user exit points 24

- batch processing, TCAM SNA 594

- BDAM files

- exit for backout of WRITE-ADD operations 49, 172

C

- CAVM data sets, opening by overseer program 505

- CEBT function in the overseer program 516

- CEDA transaction

- programmable interface to 663

- CEMT INQUIRE AUTOINSTALL 442

- CEMT INQUIRE MONITOR 546

- CEMT PERFORM STATISTICS 580

- CEMT SET AUTOINSTALL 442

- CEMT SET MONITOR 546

- CHANGE PRIORITY function of the XPI 262

- CICS system definition utility program (DFHCSDUP)

- EXTRACT command 644

- invocation from a user program 653

- running under TSO 653

- sample programs 647

- DFH\$CRFA 647

- DFH\$CRFP 647

- CICS system definition utility program (DFHCSDUP) (*continued*)

- sample programs (*continued*)

- DFH\$FORA 647

- DFH\$FORP 647

- DFH0CBDC 647

- DFH0CRFC 647

- DFH0FORC 647

- user exits 656

- writing a user program 644

- invocation points 645

- link-edit statements 649

- parameters passed from DFHCSDUP 646

- sample JCL, assembler-language 650

- sample JCL, PL/I 652

- sample JCL, VS COBOL II 651

- CICS-DBCTL interface status program (DFHDBUEX)

- communications area 497

- introduction to 497

- link-edit statements 348

- sample program 498

- CINIT request unit 434

- CINIT, VTAM 443

- clock, definition

- for monitoring 557

- closing journal data sets 522

- CLSDSTP, system initialization parameter 410

- CODE operand

- DFHSNEP TYPE=ERRPROC 417

- DFHTEPM TYPE=ERRPROC 376

- DFHTEPT TYPE=BUCKET 383

- DFHTEPT TYPE=PERMCODE|ERRCODE 380

- common subroutine vector table (CSVT) 412, 420

- communication area

- terminal error program 363

- communication control bytes 593, 604

- communications area

- autoinstall control program

- APPC connections 454

- programs 470

- terminals 434

- CICS-DBCTL interface status program 497

- dynamic transaction routing program 484

- journaling user-replaceable programs 493

- node error program 404

- program error program 353

- terminal error program 385

- transaction restart program 358

- COUNT operand

- DFHSNET macro 419

- DFHTEPT TYPE=PERMCODE|ERRCODE 380

- limits, default threshold for TEP 380

- CS operand

- DFHSNEP TYPE=INITIAL 415

- CSD utility program (DFHCSDUP) 649

CSNE transaction 396
CSVT (common subroutine vector table) 412
customizing the overseer program 516

D

data format
 TCAM 596
data sets, journal
 closing 522
 opening 522
data tables 124
database control (DBCTL)
 DBCTL information about the CAVM data sets 514
 DFHDBUEX 497
 DFHMCT TYPE=EMP entries 544
 in an XRF environment 516
 interface status 497
DBCTL (database control)
 DBCTL information about the CAVM data sets 514
 DFHDBUEX 497
 DFHMCT TYPE=EMP entries 544
 in an XRF environment 516
 interface status 497
DBLID values 513
DD card correlation TCAM 595
DECB, terminal error program
 information 370
 operand 370
DEFAULT operand
 DFHZNEPI TYPE=INITIAL 425
default threshold count limits
 DFHTEP (terminal error program) 380
DEFINE PROGRAM function of the XPI 268
defining terminal error blocks 379
DELETE PROGRAM function of the XPI 275
DELETE SUSPEND function of the XPI 257
device message handler (DMH) 592
DFH\$AXRO, sample overseer program 505
DFH\$CRFA, cross-reference program,
 assembler-language 647
DFH\$CRFP, cross-reference program, PL/I 647
DFH\$FORA, formatting program,
 assembler-language 647
DFH\$FORP, formatting program, PL/I 647
DFH\$PCPI, sample program for global user exits 15
DFH\$SDAP, sample shutdown program 338
DFH\$TDWT, transient data write-to-terminal
 program 713
DFH\$XTSE, example program for XTSEREQ exit 717
DFH\$XZIQ, sample global user exit program 195
DFH0CBDC program, write DEFINE commands for VS
 COBOL II 647
DFH0CRFC, cross-reference program, VS COBOL
 II 647
DFH0FORC, formatting program, VS COBOL II 647
DFH0GNIT, sample "good night" program 637
DFHAPIQX macro 297
DFHCSDUP, system definition utility program
 See CICS system definition utility program
 (DFHCSDUP)
DFHDBUEX, CICS-DBCTL interface status program
 communications area 497
 introduction to 497
 link-edit statements 348
 sample program 498
DFHDSATX macro 249
DFHDSRX macro 249
DFHDUDUX macro 263
DFHDXGHD DSECT 514
DFHDYP, dynamic transaction routing program
 changing the program name 480
 changing the target region 479
 communications area 484
 error handling 480
 information passed to 478
 invoking on abend 481
 link-edit statements 347
 modifying application's communications area 482
 modifying initial terminal data 482
 overview 477
 processing considerations 483
 receiving information from routed transaction
 monitoring the output COMMAREA 483
 monitoring the output TIOA 483
 renaming customized version 491
 routing a transaction 480
 sample program 492
 testing customized version 491
 UOW considerations 484
 when invoked 477
DFHEIP, EXEC interface program 344
DFHJCADS, journal control area DSECT 522
DFHJCJCX macro 330
DFHJCP, global user exits 537
DFHJUP, journal utility program 523
DFHKEDSX macro 267
DFHLDLX macro 268
DFHMCTDR, monitoring dictionary DSECT 549
DFHMNMNX macro 276
DFHNET DSECTs 420
DFHOSD data set 505
DFHPEP, program error program
 communication area for assembler-language
 programs 353
 link-edit statements 346
 source code 353
 writing 351
DFHPGADX, user-replaceable autoinstall program
 customizing 473
 installation of mapsets 468

DFHPGADX, user-replaceable autoinstall program
(*continued*)
 introduction to 467
 parameter list at install 470
 sample program 472
 supplied definition of 474
 use of model definitions 468
 when invoked 467

DFHPGAQX macro 280

DFHPGISX macro 280

DFHREST, transaction restart program
 communications area 358
 default program 360
 introduction 357
 link-edit statements 346
 transactions suitable for restart 357
 when invoked 357

DFHRMCAL macro 201

DFHSAIQX macro 299, 303

DFHSIT (system initialization table)
 entries for CICS monitoring 546

DFHSMFDS, SMF header DSECT 535, 547

DFHSMMCX macro 305

DFHSMSRX macro 310

DFHSNEP macro
 TYPE=DEF3270 416
 TYPE=DEFILU 416
 TYPE=ERRPROC 400, 417
 TYPE=FINAL 416
 TYPE=INITIAL 399, 415
 TYPE=USTOR 415
 TYPE=USTOREND 415

DFHSNEP, sample node error program 414

DFHSNET macro 418
 COUNT operand 419
 ESB structure 419
 ESBS operand 419
 NAME operand 419
 NEBNAME operand 419
 NEBS operand 419
 TIME operand 419

DFHSTUP, statistics processing program 588

DFHTACP (terminal abnormal condition program)
 default actions (TCAM) 603

DFHTACP, terminal abnormal condition program 362
 terminal error-handling 362

DFHTEPM macro
 examples 376
 TYPE=ENTRY 375
 TYPE=ERRPROC 375
 TYPE=EXIT 375
 TYPE=FINAL 376
 TYPE=INITIAL 372

DFHTEPT macro
 examples 383
 TYPE=BUCKET 382

DFHTEPT macro (*continued*)
 TYPE=FINAL 383
 TYPE=INITIAL 378
 TYPE=PERMCODE|ERRCODE 379
 TYPE=PERMTID 379

DFHTRPTX macro 313

DFHUEPAR DSECT 8, 205

DFHUERTR DSECT 208

DFHUEXIT macro 8

DFHWOS, overseer module 505

DFHWOSM macros 506
 FUNC=BUILD 507
 FUNC=CLOSE 508
 FUNC=DSECT 508
 FUNC=JJC 509
 FUNC=JJS 509
 FUNC=OPEN 510
 FUNC=OSCMD 511
 FUNC=QJJS 509
 FUNC=READ 512, 513
 FUNC=TERM 515
 token values 506

DFHXJCC, user-replaceable journal program
 communication area 493
 introduction 493
 limitations 494
 link-edit statements 347
 sample program 495

DFHXJCO, user-replaceable journal program
 communication area 493
 introduction 493
 limitations 494
 link-edit statements 347
 sample program 495

DFHXMCLX macro 317

DFHXMIQX macro 325, 329

DFHXMSRX macro 314, 315

DFHXMIDX macro 318

DFHXSEAI, security domain API stub 630

DFHXTEP, sample terminal error program 363

DFHZATDX, user-replaceable autoinstall program
 action at delete 441
 action at install 433
 communications area 442
 customizing 446
 introduction 431
 link-edit statements 347
 sample control program 444
 source code 444
 suggestions for use 444
 used to install shipped terminals 459

DFHZATDY, user-replaceable autoinstall program
 communications area 454
 default actions 456
 for APPC single-session connections
 initiated by CINIT 451

DFHZATDY, user-replaceable autoinstall program
(*continued*)
 for parallel-session APPC connections 452
 for single-session APPC connections
 initiated by BIND 452
 introduction to 451
 purpose of 453
 supplied definition of 457
 the sample program 456
 used to install shipped terminals 459
 when invoked 453

DFHZNAC, node abnormal condition program 396
 action flag settings 711
 default actions
 for system sense codes 709
 for terminal error codes 697
 execution after XRF takeover 427
 execution with persistent session support 427
 execution with VTAM generic resources 429
 logging facility 410
 terminal error-handling 403

DFHZNEP, user-replaceable node error program 395

DFHZNEPI macros
 TYPE=ENTRY 425
 TYPE=FINAL 426
 TYPE=INITIAL 425

dictionary data section, CICS monitoring records 549,
 554

dispatcher functions of the XPI 249

display function of the overseer program 502

DL/I
 DFHMCT TYPE=EMP entries 543
 journal records 535

DMH (device message handler) 592

DSECTPR operand
 DFHTEPM TYPE=INITIAL 372

DTRPGM, system initialization parameter 491

DTRTRAN, system initialization parameter 479

dump control functions of the XPI 263

DYNALLOC (dynamic allocation sample program)
 See dynamic allocation sample program
 (DYNALLOC)

dynamic allocation sample program (DYNALLOC)
 flow of control 616
 help feature 614
 introduction 613
 keywords, abbreviation rules 616
 system programming considerations 616
 terminal operation 614
 values 615

DYNAMIC option 477

dynamic transaction routing
 overview 477
 sample programs 492
 the user program
 error handling procedure 480
 link-edit statements 347

dynamic transaction routing (*continued*)
 the user program (*continued*)
 naming of 491
 parameters 484
 testing of 491
 when invoked 477

dynamic transaction routing program (DFHDYP)
 changing the program name 480
 changing the target region 479
 communications area 484
 error handling 480
 information passed to 478
 invoking on abend 481
 link-edit statements 347
 modifying application's communications area 482
 modifying initial terminal data 482
 overview 477
 processing considerations 483
 receiving information from routed transaction
 monitoring the output COMMAREA 483
 monitoring the output TIOA 483
 renaming customized version 491
 routing a transaction 480
 sample program 492
 testing customized version 491
 UOW considerations 484
 when invoked 477

dynamic transactions 477

E

early verification processing 629

EDF (Execution Diagnostic Facility)
 with global user exits 6
 with task-related user exits 204

EMP (event-monitoring point) 540

END_BROWSE_PROGRAM function of the XPI 294

error group index 412, 419

error groups 398

error processing 361
 in node error program (NEP) 411
 in terminal error program (TEP) 361

error status block (ESB) 419

error status element (ESE) 365, 370
 DFHTEPT TYPE=PERMCODE|ERRCODE 379

ESB (error status block) 419

ESBS operand
 DFHSNET macro 419

ESE (error status element) 365, 370
 DFHTEPT TYPE=PERMCODE|ERRCODE 379

ESM (external security manager) 621

ESMEXITS, system initialization parameter 624

event-monitoring point (EMP) 540

exception class monitoring 544

exception class monitoring records 539

- exception class statistics records 579
- exception data section format 556
- EXEC CICS HANDLE command
 - as alternative to node error program 395
- EXEC CICS INQUIRE command
 - for autoinstall 442
- EXEC CICS PERFORM command
 - for requested statistics 580
- EXEC CICS SET command
 - for autoinstall 442
 - for opening and closing journal data sets 522
- EXEC CICS WRITE JOURNALNUM command 522
- EXEC interface program (DFHEIP) 344
- Execution Diagnostic Facility (EDF)
 - with global user exits 6
 - with task-related user exits 204
 - with user-replaceable programs 343
- exit programming interface (XPI)
 - See XPI (exit programming interface)
- extended recovery facility (XRF)
 - node error program 427
 - overseer program 501
- external CICS interface
 - the user program
 - link-edit statements 348
- external security
 - changes to link security for MRO 633
- external security manager (ESM) 621
- EXTRACT command
 - for task-related user exits 230
 - of DFHCSDUP 644

F

- field connectors, CICS monitoring 554, 555
- field identifiers, CICS monitoring 555
- FMH processing 593
- FREEMAIN function of the XPI 308
- function shipping, queuing due to
 - See XISCONA, global user exit
 - See XZIQUE, global user exit

G

- generalized message format, TCAM 605
- generic resources, VTAM
 - node error program 429
- GET_NEXT_PROGRAM function of the XPI 292
- GETMAIN function of the XPI 306
- global user exits
 - example programs 17
 - for EXEC interface exits 717
 - for mixing API and XPI calls 6, 717
 - for modifying TS requests 150, 717
 - for XFCREQ 63, 66
 - for XFCREQC 63, 66
 - for XICEREQ 95

- global user exits (*continued*)
 - example programs (*continued*)
 - for XICEREQ 95
 - for XPCREQ 111
 - for XTDEREQ 177
 - for XTDEREQC 177
 - for XTSEREQ 150, 717
 - for XTSEREQC 150, 717
 - exit points
 - for 'terminal not known' condition 157
 - in activity keypoint program 23
 - in BMS 24
 - in data tables programs 124
 - in DBCTL interface control program 29
 - in DBCTL tracking program 30
 - in dispatcher domain 32
 - in DL/I interface program 34
 - in dump domain 39
 - in dynamic transaction backout program 46
 - in EXEC interface program 51
 - in file control EXEC interface program 56
 - in file control open/close program 54
 - in file control state program 67
 - in Front End Programming Interface 75
 - in good-morning message program 76
 - in intersystem communication program 77
 - in interval control program 82
 - in journal control program 96, 537
 - in loader domain 98
 - in message domain 100
 - in monitoring domain 104
 - in program control program 106
 - in resource management modules 121
 - in resource manager interface program 119
 - in security manager domain 131
 - in statistics domain 133, 588
 - in system recovery program 135
 - in system termination program 138
 - in task control program 139
 - in temporary storage control program 140
 - in temporary storage EXEC interface program 142
 - in terminal allocation program 152
 - in terminal control program 154
 - in transaction backout programs 165
 - in transaction manager domain 173
 - in transient data EXEC interface program 177
 - in transient data program 175
 - in VTAM terminal management program 185
 - in VTAM working-set module 186
 - in XRF request-processing program 198
 - exit programs
 - addressing implications 5
 - defining, enabling, and disabling 13
 - errors 12
 - global work area 7
 - multiple at one exit 13

- global user exits (*continued*)
 - exit programs (*continued*)
 - one at several exits 14
 - parameters passed 7
 - programming interface restrictions 11
 - register conventions 4
 - returning values to CICS 10
 - using CICS services 5
 - using EDF 6
 - example programs
 - for XPCREQC 111
 - overview 3
 - sample programs
 - DFH\$PCEX 114
 - DFH\$PCGA 14
 - DFH\$PCPI 15
 - DFH\$PCTA 117
 - DFH\$SXP1 103
 - DFH\$SXP2 103
 - DFH\$SXP3 103
 - DFH\$SXP4 103
 - DFH\$SXP5 103
 - DFH\$SXP6 103
 - DFH\$ZCAT 14
 - DFH\$ZCGA 14
 - DFHXIS 78
 - DFHXTENF 163
 - list of 14
 - summary of 14
 - trace table entries 7
 - with storage protection
 - data storage key 12
 - execution key 11
 - GMTRAN, system initialization parameter 408
 - GNTRAN, system initialization parameter 635, 639
 - GROUP operand
 - DFHSNEP TYPE=ERRPROC 417

I

- IEDRH macro 593
- initialization programs
 - considerations when writing 335
- INITPARMS, system initialization parameter 15
- INQ_APPLICATION_DATA function of the XPI 297
- INQUIRE MONITOR command 546
- INQUIRE MONITORING DATA function of the XPI 279
- INQUIRE_ACCESS function of the XPI 309
- INQUIRE_AUTOINSTALL function of the XPI 295
- INQUIRE_CURRENT_PROGRAM function of the XPI 286
- INQUIRE_DTRTRAN function of the XPI 314
- INQUIRE_ELEMENT_LENGTH function of the XPI 310

- INQUIRE_MXT function of the XPI 315
- INQUIRE_PROGRAM function of the XPI 280
- INQUIRE_SHORT_ON_STORAGE function of the XPI 310
- INQUIRE_SYSTEM function of the XPI 299
- INQUIRE_TASK_STORAGE function of the XPI 311
- INQUIRE_TCLASS function of the XPI 317
- INQUIRE_TRANDEF function of the XPI 318
- INQUIRE_TRANSACTION function of the XPI 325
- interactive logical unit error processor 414
- intersystem queues
 - controlling the length of
 - using the XISCONA global user exit 77
 - using the XZIQUE global user exit 188
- INTLU error processor 414
- ISSUE PASS command 410
- ISTINCLM entries for automatic installation 674

J

- JCA (journal control area) 522
- job control for sample DFHTEP generation 371
- journal control area (JCA) 522
- journal control label record 530
- journal module identifiers 529
- journal records
 - data section format 535
 - delimiters 534
 - for DL/I 535
 - formats and contents 523
 - journalized data 532
 - logged output message 533
 - module identifiers 529
 - system header 524
 - system prefix 531
 - user prefix 532
 - written to SMF 535
- journaling
 - automatic journal archiving 537
 - global user exits 537
 - user-replaceable programs 493
- journals
 - closing 522
 - opening 522
 - printing 523
 - reading 522
 - offline 522

K

- kernel domain functions of the XPI 267

L

- line locking
 - permanent lock, TCAM 602

- line locking (*continued*)
 - temporary lock, TCAM 602
- line pool specifications (TCAM)
 - POOL feature 601
 - restrictions 602
- loader functions of the XPI 268
- logic flow, TCAM 597
- logical units (LUs)
 - node error program 403
- LOGON mode table, VTAM 669

M

- MAXERRS operand
 - DFHTEPT TYPE=INITIAL 378
- MAXTIDS operand
 - DFHTEPT TYPE=INITIAL 378
- MCT (monitoring control table)
 - entries for DBCTL 544
 - entries for DL/I 543
 - entries for EMPs 540
- messages, TCAM
 - control program 604, 610
 - DEST operand 600
 - format 605
 - handler 594
 - MCP (message control program) 610
 - routing 600
- MN, system initialization parameter 546
- MNEVE, system initialization parameter 546
- MNEXC, system initialization parameter 546
- MNPER, system initialization parameter 546
- model definitions
 - for autoinstall of APPC connections 452
 - for automatic installation of programs 468
- model terminal support
 - coding entries 433
- MONITOR function of the XPI 276
- monitoring
 - clock definition 557
 - control commands 546
 - control table (MCT) 540
 - data produced 557
 - data section format 549
 - DFHMCT entries for DBCTL 544
 - DFHMCT entries for DL/I 543
 - DFHMCTDR, the dictionary DSECT 549
 - DFHSIT entries 546
 - DFHSMFDS, SMF header DSECT 547
 - dictionary data section 549, 554
 - event-monitoring point (EMP) 540
 - exception class data 539
 - exception data section format 556
 - exception data sections 556
 - field connectors 554, 555
 - field identifiers 555

- monitoring (*continued*)
 - functions of the XPI 276
 - monitoring control table 540
 - overview 539
 - passing the data to SMF 546
 - performance class data 539, 558
 - performance record format 553
 - purpose 539
 - record formats 547
 - record types 539
 - RMF 545
 - SMF 546
 - SMF header 547
 - SMF product section 547
 - SYSEVENT class of data 545
 - time stamp definition 558
- monitoring control table (MCT) 540
- MRO (multiregion operation)
 - IEAICS parameters 545
- MVS IEAICS member 545
- MVS router exit 622
- MVS/ESA
 - IEAICS member 545

N

- NAME operand
 - DFHSNEP TYPE=INITIAL 416
 - DFHSNET macro 419
- national characters
 - uppercase translation 715
- NEB (node error block) 420
- NEBNAME operand
 - DFHSNET macro 419
- NEBS operand
 - DFHSNET macro 419
- NEP (node error program)
 - 3270 unavailable printer 422
 - ACF/VTAM error handling
 - background 396
 - application routing failure 410
 - common subroutine vector table (CSVT) 420
 - communication area 404
 - conventions for registers 417
 - default actions of DFHZNAC
 - for system sense codes 709
 - for terminal error codes 697
 - default node error program 398
 - default transaction-class routine 425
 - DFHNET DSECT 420
 - DFHSNET 418
 - DFHZNAC 403
 - DFHZNAC action flag settings 711
 - DFHZNAC logging facility 410
 - DFHZNAC/DFHZNEP interface 396
 - DFHZNEP 396, 403

NEP (node error program) (*continued*)

- DFHZNEPI interface module 424
- DFHZNEPI macros 424
- DFHZNEPI TYPE=INITIAL 425
- DSECTs 420
- error groups 398
- error status blocks 420
- error table header 420
- in an XRF environment 427
 - changing the recovery message 428
 - changing the recovery notification 428
 - changing the recovery transaction 428
- link-edit statements 346
- multiple NEPs 401
- NEPCLASS 401
- NET generation 398
- node abnormal condition program 403
- node error block, format 413
- node error blocks 420
- node error table 413
 - format 413
 - generation 398
- reasons for writing your own 396
- routing considerations 402
- sample 398, 411
 - addressability 412
 - coding description 399
 - common subroutine vector table (CSVT) 412
 - compatibility with sample TEP 411
 - components 412
 - conditions 401
 - CSVT (common subroutine vector table) 412
 - DFHSNEP TYPE=INITIAL macro 415
 - DFHSNEP TYPE=USTOR macro 415
 - DFHSNEP TYPE=USTOREND macro 415
 - error processor vector table (EPVT) 412, 416
 - error processors for INTLU, DFHSNEP TYPE=DEFILU 416
 - error processors, DFHSNEP TYPE=DEF3270 416
 - error status information 413
 - generating by DFHSNEP 414
 - node error table 413
 - optional common subroutines 413
 - optional error processor for INTLU 414
 - optional error processors for 3270 413
 - routing mechanism (ACF/VTAM) 412
- session failures 423
- TERMERR condition 395
- terminal control program (ACF/VTAM section) 403
- user-supplied error processors, DFHSNEP TYPE=ERRPROC 417
- user-written 421
 - addressability 422
 - restrictions on use 421
- user-written error processors 417

NEP (node error program) (*continued*)

- when abnormal condition occurs 403
- with persistent session support 427
- with VTAM generic resources 429
- writing overview 397
- NEPCLAS operand
 - DFHZNEPI TYPE=ENTRY 425
- NEPCLASS operand
 - for CEDA 401
- NEPNAME operand
 - DFHZNEPI TYPE=ENTRY 425
- NET (node error table) 398
- NETNAME operand
 - DFHSNEP TYPE=INITIAL 416
- node abnormal condition program (NACP) 403
- node error block (NEB) 420
- node error handler (CSNE transaction) 396
- node error program (NEP)
 - 3270 unavailable printer 422
 - ACF/VTAM error handling
 - background 396
 - application routing failure 410
 - common subroutine vector table (CSVT) 420
 - communication area 404
 - conventions for registers 417
 - default actions of DFHZNAC
 - for system sense codes 709
 - for terminal error codes 697
 - default node error program 398
 - default transaction-class routine 425
 - DFHNET DSECT 420
 - DFHSNET 418
 - DFHZNAC 403
 - DFHZNAC action flag settings 711
 - DFHZNAC logging facility 410
 - DFHZNAC/DFHZNEP interface 396
 - DFHZNEP 396, 403
 - DFHZNEPI interface module 424
 - DFHZNEPI macros 424
 - DFHZNEPI TYPE=INITIAL 425
 - DSECTs 420
 - error groups 398
 - error status blocks 420
 - error table header 420
 - in an XRF environment 427
 - changing the recovery message 428
 - changing the recovery notification 428
 - changing the recovery transaction 428
 - link-edit statements 346
 - multiple NEPs 401
 - NEPCLASS 401
 - NET generation 398
 - node abnormal condition program 403
 - node error block, format 413
 - node error blocks 420
 - node error table 413
 - format 413

- node error program (NEP) *(continued)*
 - node error table *(continued)*
 - generation 398
 - reasons for writing your own 396
 - routing considerations 402
 - sample 398, 411
 - addressability 412
 - coding description 399
 - common subroutine vector table (CSV T) 412
 - compatibility with sample TEP 411
 - components 412
 - conditions 401
 - CSV T (common subroutine vector table) 412
 - DFHSNEP TYPE=INITIAL macro 415
 - DFHSNEP TYPE=USTOR macro 415
 - DFHSNEP TYPE=USTOREND macro 415
 - error processor vector table (EPVT) 412, 416
 - error processors for INTLU, DFHSNEP
 - TYPE=DEFILU 416
 - error processors, DFHSNEP
 - TYPE=DEF3270 416
 - error status information 413
 - generating by DFHSNEP 414
 - node error table 413
 - optional common subroutines 413
 - optional error processor for INTLU 414
 - optional error processors for 3270 413
 - routing mechanism (ACF/VTAM) 412
 - session failures 423
 - TERMERR condition 395
 - terminal control program (ACF/VTAM section) 403
 - user-supplied error processors, DFHSNEP
 - TYPE=ERRPROC 417
 - user-written 421
 - addressability 422
 - restrictions on use 421
 - user-written error processors 417
 - when abnormal condition occurs 403
 - with persistent session support 427
 - with VTAM generic resources 429
 - writing overview 397
- node error table (NET) 398
- nonpurgeable task 386
- notation, syntax xii

O

- OPTCD operand 600
- OPTIONS operand
 - DFHTEPM TYPE=INITIAL 372
 - DFHTEPT TYPE=INITIAL 379
- overseer program
 - customizing the sample program 516
 - including the CEBT command 516
 - loop or wait detection in the active 517
 - DFH\$AXRO 505

- overseer program *(continued)*
 - DFHOSD data set 505
 - DFHWOSM macros 506
 - FUNC=BUILD 507
 - FUNC=CLOSE 508
 - FUNC=DSECT 508
 - FUNC=JJC 509
 - FUNC=JJS 509
 - FUNC=OPEN 510
 - FUNC=OSCMD 511
 - FUNC=QJJS 509
 - FUNC=READ 512
 - FUNC=TERM 515
 - display function 502
 - interface with CICS 505
 - module DFHWOS 505
 - opening CAVM data sets dynamically 505
 - restart-in-place function 503
 - enabling and disabling 504
 - rules governing restart in place 504

P

- PEP (program error program) 351
 - communication area for assembler-language
 - programs 353
 - source code 353
 - writing 351
 - performance class data, CICS monitoring 558
 - performance class monitoring records 539
 - performance class statistics records 579
 - performance record format 553
 - permanent line lock, TCAM 602
 - persistent session support
 - node error program 427
 - PGAICTLG, system initialization parameter 470
 - PGAEXIT, system initialization parameter 470
 - PGAIPGM, system initialization parameter 335, 470
 - PLT programs 338
 - PLTPI programs
 - general considerations 338
 - introduction 335
 - second stage 335
 - third stage 336
 - PLTPI, system initialization parameter 335
 - PLTSD programs
 - for first quiesce stage 337
 - for second quiesce stage 337
 - general considerations 338
 - introduction 337
 - sample program (DFH\$SDAP) 338
 - PLTSD, system initialization parameter 337
 - POOL feature, TCAM 601
 - pool of common TCTTEs 596
 - PRINT operand
 - DFHTEPM TYPE=INITIAL 373

- processing output from CICS statistics 588
- program error program (PEP)
 - communication area for assembler-language programs 353
 - link-edit statements 346
 - source code 353
 - writing 351
- program list table (PLT) programs
 - general considerations 338
 - PLTPI programs
 - introduction 335
 - second stage 335
 - third stage 336
 - PLTSD programs
 - for first quiesce stage 337
 - for second quiesce stage 337
 - introduction 337
 - sample 338
 - with storage protection
 - data storage key 339
 - execution key 339
- program management functions of the XPI 280
- programmable interface to RDO transactions 663
- programs, automatic installation of 467
- protocol management, TCAM 592
- PSERVIC values for automatic installation 679

Q

- queue considerations, TCAM 603
- queues for intersystem sessions
 - controlling the length of
 - using the XISCONA global user exit 77
 - using the XZIQUE global user exit 188

R

- RACROUTE macros 627
- RDO transactions
 - EXEC CICS LINK to DFHEDAP 663
 - programmable interface to 663
- reading DBCTL information from the CAVM data sets 514
- recovery and restart
 - node error program (DFHZNEP) 395
 - program error program (PEP) 351
 - routing mechanism (ACF/VTAM) 412
- recursive retry routine, in DFHTEP
 - example 392
- RELEASE PROGRAM function of the XPI 274
- RENTPGM, system initialization parameter 270, 289
- resource definition online transactions
 - EXEC CICS LINK to DFHEDAP 663
 - programmable interface to 663
- resource manager interface (RMI) 201

- restart-in-place function of the overseer program 503
- RESUME function of the XPI 256
- RMF (Resource Measurement Facility) 545
- RMI (resource manager interface) 201
- RMTRAN, system initialization parameter 428
- routing mechanism, VTAM 412

S

- sample programs
 - “good night” transaction (DFH0GNIT) 637
 - CICS–DBCTL interface status program (DFHDBUEX) 498
 - for automatic installation of APPC connections 456
 - for automatic installation of programs 472
 - for automatic installation of terminals 444
 - for dynamic allocation (DYNALLOC) 613
 - for dynamic transaction routing 492
 - for global user exits
 - DFH\$PCPI, description of 15
 - DFH\$XZIQ, for the XZIQUE exit 195
 - DFHXTENF, for the XALTENF exit 163
 - list of 14
 - for journaling 495
 - for the system definition utility program, DFHCSDUP 647
 - node error program (DFHSNEP) 411
 - PLT shutdown program (DFH\$SDAP) 338
 - program error program (DFHPEP) 355
 - terminal error program (DFHXTEP) 363
 - transaction restart program (DFHREST) 360
 - transient data write-to-terminal program (DFH\$TDWT) 713
 - schedule flag word 214
 - SCS (SNA character string) 592
 - security
 - changes to link security for MRO 633
 - early verification processing 629
 - interface to external manager 621
 - MVS router 621
 - MVS router exit 622
 - RACROUTE macros 627
 - system authorization facility (SAF) 621
 - the CICS–ESM interface 621
 - segment processing, TCAM 600
 - sequence of events, TCAM 597
 - service level reporter (SLR) 588
 - session failures, user actions 423
 - SET MONITOR command 546
 - SET_AUTOINSTALL function of the XPI 296
 - SET_PROGRAM function of the XPI 288
 - SET_SYSTEM function of the XPI 303
 - SET_TRANSACTION function of the XPI 329
 - SETEOM macro 594
 - shipped terminals, automatic installation of 459

- shutdown (PLTSD) programs
 - considerations when writing 337
- SLR (service level reporter) 588
- SMF (system management facility) 546
 - header 535, 547, 584
 - product section 535, 584
- SNA character string (SCS) 592
- specifying processing at EMPs
 - MCT entries for DBCTL 544
 - MCT entries for DL/I 543
- START_BROWSE_PROGRAM function of the XPI 291
- START_PURGE_PROTECTION function of the XPI 267
- startup, TCAM 607
- state data access functions of the XPI 297
- statistics
 - control commands 580
 - data section format 586
 - exception class data 579
 - global user exit 588
 - overview 579
 - performance class data 579
 - processing output from 588
 - purpose 579
 - record formats 583
 - record types 579
 - SLR (service level reporter) 588
 - SMF header 584
 - SMF product section 584
- STATRCD, system initialization parameter 579
- STOP_PURGE_PROTECTION function of the XPI 267
- storage control functions of the XPI 305, 310
- storage protection facility
 - with global user exit programs 11
 - with PLT programs 338
 - with task-related user exit programs 217
 - with user-replaceable programs 348
- stub program, for task-related user exits 201, 202
- SUSPEND function of the XPI 253
- SWITCH_SUBSPACE function of the XPI 312
- syncpoint management
 - module DFHDBP 229
 - syncpoint manager parameters 208
- syntax notation xii
- SYSEVENT monitoring data 545
- system autoinstall 468
- system definition utility program (DFHCSDUP)
 - See CICS system definition utility program (DFHCSDUP)
- SYSTEM DUMP function of the XPI 263
- system header, journal records 524
- system initialization parameters
 - AIEXIT 432, 452
 - AILDELAY 121, 581
 - AIRDELAY 428, 581

- system initialization parameters (*continued*)
 - CLSDSTP 410
 - DTRPGM 491
 - DTRTRAN 479
 - ESMEXITS 624
 - GMTRAN 408
 - GNTRAN 635, 639
 - INITPARMS 15
 - MN 546
 - MNEVE 546
 - MNEXC 546
 - MNPER 546
 - PGAICTLG 470
 - PGAIEEXIT 470
 - PGAIPGM 335, 470
 - PLTPI 335
 - PLTSD 337
 - RENTPGM 270, 289
 - RMTRAN 428
 - STATRCD 579
 - TBEXITS 13, 166
- system initialization table (DFHSIT)
 - entries for CICS monitoring 546
- system management facility (SMF) 546
 - header 535, 547
 - product section 535, 547
- system prefix, journal records 531
- system-defined event-monitoring point 540

T

- TACLE (terminal abnormal condition line entry)
 - address contents 387
 - DSECT, format description 388
 - terminal error program 363
- task manager parameters in task-related user exits 211
- task-related user exits 201
 - adapter
 - installing and withdrawing 229
 - responses to the caller 216
 - structure and components 201
 - addressability of the parameter list 205
 - addressing mode implications 217
 - administration 202, 229
 - application program parameters 208
 - caller parameter lists 208
 - CEDA 229
 - CICS termination calls 224
 - limitations 225
 - sample code 226
 - use of DFHEIENT 225
 - DFHEIENT macros 218
 - DFHUEPAR DSECT 205
 - DFHUERTR, function definition 208
 - DFHUEEXIT TYPE=RM macro 205

task-related user exits (*continued*)

- EDF 204
- enabling and disabling
 - EXEC CICS DISABLE command 230
 - EXEC CICS ENABLE command 230
- EXTRACT command 230
- global work area 219, 230
- local work area 219
- parameter lists 205
- PPT entries 229
- protocols
 - read-only 220
 - single-update 220
- recovery considerations 219
- sample code for CICS termination call 226
- sample code for syncpoint manager calls 221
- schedule flag word 214
- stub program 201, 202
 - ename 203
 - statname 203
- syncpoint manager calls 219, 223
 - backing out changes 223
 - committing changes 223
 - restart resynchronization 223
 - sample pseudocode 221
- syncpoint manager parameters 208
- table entries 229
- task manager calls 224
 - limitations 224
- task manager parameters 211
- UEPCALAM, address of the caller's AMODE indication byte 207
- UEPEIB, address of EIB 206
- UEPEXN, address of function definition 205
- UEPFLAGS, address of schedule flag word 206
- UEPGAA, address of global work area 206
- UEPGAL, length of global work area 206
- UEPHMSA, address of register save area 206
- UEPPBTOK, address of performance block token 208
- UEPRMQUA, address of the resource manager qualifier name 207
- UEPRMSTK, address of the kernel stack entry 206
- UEPSECBLK, address of a fullword addressing the user security block 207
- UEPSECFLG, address of the user security flag 207
- UEPSYNCA, address of the single-update indication byte 207
- UEPTAA, address of local work area 206
- UEPTAL, length of local work area 206
- UEPTIND, address of the caller's task indicators 207
- UEPUOWDS, address of the APPC unit of work identifier 206
- UEPURID, address of unit of recovery identifier 206
- UERTFGP, function group indicator 208

task-related user exits (*continued*)

- UERTFID, caller identifier 208
- using CICS commands 218
- using EDF 228
- with storage protection
 - data storage key 218, 339
 - execution key 217
 - work areas 219
- TBEXITS, system initialization parameter 13, 166
- TCAM (telecommunications access method)
 - 3270 606
 - abend and restart 608
 - application program 609
 - application program interface 594
 - attach TIOA 598
 - communication control bytes 604
 - data format 596
 - DD card correlation 595
 - default actions taken by DFHTACP 603
 - devices 604
 - generalized message format 605
 - input process queue 596
 - line locking 602
 - line pool restrictions 602
 - line pool specifications 601
 - logic flow 597
 - message control program (MCP) 594, 610
 - message handler 594
 - message routing 600
 - OPTCD operand 600
 - permanent line lock 602
 - POOL feature 601
 - pool of common TCTTEs 596
 - queue considerations 603
 - queue locks 603
 - read 598
 - segment processing 600
 - sequence of events 597
 - startup 607
 - temporary line lock 602
 - terminal abnormal condition program (DFHTACP) 361
 - terminal control program 361
 - terminal entries 596
 - terminal error program 600
 - terminal error program (DFHTEP) 361
 - termination 608
 - TPROCESS block 594
 - unsolicited input 603
 - user exits 607
- TCAM SNA 592
 - batch processing 594
 - communication control bytes 593
 - device message handler (DMH) 592
 - error processing 594
 - batch logical units 594

TCAM SNA (*continued*)
 FMH processing 593
 IEDRH macro 593
 protocol management 592
 SETEOM macro 594
 SNA character string (SCS) 592
 TCAMFET=SNA operand 592
 transaction control 593
 with CICS 592

TCP (terminal control program)
 ACF/VTAM section 403
 TACLE (terminal abnormal condition line entry) 362

TEB (terminal error block) 364

telecommunications access method (TCAM)
 See TCAM (telecommunications access method)

templates, for autoinstall of APPC connections 452

temporary line lock 602

TEP (terminal error program)
 abnormal conditions 362
 CICS components 361
 communication area 363
 address contents 385
 default table 365
 define terminal error blocks
 tables, DFHTEPT TYPE=PERMTID 379
 DFHTEP recursive retry routine 391
 example 392
 system count (TCTTENI) 391
 user field a (PCISAVE) 391
 user field b (PCICNT) 391
 DFHTEP tables 378
 DFHTEPM TYPE=ENTRY 375
 DFHTEPM TYPE=EXIT 375
 DFHTEPT TYPE=PERMCODE|ERRCODE 379
 error processor source 375
 error table 364
 errors and actions unique to TCAM 600
 generating 371
 job control for sample DFHTEP generation 371
 link-edit statements 346
 replace error processors, DFHTEPM
 TYPE=ERRPROC 375
 sample
 action flag names 369
 common subroutines 367
 components 364
 DECB information 370
 DECB operand 370
 DFHTEPM TYPE=INITIAL 371
 entry and initialization 366
 error processing execution 366
 error processor selection 366
 error status element (ESE) 365
 ESE information 370
 exit 367
 generate sample module 371
 messages 369

TEP (terminal error program) (*continued*)
 sample (*continued*)
 overview 366
 TACLE information 370
 terminal error block (TEB) 364
 terminal identification and error-code lookup 366

tables
 default threshold count limits 381
 DFHTEPT macro examples 383
 DFHTEPT TYPE=BUCKET 382
 DFHTEPT TYPE=INITIAL 378

TCAM 600
 terminal abnormal condition line entry (TACLE) 363
 user-written program
 abend-transaction bit 386
 abnormal conditions 384
 abort write bit 386
 address contents of communication area 385
 address contents of TACLE 387
 dummy terminal indicator 386
 example 391
 format description of TACLE DSECT 388
 nonpurgeable task 386

TERMERR condition 395

terminal abnormal condition line entry (TACLE) 363

terminal abnormal condition program (DFHTACP) 362

terminal control program (TCP)
 See TCP (terminal control program)

terminal entries, TCAM 596

terminal error block (TEB) 364

terminal error program (TEP)
 abnormal conditions 362
 CICS components 361
 communication area 363
 address contents 385
 default table 365
 define terminal error blocks
 tables, DFHTEPT TYPE=PERMTID 379
 DFHTEP recursive retry routine 391
 example 392
 system count (TCTTENI) 391
 user field a (PCISAVE) 391
 user field b (PCICNT) 391
 DFHTEP tables 378
 DFHTEPM TYPE=ENTRY 375
 DFHTEPM TYPE=EXIT 375
 DFHTEPT TYPE=PERMCODE|ERRCODE 379
 error processor source 375
 error table 364
 generating 371
 job control for sample DFHTEP generation 371
 link-edit statements 346
 replace error processors, DFHTEPM
 TYPE=ERRPROC 375
 sample
 action flag names 369
 common subroutines 367

terminal error program (TEP) (*continued*)
 sample (*continued*)
 components 364
 DECB information 370
 DECB operand 370
 DFHTEPM TYPE=INITIAL 371
 entry and initialization 366
 error processing execution 366
 error processor selection 366
 error status element (ESE) 365
 ESE information 370
 exit 367
 generate sample module 371
 messages 369
 overview 366
 TACLE information 370
 terminal error block (TEB) 364
 terminal identification and error-code lookup 366

tables
 default threshold count limits 381
 DFHTEPT macro examples 383
 DFHTEPT TYPE=BUCKET 382
 DFHTEPT TYPE=INITIAL 378

terminal abnormal condition line entry (TACLE) 363

user-written program
 abend-transaction bit 386
 abnormal conditions 384
 abort write bit 386
 address contents of communication area 385
 address contents of TACLE 387
 dummy terminal indicator 386
 example 391
 format description of TACLE DSECT 388
 nonpurgeable task 386

terminal identification and error-code lookup 366

terminals, automatic installation 431

termination, TCAM 608

TIME operand
 DFHSNET macro 419
 of DFHTEPT TYPE=PERMCODE|ERRCODE
 macro 380

time stamp, definition
 for monitoring 558

TPROCESS block 594

trace control functions of the XPI 313

trace table entries, global user exit interface 7

TRACE_PUT function of the XPI 313

transaction abends
 program error program (PEP) 351

transaction control
 TCAM SNA 593

TRANSACTION DUMP function of the XPI 264

transaction management functions of the XPI 314

transaction restart program (DFHREST) 357
 communications area 358
 default program 360

transaction restart program (DFHREST) (*continued*)
 introduction 357
 link-edit statements 346
 transactions suitable for restart 357
 when invoked 357

transaction routing
 See dynamic transaction routing

transaction-class error-handling routine 403, 425

transient data write-to-terminal program
 (DFH\$TDWT) 713

TRMIDNT operand
 DFHTEPT TYPE=PERMTID 379

TSO
 DFHCSDUP 653

U

unsolicited input
 TCAM 603

uppercase translation
 of national characters 715

user event-monitoring points 540

user exits
 See *also* global user exits
 See *also* resource manager interface
 See *also* task-related user exits
 DFHCSDUP 656
 global 3
 task-related 201
 TCAM 607

user journaling functions of the XPI 330

user prefix, journal records 532

user-replaceable programs 348
 assembling and link-editing 344
 DBCTL interface status program (DFHDBUEX) 497
 dynamic transaction routing program
 (DFHDYP) 477
 for automatic installation of APPC connections
 (DFHZATDY) 451
 for automatic installation of programs
 (DFHPGADX) 467
 for automatic installation of shipped terminals
 (DFHZATDX) 459
 for automatic installation of shipped terminals
 (DFHZATDY) 459
 for automatic installation of terminals
 (DFHZATDX) 431
 for journaling 537
 DFHXJCC 493
 DFHXJCO 493
 general rules 343
 node error program (DFHZNEP) 395
 program error program (DFHPEP) 351
 rewriting 343
 terminal error program (DFHTEP) 361
 testing with EDF 343

user-replaceable programs (*continued*)
 transaction restart program (DFHREST) 357
 with storage protection
 data storage key 349
 execution key 348
 user-supplied error processors, DFHSNEP
 TYPE=ERRPROC 417
 user-written node error programs 421

V

VSAM files
 backout of WRITE-ADD operations
 DFHDBP file error exit 49
 DFHFCCBP error exit 172
 VTAM (virtual telecommunications access method)
 See ACF/VTAM

W

work areas in task-related user exits 219
 WRITE JOURNAL DATA function of the XPI 331

X

XAKUSER, global user exit 23
 XALCAID, global user exit 152
 XALTENF, global user exit 159
 XBMIN, global user exit 25
 XBMOU, global user exit 25
 XDBDERR, global user exit 50
 XDBFERR, global user exit 48
 XDBIN, global user exit 47
 XDBINIT, global user exit 47
 XDLIPOST, global user exit 36
 XDLIPRE, global user exit 34
 XDSAWT, global user exit 33
 XDSBWT, global user exit 32
 XDTAD, global user exit 128
 XDTLC, global user exit 130
 XDTRD, global user exit 125
 XDUCLE, global user exit 43
 XDUOUT, global user exit 45
 XDUREQ, global user exit 39
 XDUREQC, global user exit 41
 XEIIN, global user exit 52
 XEIOU, global user exit 53
 XFCNREC, global user exit
 description 54
 parameter list and return codes 55
 XFCREQ, global user exit
 command parameter structure 56
 description 56
 example of use 63
 parameter list and return codes 64
 UEPCLPS parameter 57

XFCREQC, global user exit
 command parameter structure 56
 description 56
 example of use 63
 parameter list and return codes 65
 UEPCLPS parameter 57
 XFCSREQ, global user exit 68
 XFCSREQC, global user exit 71
 XGMTEXT, global user exit 76
 XICEREQ, global user exit
 command parameter structure 87
 example of use 95
 parameter list and return codes 84
 UEPCLPS parameter 88
 XICEREQC, global user exit
 command parameter structure 87
 example of use 95
 parameter list and return codes 86
 UEPCLPS parameter 88
 XICEXP, global user exit 83
 XICREQ, global user exit 82
 XICTENF, global user exit 161
 XISCONA, global user exit 77, 79
 XISLCLQ, global user exit 81
 XJCWB, global user exit 97
 XJCWR, global user exit 96
 XKCREQ, global user exit 139
 XLDELETE, global user exit 99
 XLDLOAD, global user exit 98
 XMEOUT, global user exit 101
 XMNOU, global user exit 104
 XPCABND, global user exit 118
 XPCFTCH, global user exit 113
 XPCHAIR, global user exit 115
 XPCREQ, global user exit
 command parameter structure 107
 description 106
 example of use 111
 parameter list and return codes 106
 UEPCLPS parameter 108
 XPCREQC, global user exit
 command parameter structure 107
 description 106
 example of use 111
 parameter list and return codes 107
 UEPCLPS parameter 108
 XPCTA, global user exit 116
 XPI (exit programming interface)
 dispatcher functions
 ADD SUSPEND 251
 CHANGE PRIORITY 262
 DELETE SUSPEND 257
 RESUME 256
 SUSPEND 253
 WAIT_MVS 258
 dump control functions
 SYSTEM DUMP 263

XPI (exit programming interface) *(continued)*
 dump control functions *(continued)*
 TRANSACTION DUMP 264
 format of an XPI call 234
 journaling function
 WRITE JOURNAL DATA 331
 kernel domain functions
 START_PURGE_PROTECTION 267
 STOP_PURGE_PROTECTION 267
 loader functions
 ACQUIRE PROGRAM 271
 DEFINE PROGRAM 268
 DELETE PROGRAM 275
 RELEASE PROGRAM 274
 mixing API and XPI calls 6, 717
 monitoring functions
 INQUIRE MONITORING DATA 279
 MONITOR 276
 overview 231
 program management functions
 END_BROWSE_PROGRAM 294
 GET_NEXT_PROGRAM 292
 INQUIRE_AUTOINSTALL 295
 INQUIRE_CURRENT_PROGRAM 286
 INQUIRE_PROGRAM 280
 SET_AUTOINSTALL 296
 SET_PROGRAM 288
 START_BROWSE_PROGRAM 291
 programming examples 240, 717
 state data access functions
 INQ_APPLICATION_DATA 297
 INQUIRE_SYSTEM 299
 SET_SYSTEM 303
 storage control functions
 FREEMAIN 308
 GETMAIN 306
 INQUIRE_ACCESS 309
 INQUIRE_ELEMENT_LENGTH 310
 INQUIRE_SHORT_ON_STORAGE 310
 INQUIRE_TASK_STORAGE 311
 SWITCH_SUBSPACE 312
 trace control function
 TRACE_PUT 313
 transaction management functions
 INQUIRE_DTRTRAN 314
 INQUIRE_MXT 315
 INQUIRE_TCLASS 317
 INQUIRE_TRANDEF 318
 INQUIRE_TRANSACTION 325
 SET_TRANSACTION 329
 XRCFCER, global user exit 170
 XRCINIT, global user exit 166, 167, 168
 XRCINPT, global user exit 168, 169
 XRCOPER, global user exit 170
 XRF (extended recovery facility)
 node error program 427

XRF (extended recovery facility) *(continued)*
 overseer program 501
 XRF overseer program
 customizing the sample program 516
 including the CEBT command 516
 loop or wait detection in the active 517
 DFH\$AXRO 505
 DFHOSD data set 505
 DFHWOSM macros 506
 FUNC=BUILD 507
 FUNC=CLOSE 508
 FUNC=DSECT 508
 FUNC=JJC 509
 FUNC=JJS 509
 FUNC=OPEN 510
 FUNC=OSCMD 511
 FUNC=QJJS 509
 FUNC=READ 512
 FUNC=TERM 515
 display function 502
 interface with CICS 505
 module DFHWOS 505
 opening CAVM data sets dynamically 505
 restart-in-place function 503
 enabling and disabling 504
 rules governing restart in place 504
 XRMIOU, global user exit 120
 XRMML, global user exit 119
 XRSINDI, global user exit 122
 XSNOFF, global user exit 132
 XSNON, global user exit 131
 XSRAB, global user exit 135
 XSTERM, global user exit 138
 XSTOUT, global user exit 133
 XSZARQ, global user exit 75
 XSZBRQ, global user exit 75
 XTCATT, global user exit 155, 607
 XTCIN, global user exit 154
 XTCOUT, global user exit 154
 XTCTIN, global user exit 155
 XTCTOUT, global user exit 156
 XTDEREQ, global user exit
 command parameter structure 180
 parameter list and return codes 178
 UEPCLPS parameter 181
 XTDEREQC, global user exit
 command parameter structure 180
 parameter list and return codes 179
 UEPCLPS parameter 181
 XTDIN, global user exit 175
 XTDOUT, global user exit 176
 XTDREQ, global user exit 175
 XTSEREQ, global user exit 143
 command parameter structure 145
 example program 150, 717
 UEPCLPS parameter 145

- XTSEREQC, global user exit 144
 - command parameter structure 145
 - example program 150, 717
 - UEPCLPS parameter 145
- XTSIN, global user exit 140
- XTSOUT, global user exit 141
- XTSREQ, global user exit 140
- XXDFA, global user exit 29
- XXDFB, global user exit 30
- XXDTO, global user exit 30
- XXMATT, global user exit 173
- XXRSTAT, global user exit 198
- XZCATT, global user exit 185
- XZCIN, global user exit 186
- XZCOUT, global user exit 186
- XZCOUT1, global user exit 187
- XZIQUE, global user exit 188, 192
 - designing your exit program 195
 - how to use 188
 - interaction with XISCONA 188
 - overview 188
 - the sample program, DFH\$XZIQ 195
 - using IRC/ISC statistics 195
 - when invoked 188

Sending your comments to IBM

CICS for MVS/ESA

Customization Guide

SC33-1165-03

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book. Please limit your comments to the information in this book and the way in which the information is presented.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, use the Readers' Comment Form
- By fax:
 - From outside the U.K., after your international access code use 44 1962 870229
 - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
 - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
 - IBMLink: HURSLEY(IDRCF)
 - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

CICS for MVS/ESA

Customization Guide

SC33-1165-03

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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

Be sure to print your name and address below if you would like a reply.

Name

Address

Company or Organization

Telephone

Email



You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

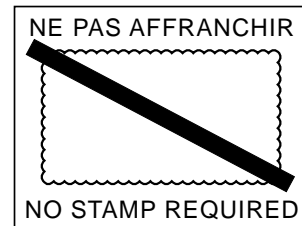
1 Cut along this line

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO



REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories Limited
Information Development Department (MP 095)
Hursley Park
WINCHESTER, Hants
SO21 2ZZ
United Kingdom

3 Fold along this line

From: Name _____
Company or Organization _____
Address _____

EMAIL _____
Telephone _____

1 Cut along this line

4 Fasten here with adhesive tape





Program Number: 5655-018



Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1165-03



Spine information:



CICS for MVS/ESA

Customization Guide

Version 4 Release 1