CICS® Transaction Server for OS/390®

CICS External Interfaces Guide

SC33-1944-01

CICS® Transaction Server for OS/390®

CICS External Interfaces Guide

SC33-1944-01

Note!

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

Second Edition (March 1999)

This edition applies to Release 3 of the IBM licensed program CICS Transaction Server for OS/390, program number 5655-147, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on this product.

This book is based on the following manual, which remains current for CICS Transaction Server for OS/390 Release 2:

- *CICS Internet and External Interfaces Guide*, SC33-1944

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

At the back of the 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,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN.

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

© Copyright International Business Machines Corporation 1994, 1998. All rights reserved.
US Government Users Restricted Rights – Use duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Notices

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

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

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

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

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact IBM United Kingdom Laboratories, MP151, Hursley Park, Winchester, Hampshire, England, SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

© Copyright IBM Corp. 1994, 1998

The licensed program described in this document and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Programming interface information

This book is intended to help you use the external interfaces provided by the CICS Transaction Server for OS/390. This book documents General-use Programming Interface and Associated Guidance Information provided by CICS.

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

This book also documents Product-sensitive Programming Interface and Associated Guidance Information and Diagnosis, Modification or Tuning 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.

Product-sensitive Programming Interface and Associated Guidance Information is identified, where it occurs, by an introductory statement to a chapter or section.

Diagnosis, Modification, or Tuning Information is provided to help you diagnose problems in your CICS system.

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

Diagnosis, Modification, or Tuning Information is identified, where it occurs, by an introductory statement to a chapter or section.

Trademarks

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

BookManager
C/370
CICS
CICS/ESA
DB2
IBM
IBMLink
IMS
MQ
MQSeries
MVS/ESA
OS/390
OpenEdition
RACF
VTAM

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and other countries.

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

Preface

What this book is about

This book describes how you can make the CICS® transaction processing services of CICS TS for OS/390® available to a variety of external users.

How to use this book

Read “Part 1. Overview” on page 1 for planning information, and for guidance about which other parts of the book to consult.

What you need to know to understand this book

This book assumes that you are familiar with CICS, either as a system administrator or as a system or application programmer. Some parts of the book assume additional knowledge about CICS and other products.

Notes on terminology

When the term “CICS” is used without any qualification in this book, it refers to the CICS element of IBM® CICS Transaction Server for OS/390.

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 usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

Subsequent updates will probably be available in softcopy before they are available in hardcopy. This means that at any time from the availability of a release, softcopy versions should be regarded as the most up-to-date.

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.

Updates to the softcopy are clearly marked by revision codes (usually a “#” character) to the left of the changes.

© Copyright IBM Corp. 1994, 1998

Contents

© Copyright IBM Corp. 1994, 1998

Bibliography

CICS Transaction Server for OS/390

CICS Transaction Server for OS/390: Planning for Installation	GC33-1789
CICS Transaction Server for OS/390 Release Guide	GC34-5352
CICS Transaction Server for OS/390 Migration Guide	GC34-5353
CICS Transaction Server for OS/390 Installation Guide	GC33-1681
CICS Transaction Server for OS/390 Program Directory	GI10-2506
CICS Transaction Server for OS/390 Licensed Program Specification	GC33-1707

CICS books for CICS Transaction Server for OS/390

General	
CICS Master Index	SC33-1704
CICS User's Handbook	SX33-6104
CICS Transaction Server for OS/390 Glossary (softcopy only)	GC33-1705
Administration	
CICS System Definition Guide	SC33-1682
CICS Customization Guide	SC33-1683
CICS Resource Definition Guide	SC33-1684
CICS Operations and Utilities Guide	SC33-1685
CICS Supplied Transactions	SC33-1686
Programming	
CICS Application Programming Guide	SC33-1687
CICS Application Programming Reference	SC33-1688
CICS System Programming Reference	SC33-1689
CICS Front End Programming Interface User's Guide	SC33-1692
CICS C++ OO Class Libraries	SC34-5455
CICS Distributed Transaction Programming Guide	SC33-1691
CICS Business Transaction Services	SC34-5268
Diagnosis	
CICS Problem Determination Guide	GC33-1693
CICS Messages and Codes	GC33-1694
CICS Diagnosis Reference	LY33-6088
CICS Data Areas	LY33-6089
CICS Trace Entries	SC34-5446
CICS Supplementary Data Areas	LY33-6090
Communication	
CICS Intercommunication Guide	SC33-1695
CICS Family: Interproduct Communication	SC33-0824
CICS Family: Communicating from CICS on System/390	SC33-1697
CICS External Interfaces Guide	SC33-1944
CICS Internet Guide	SC34-5445
Special topics	
CICS Recovery and Restart Guide	SC33-1698
CICS Performance Guide	SC33-1699
CICS IMS Database Control Guide	SC33-1700
CICS RACF Security Guide	SC33-1701
CICS Shared Data Tables Guide	SC33-1702
CICS Transaction Affinities Utility Guide	SC33-1777
CICS DB2 Guide	SC33-1939

© Copyright IBM Corp. 1994, 1998

CICSPlex SM books for CICS Transaction Server for OS/390

General	
CICSPlex SM Master Index	SC33-1812
CICSPlex SM Concepts and Planning	GC33-0786
CICSPlex SM User Interface Guide	SC33-0788
CICSPlex SM View Commands Reference Summary	SX33-6099
Administration and Management	
CICSPlex SM Administration	SC34-5401
CICSPlex SM Operations Views Reference	SC33-0789
CICSPlex SM Monitor Views Reference	SC34-5402
CICSPlex SM Managing Workloads	SC33-1807
CICSPlex SM Managing Resource Usage	SC33-1808
CICSPlex SM Managing Business Applications	SC33-1809
Programming	
CICSPlex SM Application Programming Guide	SC34-5457
CICSPlex SM Application Programming Reference	SC34-5458
Diagnosis	
CICSPlex SM Resource Tables Reference	SC33-1220
CICSPlex SM Messages and Codes	GC33-0790
CICSPlex SM Problem Determination	GC33-0791

Other CICS books

CICS Application Programming Primer (VS COBOL II)	SC33-0674
CICS Application Migration Aid Guide	SC33-0768
CICS Family: API Structure	SC33-1007
CICS Family: Client/Server Programming	SC33-1435
CICS Family: General Information	GC33-0155
CICS 4.1 Sample Applications Guide	SC33-1173
CICS/ESA 3.3 XRF Guide	SC33-0661

If you have any questions about the CICS Transaction Server for OS/390 library, see *CICS Transaction Server for OS/390: Planning for Installation* which discusses both hardcopy and softcopy books and the ways that the books can be ordered.

Summary of changes

This book is based on the *CICS Internet and External Interfaces Guide* for CICS Transaction Server for OS/390 Release 2. Changes from that edition are marked by vertical lines to the left of the changes.

Changes for this edition

The major changes to CICS that affect this book are:

- The addition of CICS support for MVS recoverable resource management services (RRMS) for applications that use the external CICS interface (EXCI). See “Part 3. External CICS Interface” on page 111.
- Enhancements to the 3270 Bridge function currently provided, including a new START command option to simplify the establishment of the Bridge environment, and relaxation of some restrictions.
- Program-link requests received from outside CICS can now be dynamically routed. See “Appendix. Routing program-link requests” on page 335.

Changes to the book for this edition are:

- The parts have been reordered as follows:
  - “Part 2. Bridging to 3270 transactions” on page 19
  - “Part 3. External CICS Interface” on page 111
  - “Part 4. CICS ONC RPC support” on page 227
  - “Part 5. Using CICS as a DCE server” on page 319
- “Chapter 1. Introduction” on page 3 has been expanded to include general information from other parts of the book.

© Copyright IBM Corp. 1994, 1998

Part 1. Overview

This part of the book outlines some the ways in which you can make CICS transaction processing services available to a variety of external users.

This part contains:

© Copyright IBM Corp. 1994, 1998

Chapter 1. Introduction

This book describes the following sources of external requests, and the routes that they can use into CICS:

MQSeries® users
MQSeries users can use the 3270 CICS bridge to access CICS transactions. See “Part 2. Bridging to 3270 transactions” on page 19

MVS™ applications
Applications running in MVS address spaces can use the External CICS Interface (EXCI) to access CICS programs. See “Part 3. External CICS Interface” on page 111.

ONC RPC clients
ONC RPC clients can use CICS ONC RPC support to access CICS programs. See “Part 4. CICS ONC RPC support” on page 227

DCE RPC clients
DCE RPC clients use the Application Support (AS) server to access CICS programs. See “Part 5. Using CICS as a DCE server” on page 319.

The following types of external requests are described in other books:

User socket applications
User socket applications can use the CICS Sockets feature of CICS TS. See *TCP/IP for MVS Version 3.1 CICS TCP/IP Socket Interface Guide and Reference*.

Web browsers
Web browsers can use a variety of methods:

CICS Web support
The CICS Web support is a CICS-provided facility for supporting Web browsers. See the *CICS Transaction Server for OS/390 CICS Internet Guide.*

IBM Websphere
The IBM Websphere Application Server for OS/390 is an MVS application that supports Web browsers and routes their requests into CICS.

CICS Transaction Gateway
The CICS Internet Gateway is a workstation application that can accept requests from Web browsers and route them into CICS. It uses a CICS client and the EPI.

JVM applications
Java Virtual Machine applications can use a local gateway connection that uses the EXCI to pass requests to CICS. See *CICS Transaction Server for OS/390 CICS Internet Guide*.

Java-enabled Web browsers
Java-enabled Web browsers can use applets that communicate with CICS. Writers of applets can use CICS-provided Java classes to construct external call interface (ECI) and external presentation interface (EPI) requests. The Web browsers communicate with Web servers, and with one of the following:

CICS Transaction Gateway
The CICS Transaction Gateway, a workstation application that uses a CICS client to route ECI and EPI requests to a CICS server.

© Copyright IBM Corp. 1994, 1998

CICS Transaction Gateway (MVS)

The CICS Transaction Gateway (MVS), a version of the CICS Transaction Gateway that runs in an MVS address space, and uses the CICS external CICS interface (EXCI) to pass requests to CICS. The CICS Transaction Gateway (MVS) supports the use of ECI requests, but not EPI requests. See the *CICS Transaction Server for OS/390 CICS Internet Guide*.

CICS client applications

CICS client applications use a CICS client and the ECI or the EPI. See *CICS Family: Client/Server Programming*.

CICS programs

Programs running in CICS Servers on any platform can use EXEC CICS LINK to call a CICS program, or can use transaction routing to send transaction requests to CICS TS. Programs running in CICS TS can use the CICS front end programming interface (FEPI) to start transactions in the same or another instance of CICS TS. See *CICS Front End Programming Interface User's Guide*.

Telnet clients

Telnet clients can use TN3270 to start transactions.

3270 users

Users of the IBM 3270 Display System can start transactions. This is the most familiar method of introducing work to CICS TS.

Figure 2 on page 5 shows the principal ways of using CICS transaction processing services from outside CICS.

Key to figure 2

TC	=	Terminal Control
TR	=	Transaction Routing
DPL	=	Distributed Program Link
EXCI	=	EXternal CICS Interface
ECI	=	External Call Interfaces
EPI	=	External Presentation Interface
CWP	=	CICS WebServer Plugin
⬭	=	Sources of external requests
▨	=	Targets of external requests
⬚	=	CICS provided interfaces
■	=	CICS components
□	=	Other product components

Figure 1.

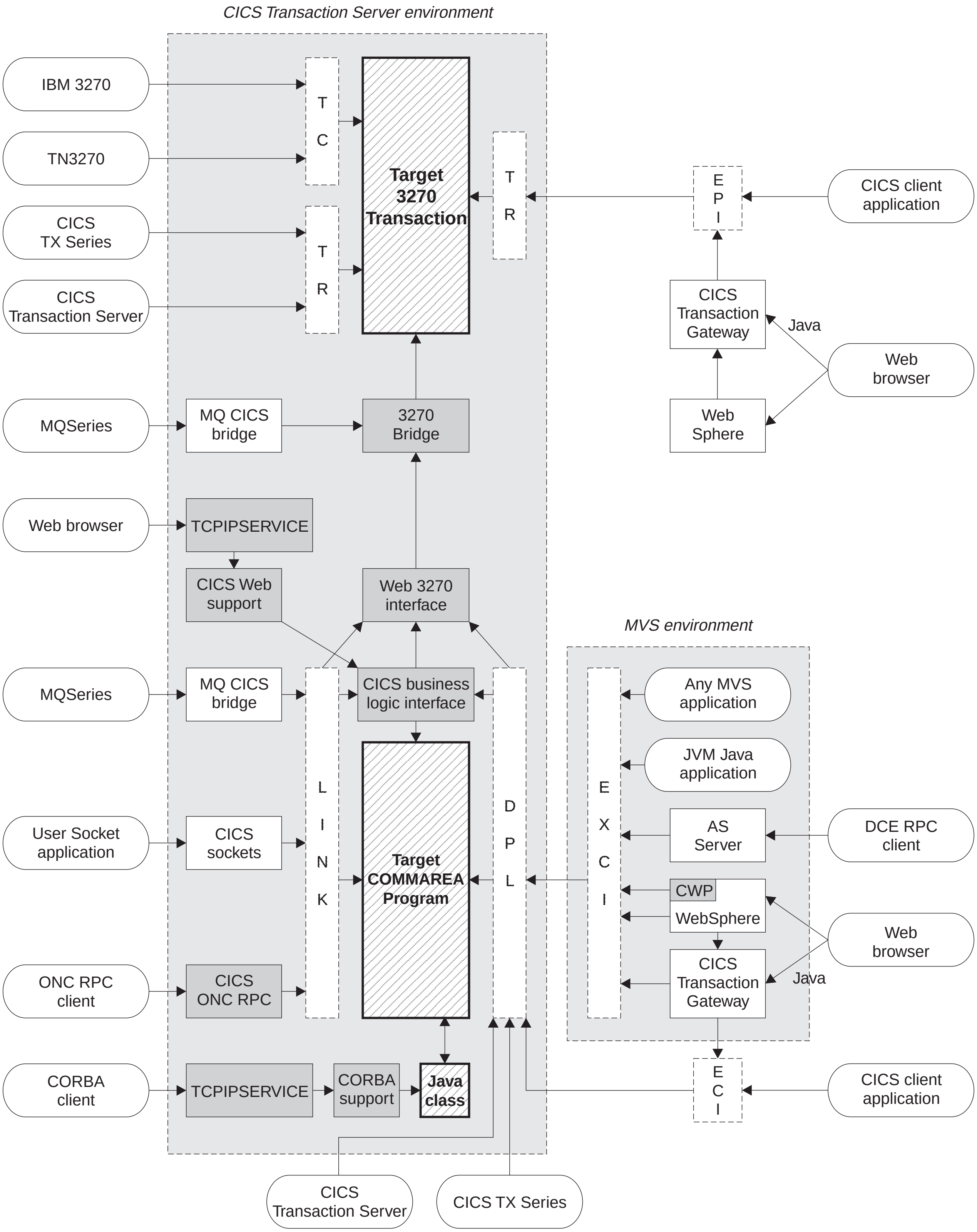


Figure 2. Client access to existing business logic

General concepts

All the mechanisms described in this book follow a similar pattern. A client is the source of the external request which comes into CICS over a network using a variety of transport protocols, or from another CICS region, using Inter Region Communication (IRC). CICS (or another product) provides a transport-specific listener (a long-running task) that starts another task (a facilitator such as an **alias** or a **mirror**), to process the incoming request. The facilitator uses CICS services to access the application.

The priorities of different alias transactions can be adjusted to determine the service that a client request receives. There must be enough free tasks to service the alias transactions as they are started by the listener. The CICS programs that service the client requests are subject to contention for resources in the CICS system, and to transmission delays if they are remote from the CICS system, or if they request the use of remote resources by function shipping or distributed program link.

The CICS server is independent of the application model (2/3-tier, 2/3 platforms). The listener/facilitator deals with the different transports used and sets the rules for which programming models are supported.

Distributed computing

Distributed computing involves the cooperation of two or more machines communicating over a network. The machines participating in the system can range from personal computers to super computers; the network can connect machines in one building or on different continents.

The main benefit of distributed computing is that it enables you to optimize your computing resources for both responsiveness and economy. For example, it enables you to:

- Share the cost of expensive resources, such as a typesetting and printing service, across many desktops. It also gives you the flexibility to change the desktop-to-server ratio, depending on the demand for the service.
- Allocate an application's presentation, business, and data logic appropriately. Often, the desktop is the best place to perform the presentation logic, as it is nearest to the end user and can provide highly responsive processing for such actions as drag and drop GUI interfaces.

  Conversely, you may feel that the best place for the database access logic is close to the actual storage device - that is, on an enterprise or departmental server. The most appropriate place for the business logic may be less clear, but there is much to be said for placing this too in the same node as the data logic, thus allowing a single desktop request to initiate a substantial piece of server work without intervening network traffic.

  Distributed computing enables you to make such trade-offs in a flexible way.

Along with the advantages of distributed computing come new challenges. Examples include keeping multiple copies of data consistent, keeping clocks in individual machines synchronized, and providing network-wide security. A system that provides distributed computing support must address these new issues.

CICS supports distributed computing and the client/server model by means of:

Internet Inter-Orb Protocol (IIOP)
CORBA clients can access CICS Java servers using IIOP.

Distributed Computing Environment (DCE)
The remote procedure call model implemented by the Open Software Foundation's DCE is supported in CICS.

Distributed program link (DPL)
This is similar to a DCE remote procedure call. A CICS client program passes parameters to a remote CICS server program and waits for the server to send data in reply. Parameters and data are exchanged by means of a communications area.

The external CICS interface (EXCI)
An MVS client program links to a CICS server program. Again, this is similar to a DCE RPC.

The external call interface (ECI)
The ECI enables CICS Transaction Server for OS/390 server programs to be called from client programs running on a variety of operating systems. For information about CICS Clients, see the *CICS Family: Client/Server Programming* manual.

Function shipping
The parameters for a single CICS API request are intercepted by CICS code and sent from the client system to the server. The CICS mirror transaction in the server executes the request, and returns any reply data to the client program. This can be viewed as a specialized form of remote procedure call.

Asynchronous transaction processing
A CICS client transaction uses the EXEC CICS START command to initiate another CICS transaction, and pass data to it. The START request can be intercepted by CICS code, and function shipped to a server system. The client transaction and started transactions execute independently. This is similar to a remote procedure call with no response data.

Distributed transaction processing
A program in the client system establishes a conversation with a complementary program in the server, and exchanges messages. The programs may use the APPC protocols.

Transaction routing
Terminals owned by one CICS system to run transactions owned by another.

The CICS family of products runs on a variety of operating systems, and provides a standard set of functions to enable members to communicate with each other. For information about the CICS family, see the *CICS Family: Interproduct Communication* manual.

Security support

CICS Transaction Server for OS/390 supports:

- A single network signon (through the ATTACHSEC option of the DEFINE CONNECTION command)
- Authentication of the client system through bind-time security.

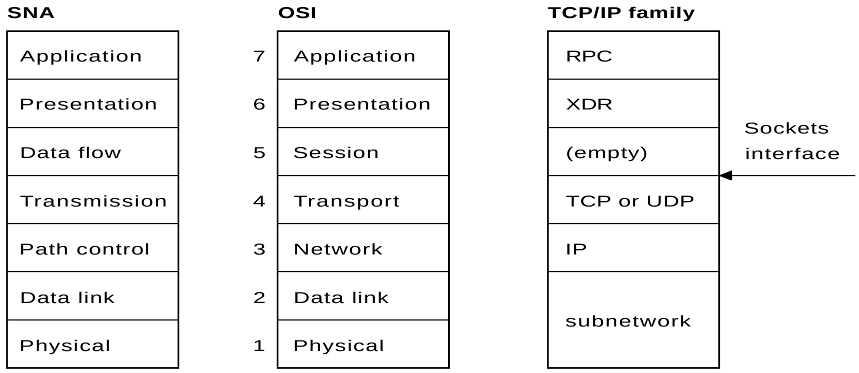
RACF or an equivalent security manager provides mechanisms similar to the DCE access control lists and login facility.

There is no CICS concept similar to the DCE Directory Service. In all the above scenarios the client environment must know which server CICS system to communicate with. This is normally done by specifying the name of the required remote CICS system in the definition of the relevant remote CICS resource, or in the client application program.

TCP/IP protocols

TCP/IP is a communication protocol used between physically separated computer systems. TCP/IP can be implemented on a wide variety of physical networks.

TCP/IP is a large family of protocols that is named after its two most important members, Transmission Control Protocol and Interface Protocol. Figure 3 shows the TCP/IP protocols used by CICS ONC RPC in terms of the layered Open Systems Interconnection (OSI) model. For CICS users, who may be more accustomed to SNA, the left side of Figure 3 shows the SNA layers that correspond very roughly to the OSI layers.



SNA		OSI	TCP/IP family
Application	7	Application	RPC
Presentation	6	Presentation	XDR
Data flow	5	Session	(empty)
Transmission	4	Transport	TCP or UDP
Path control	3	Network	IP
Data link	2	Data link	subnetwork
Physical	1	Physical	

Figure 3. TCP/IP protocols compared to the OSI and SNA models

The protocols used by TCP/IP are shown in the right-hand box in Figure 3.

Internet Protocol (IP)
In terms of the OSI model, IP is a network-layer protocol. It provides a *connectionless* data transmission service, and supports both TCP and UDP. Data is transmitted link by link; an end-to-end connection is never set up during the call. The unit of data transmission is the *datagram*.

Transmission Control Protocol (TCP)
In terms of the OSI model, TCP is a transport-layer protocol. It provides a *connection-oriented* data transmission service between applications, that is, a connection is established before data transmission begins. TCP has more error checking that UDP.

User Datagram Protocol (UDP)
UDP is also a transport-layer protocol and is an alternative to TCP. It provides a connectionless data transmission service between applications. UDP has less error checking than TCP. If UDP users want to be able to respond to errors, the communicating programs must establish their own protocol for error handling. With high-quality transmission networks, UDP errors are of little concern.

ONC RPC and XDR

XDR and ONC RPC correspond to the sixth and seventh OSI layers.

Sockets interface

The interface between the fourth and higher layers is the *sockets* interface. In some TCP/IP implementations, the sockets interface is the API that customers use to write their higher-level applications.

TCP/IP internet addresses and ports

TCP/IP provides for process-to-process communication, which means that calls need an addressing scheme that specifies both the physical host connection (Host A and Host B in Figure 4) and the software process or application (C, D, E, F, G, and H). The way this is done in TCP/IP is for calls to specify the host by an *internet address* and the process by a *port number*. You may find internet addresses also referred to elsewhere as internet protocol (IP) addresses or host IDs.

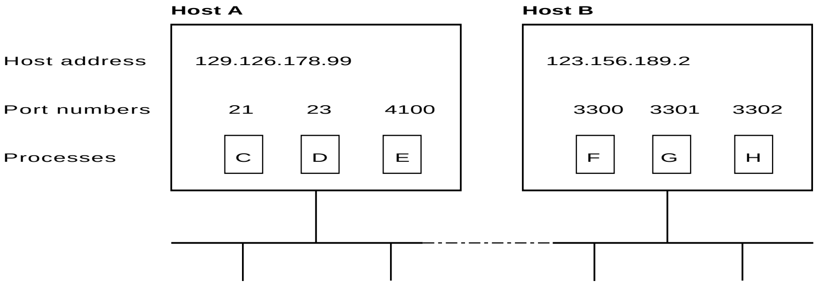


Figure 4. How applications are addressed

Internet addresses

Each host on a TCP/IP internet is identified by its internet address. An internet address is 32 bits, but it is usually displayed in dotted decimal notation. Each byte is converted to a decimal number in the range 0 to 255, and the four numbers are separated by dots thus: 129.126.178.99.

Remember that an internet is a collection of networks — hence the internet address must specify both the network and the individual host. How this is done varies with the size of the network. In the example just given, 129.126 specifies the network, 178.99 specifies the host on that network.

Port numbers (for servers)

An incoming connection request specifies the server that it wants by specifying the server’s port number. For instance, in Figure 4, a call requesting port number 21 on host A is directed to process C.

Well-known ports identify servers that carry standard services such as the File Transfer Protocol (FTP) or Telnet. The same service is always allocated the same port number, so, for example, FTP is always 21 and Telnet always 23. Networks generally reserve port numbers 1 through 255 for well-known ports.

Port numbers (for clients)

Client applications must also identify themselves with port numbers so that server applications can distinguish different connection requests. The method of allocating client port numbers must ensure that the numbers are unique; such port numbers are termed *ephemeral port numbers*. For example, in Figure 4 on page 9, process F is shown with port number 3300 on host B allocated.

Programming models

The programming models implemented in CICS are inherited from those designed for 3270s, and exhibit many of the characteristics of conversational, terminal-oriented applications. There are basically three styles of programming model:

- Terminal-initiated, that is, the conversational model
- Distributed program link, that is, the RPC model
- START, that is, the queuing model.

Once initiated, the applications typically use these and other methods of continuing and distributing themselves, for example, with pseudoconversations, RETURN IMMEDIATE or DTP. The main difference between these models is in the way that they maintain state (for example, security), and hence state becomes an integral part of the application design. This presents the biggest problem when you attempt to convert to another application model.

A pseudoconversational model is mostly associated with terminal-initiated transactions and was developed as an efficient implementation of the conversational model. With increased use of 1-in and 1-out protocols such as HTTP, it is becoming necessary to add the pseudoconversational characteristic to the RPC model.

State management and its associated token management, which were previously controlled by the terminal, now need additional techniques to support this move. Similarly, when START requests are disassociated from the terminal, difficulties arise in returning the requests to their starting point.

Comparing mechanisms

This section contrasts the different mechanisms by listing some of the characteristics and benefits of each interface.

ONC and DCE

DCE RPC is different from ONC RPC in many ways. For example, DCE RPC does not limit the number of parameters on the call, whereas an ONC RPC call is limited to one input and one output parameter (but these may be structures that contain many fields, including pointers to other data).

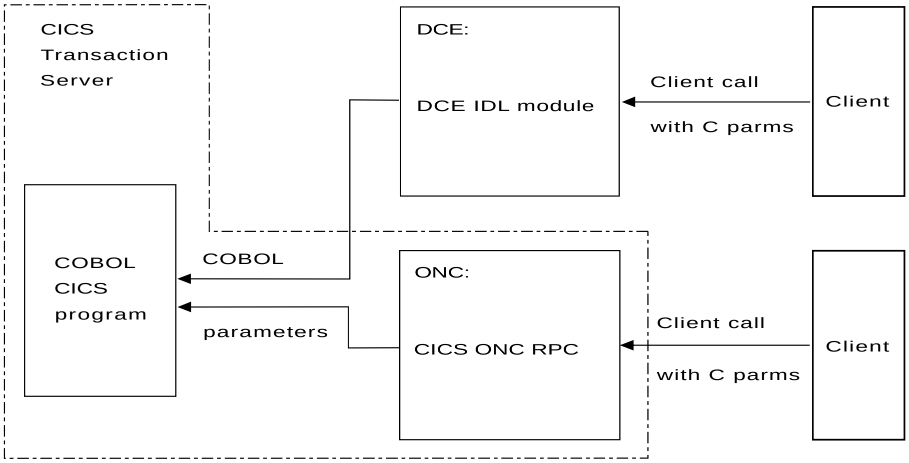


Figure 5. Remote procedures provided for DCE RPC and ONC RPC

Figure 5 shows how the two CICS RPC implementations provide the same function.

CICS programs that act as servers for DCE RPC must be written in VS COBOL II. You provide a definition of the client's parameter list in the interface definition language (IDL) provided as a part of DCE RPC. The DCE IDL module maps the incoming parameters into a CICS communication area, so the communication area format is defined by the client's parameter list.

CICS ONC RPC CICS programs can be written in any CICS-supported programming language, and the conversion from client format to communication area is done by the **Decode** function of the converter. With ONC RPC you get more flexibility, but you have more work to do.

CICS programs that are used as servers for DCE RPC clients can also be used as servers for ONC RPC clients. You need to write a **Decode** function that converts the incoming data structure into the predefined communication area, and converts the incoming data from C types to VS COBOL II types.

DCE

DCE provides a high-level, coherent environment for developing and running applications on a distributed system. The DCE components fall into two categories: tools for developing distributed applications and services for running them. The tools, such as Remote Procedure Calls and Threads, assist in the development of an application. The services, like the Directory Service, Security Service, and Time Service, provide support in a distributed system that is analogous to the support an operating system provides in a centralized system.

DCE includes management tools for administering all of the DCE services and many aspects of the distributed environment itself.

DCE is oriented towards heterogeneous rather than homogeneous systems. The DCE architecture allows for different operating systems and hardware platforms. Using DCE, a process running on one computer can interoperate with a process on a second computer, even when the two computers have different hardware or operating systems.

DCE is oriented towards heterogeneous rather than homogeneous systems. The DCE architecture allows for different operating systems and hardware platforms. Using DCE, a process running on one computer can interoperate with a process on a second computer, even when the two computers have different hardware or operating systems.

EXCI

The external CICS interface makes CICS applications more easily accessible from non-CICS environments.

Programs running in MVS can issue an EXEC CICS LINK PROGRAM command to call a CICS application programs running in a CICS region. Alternatively, the MVS programs can use the CALL interface when it is more appropriate to do so.

The provision of this programming interface means that, for example, MVS programs can:

- Update resources with integrity while CICS is accessing them.
- Take CICS resources offline, and back online, at the start and end of an MVS job. For example, you can:
  - Open and close CICS files.
  - Enable and disable transactions in CICS (and so eliminate the need for a master terminal operator during system backup and recovery procedures).

The external CICS interface opens up a new way to implement client/server applications, where the client program in a non-CICS environment calls a server program running in the CICS address space. The external CICS interface benefits not only TSO and batch applications, but allows you to extend the use of CICS application programs in an open client/server environment.

Although the CICS external interface operates over CICS MRO links, the client program can run on non-MVS platforms, and pass requests to CICS over an open system interface (OSI) using the IBM OpenEdition® Distributed Computing Environment Application Support MVS/ESA CICS feature (OE DCE AS/CICS). In this way the external CICS interface provides an open interface to a wide variety of other application platforms.

The 3270 bridge

The 3270 bridge allows you to introduce new GUI front ends to access existing 3270-based CICS applications without modifying them. This means that you can concentrate your efforts on the new user interfaces and avoid, or at least postpone, rewriting stable mainframe applications. You do not need to restructure your applications to separate the business logic from the presentation logic; the bridge effectively does this for you.

The same applications can be used both by 3270 terminals, and by the new client applications. This allows a phased migration of users from the 3270 applications to the new client applications. Applications written for 3270 terminals can be run on CICS systems without VTAM.

The bridge can process commands faster than existing front-end methods, such as FEPI and EPI, because the terminal emulation is part of the same CICS transaction. Unlike other front-end methods, there is only a single unit of work. This means that the bridge can use a recoverable MQSeries queue. This greatly simplifies recovery.

For BMS user transactions, there is no need to convert BMS data to 3270 format, because the client application receives the BMS Application Data Structure, rather than a 3270 datastream. This provides an easier method for the application programmer to interface with the user transaction compared to FEPI. A utility program (DFHBMSUP) is provided to recreate map source code from existing load modules, so that installations that do not have access to the original source code can still exploit the new ADS descriptor provided by the BMS macros.

The 3270 Bridge and FEPI

To help you decide between the 3270 bridge technology and FEPI, the following table summarizes the major characteristics.

Table 1. Comparision between 3270 bridge technology and FEPI

Bridge	FEPI
Enabling technology	An application programming interface
Based on application data structure	Based on the 3270 data stream
Enables optimization due to integral knowledge of the target	Easier to create generic driver (data structure is architected)
Efficient; no terminal control involved	VTAM managed connection between source and target
Single COMMAREA API and user replaceable program	Requires system programming and VTAM skills
CICS specific: source and target must be in the same region	Ideal for driving remote applications, not just CICS
Driven exit decides method of communication with the client	Can be freed from the workings of the target; terminal emulation
Knowledge of UOW	No coordination
Ideal when the routing is done elsewhere	Sysplex support requires three regions
Available only for CICS Transaction Server for OS/390 Release 2 and later	Available for CICS/ESA 3.3 and later

Application design

You can access existing applications originally designed for other environments by using the bridging facilities described, or write new ones specifically for a new environment. In general, it is good practice to split applications into a part containing the business code that is reusable, and a part responsible for presentation to the client. This technique enables you to improve performance by optimizing the parts separately, and allows you to reuse your business logic with different forms of presentation.

When separating the business and presentation logic, you need to consider the following:

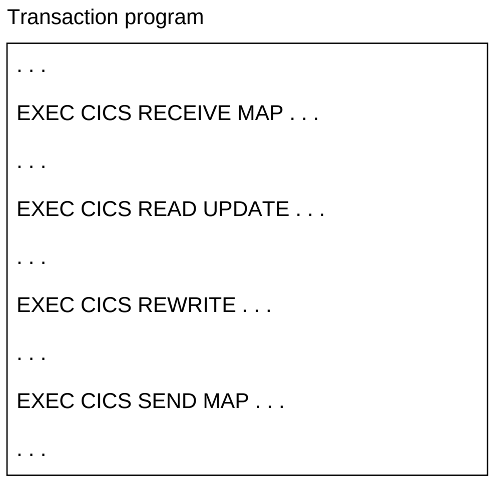
- Avoid affinities between the two parts of the application.
- Be aware of the DPL-restricted API; see *CICS Application Programming Reference* for details.

- Be aware of hidden presentation dependencies, such as EIBTRMID usage.

Separating business and presentation logic

Figure 6 illustrates a simple CICS application that accepts data from an end user, updates a record in a file, and sends a response back to the end user. The transaction that runs this program is the second in a pseudoconversation. The first transaction has sent a BMS map to the end user’s terminal, and the second transaction reads the data with the EXEC CICS RECEIVE MAP command, updates the record in the file, and sends the response with the EXEC CICS SEND MAP command.

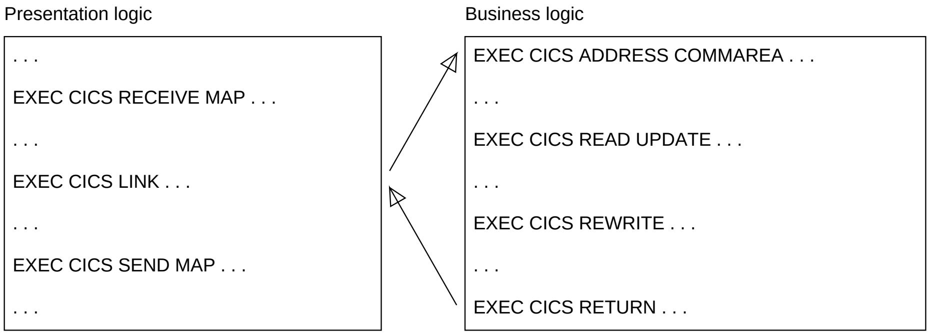
The EXEC CICS RECEIVE and EXEC CICS SEND MAP commands are part of the transaction’s presentation logic, while the EXEC CICS READ UPDATE and EXEC CICS REWRITE commands are part of the business logic.



```
. . .
EXEC CICS RECEIVE MAP . . .
. . .
EXEC CICS READ UPDATE . . .
. . .
EXEC CICS REWRITE . . .
. . .
EXEC CICS SEND MAP . . .
. . .
```

Figure 6. CICS functions in a single application program

A sound principle of modular programming in CICS application design is to separate the presentation logic from the business logic, and to use a communication area and the EXEC CICS LINK command to make them into a single transaction. Figure 7 on page 15 illustrates this approach to application design.



```
. . .
EXEC CICS RECEIVE MAP . . .
. . .
EXEC CICS LINK . . .
. . .
EXEC CICS SEND MAP . . .
. . .
```

```
EXEC CICS ADDRESS COMMAREA . . .
. . .
EXEC CICS READ UPDATE . . .
. . .
EXEC CICS REWRITE . . .
. . .
EXEC CICS RETURN . . .
```

Figure 7. Separation of business and presentation logic

Once the business logic of a transaction has been isolated from the presentation logic and given a communication area interface, it is available for reuse with different presentation methods. For example, you could use CICS Web support with the CICS business logic interface, to implement a two-tier model where the presentation logic is HTTP-based.

Chapter 2. How to use this book

The rest of the manual is organized as follows:

- “Part 2. Bridging to 3270 transactions” on page 19 describes how to use the transaction bridging facilities.
- “Part 3. External CICS Interface” on page 111 describes how to use the CICS EXCI.
- “Part 4. CICS ONC RPC support” on page 227 describes support for ONC RPC clients.
- “Part 5. Using CICS as a DCE server” on page 319 describes the use of the Application Support server with DCE RPC clients.

© Copyright IBM Corp. 1994, 1998

Part 2. Bridging to 3270 transactions

This part of the book describes the 3270 bridge. It covers the following topics:

- “Chapter 3. Introduction” on page 21
- “Chapter 4. The Bridge environment” on page 41
- “Chapter 5. Supplied 3270 bridge exits” on page 57
- “Chapter 6. Writing your own bridge programs” on page 77
- “Chapter 7. Problem determination” on page 107

© Copyright IBM Corp. 1994, 1998

Chapter 3. Introduction

This part of the book describes the function that allows you to run CICS 3270-based transactions without a 3270 terminal. It covers the following topics:

- “Overview”
- “Running 3270 transactions in a bridge environment” on page 29
- “Implementing a 3270 bridge environment” on page 32
- “The 3270 bridge” on page 12

Overview

The 3270 bridge provides an interface so that you can run 3270-based CICS transactions without a 3270 terminal. Commands for the 3270 terminal are intercepted by CICS and replaced by a messaging mechanism that provides a bridge between the end-user and the CICS transaction.

With the bridge feature, a **client** application that may be executing outside the CICS environment can use transport mechanisms such as MQSeries or the Internet to access and run a CICS 3270-based **user transaction**.

The client application can also be a CICS transaction, using, for example, a temporary storage queue to pass 3270 requests and data to a user transaction executing in the same CICS region. This provides an similar function to the FEPI interface.

The user transaction can be an existing 3270 or BMS-based CICS transaction. It runs unchanged as if it were being driven by a real terminal.

The following diagram shows the flow of a client request in a bridge environment. The client application requests execution of 3270 transactions by sending a **message** to a **bridge monitor** transaction , over a **transport mechanism** (which can be any method supported by CICS, including the Web, MQ, CICS BTS, and CICS transient data or temporary storage queues). “Components of the 3270 bridge” on page 22 describes all the components of the bridge environment shown in this diagram.

© Copyright IBM Corp. 1994, 1998

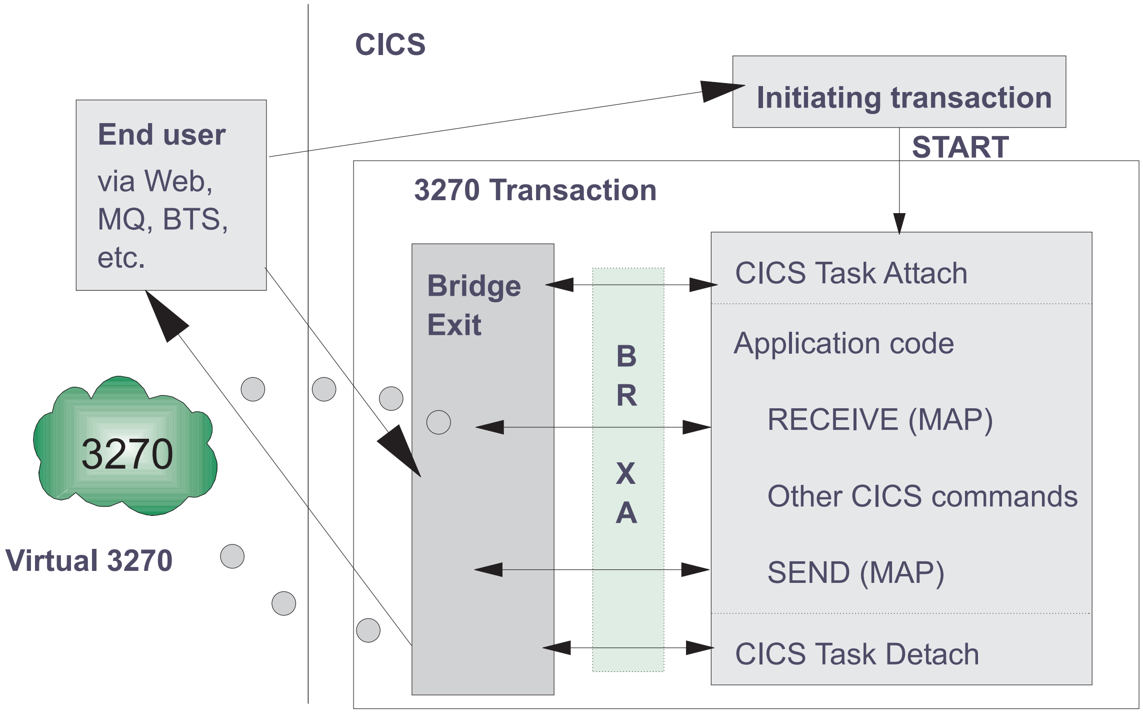


Figure 8. The 3270 bridge environment

Components of the 3270 bridge

Before you plan to use the 3270 bridge, you need to understand the following terms:

- The bridge mechanism
- The user transaction
- The client application
- The transport mechanism
- Bridge messages
- The bridge monitor
- The bridge environment
- The bridge exit
- The bridge exit area
- The bridge facility
- The FACILITYLIKE definition

The CICS 3270 bridge mechanism
The CICS 3270 bridge mechanism is the CICS function that allows a user transaction to be run without a VTAM 3270 terminal. Terminal input/output commands are intercepted by a ***bridge exit*** that emulates the terminal by passing the commands, packaged as ***messages*** to the end-user or ***client application***.

The user transaction
A user transaction is a 3270 CICS transaction.

The client application

A client application is a program, usually executing outside CICS, and possibly outside MVS/ESA, that uses the CICS 3270 bridge mechanism to run a user transaction.

The transport mechanism

A transport mechanism is used by the client application to pass ***messages*** to CICS. MQSeries; the Web; CICS BTS and CICS temporary storage and transient data, are all examples of transport mechanisms. Some transport mechanisms have separately definable queues.

Messages

A message contains information that provides all or part of the data needed to run a 3270 user transaction. Data originally written to the 3270 screen by the user transaction is packaged into messages and sent to the client application. Data originally read from the 3270 screen by the user transaction is obtained from messages sent by the client application.

The bridge monitor

A bridge monitor is usually a long-running CICS task that is associated with a specific transport mechanism, or a specific queue in a transport mechanism. When a message arrives requesting a CICS user transaction, the bridge monitor issues a START BREXIT TRANSID command to start the requested user transaction in a ***bridge environment*** where it will execute in association with the specified ***bridge exit***.

The bridge monitor must ensure that the requested transaction is started, and started only once. It can receive confirmation messages from the bridge exit after a successful start.

A bridge monitor can be is designed as a long running CICS task, to start any user transaction, or it can be designed to start only a single transaction.

The bridge environment

The bridge environment is established by CICS so that the ***user transaction*** can be executed and certain commands intercepted. A ***bridge exit*** and a ***bridge facility*** are essential components of the bridge environment.

The bridge exit

A bridge exit is a user-replaceable program that emulates the 3270 terminal API.

It does this by packaging data originally intended for the 3270 screen into messages and passing them to the transport mechanism, for delivery to the client application. Response messages from the client application are returned to the user transaction to satisfy terminal requests.

A bridge exit is usually designed to work with a specific transport mechanism.

A bridge exit can be generic if it is designed to run any user transaction, or specific if it is designed to work with a single user transaction.

In CICS Transaction Server for OS/390 Release 2, all 3270 terminal API requests were passed to the bridge exit. This provided a simple interface for specific bridge exits, but is very complicated in the generic case.

To reduce the complexity, the bridge exit can be designed to handle some of the requests, with all the terminal API requests being passed to another user-replaceable program, known as a **formatter**.

If used, the name of the formatter is obtained from the BRXA_FORMATTER field in the BRXA and the bridge exit is only called for requests that require input or output of data. If a formatter is not specified, the bridge exit is called for all requests.

The bridge exit is always called for the following requests:

- User transaction initialization
- User transaction bind
- User transaction termination
- User transaction abnormal termination
- Read and Write message
- Syncpoint (optional)

A formatter is called for the following requests:

- SEND (Terminal Control and BMS)
- RECEIVE (Terminal Control and BMS)
- CONVERSE
- FREE
- ISSUE DISCONNECT
- ISSUE ERASEAUP
- RETRIEVE (in some cases)

All requests made in a bridge exit are run as part of the same unit of work as the user transaction. Therefore, any recoverable requests made by the bridge exit are committed or rolled back at the same time as the user transaction resources.

The abend termination handler in the bridge exit is called when the user transaction terminates abnormally, so that the client application can be informed of the abend. The bridge exit can issue only non-recoverable requests in this call.

The interface between the bridge exit (the BRXA) and CICS is described fully in “Bridge exit area (BRXA)” on page 84. This interface is defined by CICS and must be used by all bridge exits.

The messaging interface between the bridge exit and the remote resource or the end-user is not formally defined. You may define this interface to suit your own environment. However, an interface has been defined that is used by both MQSeries, and the CICS TS/TD supplied exits. This interface is described in “Message data format” on page 61. You can use this interface definition as a basis for your own implementation using other transport mechanisms.

Formatter

A formatter is a user-replaceable program that can (optionally) be used in association with a bridge exit. The formatter performs all the message building and analysis functions otherwise done in the bridge exit. Separation of this function simplifies the logic of the bridge exit and allows the formatting code to be used by many different bridge exits.

The bridge exit area

The bridge exit area (BRXA) is the communication area between the bridge exit and CICS. It is a CICS COMMAREA (subject to normal length constraints) containing a number of sub-areas that are used by the bridge exit to process each call, and retain information between calls.

It contains the following subareas:

Header
This area contains version information and pointers to some of the following areas.

Transaction area
This area is used by bridge exit initialization processing. It contains information about the user transaction that CICS runs, and the real 3270 that it expects to use.

Command area
This area provides details of the command request. For CICS API requests it provides a simplified description of the command and response fields.

User area
This area is used to store data between calls to the bridge exit. It acts as a user input area to store the messages needed to satisfy RECEIVE and RETRIEVE requests, and also as a user output area to store the messages from SEND requests so that they can all be sent together when the user transaction terminates.

ADS descriptor
This area contains the ADS descriptor for BMS SEND MAP and RECEIVE MAP requests.

If the user application issues a BMS command, the bridge exit is called and passed (in the user area) the BMS Application Data Structure (ADS). This is another name for the symbolic map that is generated by the BMS macros used to define the mapping of the 3270 screen. For BMS programs this gives the client application a simplified interface to the terminal data, without the need to understand 3270 data streams.

The ADS descriptor allows the exit to interpret the BMS Application Data Structure (ADS), without requiring that the copybook for the Application Data Structure (or DSECT) be included in the source for the exit program at compile time.

The ADS descriptor is generated as part of the mapset load module produced by the map definition macros, provided these are assembled using the CICS Transaction Server for OS/390 Release 2 or later macro library. Mapsets generated with previous CICS releases do not contain the ADS descriptor. If the mapset does not contain the ADS descriptor, a null pointer is set in the bridge exit area.

The ADS descriptor can be generated in short or long form, by specifying the DSECT option on the DFHMSD macro. The long form, ADSL, contains all fields aligned on 4-byte boundaries. This is required if the transport mechanism that you are using is MQSeries, or cross-platform.

Note: If you are unable to reassemble the mapset because you do not have the source, you can use the DFHBMSUP utility provided by

CICS Transaction Server for OS/390 Release 2 to recreate source statements from your mapset load module. See the *CICS Operations and Utilities Guide* for information about DFHBMSUP.

During initialization, the bridge exit can set an indicator in the bridge exit area to control whether or not the ADS descriptor should be passed to the exit on BMS SEND MAP and RECEIVE MAP calls. (If the ADS descriptor is required, the CICS interface code has to load the mapset and locate the descriptor, thus increasing the path-length.)

The format of the BRXA is defined in “Bridge exit area (BRXA)” on page 84.

The bridge facility

The bridge facility is a virtual terminal, replacing a real 3270, which is visible only to the user transaction and does not appear in response to CEMT I TERM or CEMT I TASK.

It has a dynamically created TERMID that can be used, for example, as the basis of a unique name for a TD or TS queue.

The bridge facility emulates a real terminal in the following EXEC CICS interfaces:

- ASSIGN
- Terminal control and BMS API
- EIB
- INQUIRE TASK
- INQUIRE TERMINAL

Note: EXEC CICS INQUIRE TERMINAL and INQUIRE TASK return information about a bridge facility only if issued from within the user transaction. The bridge facility is not visible outside the user transaction.

The bridge facility is discarded at the end of the user transaction when the bridge exit (during termination processing) sets the **keep time** to zero. The keep time defines how long the bridge facility should be retained after the transaction terminates. The bridge exit must specify a non-zero value if the bridge facility is to be kept for the next part of a pseudoconversation.

The bridge facility can have a TCTUA (Terminal Control Table User Area), which can be accessed by EXEC CICS ADDRESS TCTUA in the normal way. The TCTUA is initialized to nulls when the bridge facility is created.

A global user exit (GLUE) called XFAINTU is called when a bridge facility is created and discarded. XFAINTU is passed the address of the TCTUA, so you can use this exit to initialize the TCTUA.

The characteristics of a bridge facility are copied from a FACILITYLIKE definition, with the addition of preset security.

The FACILITYLIKE definition

FACILITYLIKE is the name of a real terminal definition that is used as a template for some of the properties of the bridge facility.

The name of the FACILITYLIKE definition to be used can be passed to CICS in one of three ways (the first non-blank value found is used):

- The bridge exit can return the FACILITYLIKE name in the BRXA_FACILITYLIKE parameter of the bridge exit initialization call.
- The parameter can be obtained from the PROFILE definition for the user transaction.
- The default is CBRF, a definition supplied by CICS to support the bridge.

Once the bridge facility has been defined, its FACILITYLIKE template cannot be changed. Therefore, if the bridge facility is reused in a pseudoconversation, CICS does not search for a new FACILITYLIKE value.

Note: If you are running in a CICS system started with the VTAM=NO system initialization parameter, the resource definition specified by FACILITYLIKE must be defined as REMOTE. A default definition of CBRF, defined as REMOTE, is provided in the group DFHTERM.

Bridge implementations provided

The following bridge programs are provided:

DFH0CBRE
A COBOL bridge exit program that uses CICS temporary storage or transient data queues to pass messages (in MQCIH format) to the end user application (another CICS application).

DFH0CBRF
A COBOL bridge exit formatter designed to work with DFH0CBRE. This builds and interprets BRMQ message vectors.

DFH0CBAC
A COBOL client application that uses CICS BTS CONTAINERs to run an end-user application as a CICS BTS ACTIVITY. See the *CICS Business Transaction Services* for further information about this application.

DFH0CBAI
A COBOL program that is used in DFH0CBAC to obtain message input. It uses a TS queue to obtain the input, so the program can be used by the BRCB transaction in the CA1E SupportPak.

DFH0CBAO
A COBOL program that is used in DFH0CBAC to send message output. It uses a TS queue to send the output, so the program can be used by the BRCB transaction in the CA1E SupportPak.

DFH0CBAE
A COBOL bridge exit program that uses CICS BTS CONTAINERs to pass messages (in MQCIH format) to the client application (DFH0CBAC).

DFHWBLT
An object code bridge exit that allows you to access a CICS transaction from the World Wide Web. This exit uses the CICS Web Interface support described in the *CICS Internet Guide*.

An MQ exit
An object code bridge exit that allows you to access a CICS transaction from MQSeries. This exit is provided as part of the MQSeries - CICS/ESA bridge feature of MQSeries for MVS/ESA Version 2.1

Copybooks

The following copybooks are provided:

DFH0CBRD
COBOL copybook used by DFH0CBRE, and DFH0CBAE.

DFH0CBRU
COBOL copybook used by DFH0CBRF

DFHBRSDx
Copybooks in all supported languages defining the interface between the bridge monitor and the bridge exit.

DFHBRMHx
Copybooks in all supported languages defining the message header included in all messages passed between the supplied bridge exit and the client application. Constants are also supplied for these copybooks.

DFHBRMQx
Copybooks in all supported languages defining the command vectors in the messages passed between the supplied bridge exit and the client application. Constants are also supplied for these copybooks.

See “Data formats for the supplied bridge exits” on page 59 for more information about the formats of message headers and vectors.

Resource definitions

The following resource definition groups are provided:

DFH$BR
resource definitions to support the DFH0CBRE bridge implementation.

DFH$BABR
resource definitions to support the DFH0CBAC CICS BTS bridge implementation.

The CA1E SupportPak

The CA1E SupportPak is a support package providing the CICS 3270 Bridge Passthrough tool. This allows you to run a CICS 3270 user transaction from a 3270 terminal, using the CICS 3270 Bridge facility rather than standard CICS terminal control function. You can then evaluate whether a CICS 3270 transaction is suitable to be driven using the 3270 bridge.

The Passthrough transactions also allow you to examine the 3270 data streams created by the bridge exit, and log them for further analysis. You can then use this information to write your own end-user application to drive the CICS 3270 transaction instead of a real 3270 terminal.

The CA1E SupportPak can be obtained from the Web, at the following URL:

`http://www.software.ibm.com/ts/cics/txppacs`

Running 3270 transactions in a bridge environment

A user transaction is started directly by a bridge monitor transaction using the START BREXIT TRANSID command.

The user transaction is initialized in a bridge environment; all 3270 terminal commands are intercepted and passed to the named bridge exit that emulates the 3270 terminal by packaging the commands into messages and passing them to the transport mechanism for delivery to a client application.

A formatter may be used to package the commands into messages, to simplify the role of the bridge exit as a message handler only.

The bridge monitor transaction is normally a long running task associated with a message queue. It looks at the contents of each message to search for requests for new work, and identifies the name of the user transaction to run. It then starts the requested transaction and checks that it has started successfully.

The client application, which may be executing anywhere accessible by the transport mechanism, extracts the 3270 data from the messages and constructs reply messages, containing 3270 data and commands, to pass to the bridge exit to satisfy the 3270 terminal commands.

Simple terminal transactions

A simple terminal transaction is one in which all user data required to run the transaction is available before the transaction has started. There is a single input screen and one or more output screens. It may issue recoverable requests.

The following example of a simple inquiry or update transaction shows how a bridge exit is called to process various requests, and is passed a structured COMMAREA containing the full description of the request.

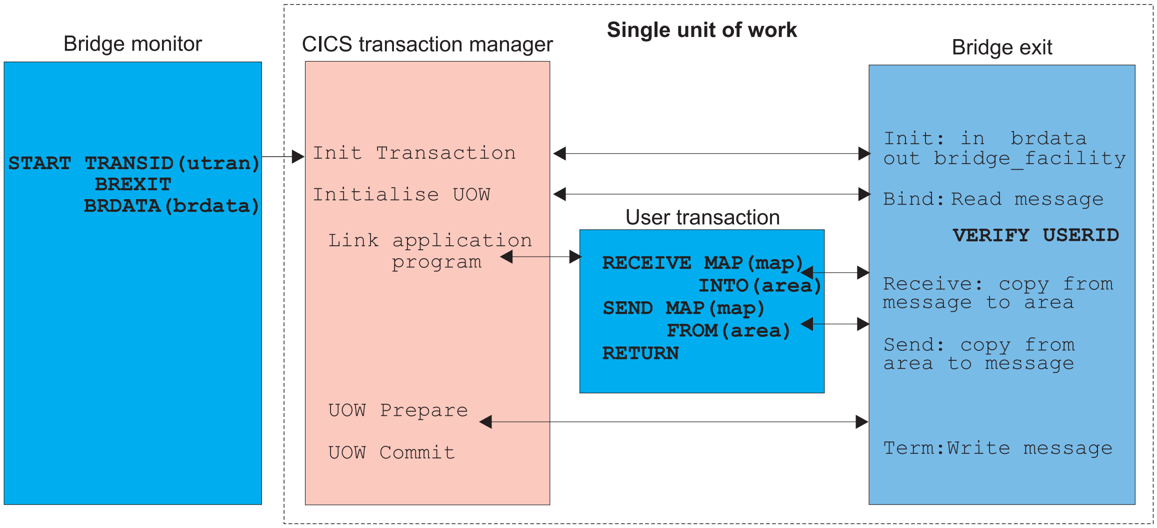


Figure 9. A simple terminal transaction

The following example shows a simple inquiry or update transaction when a formatter is used.

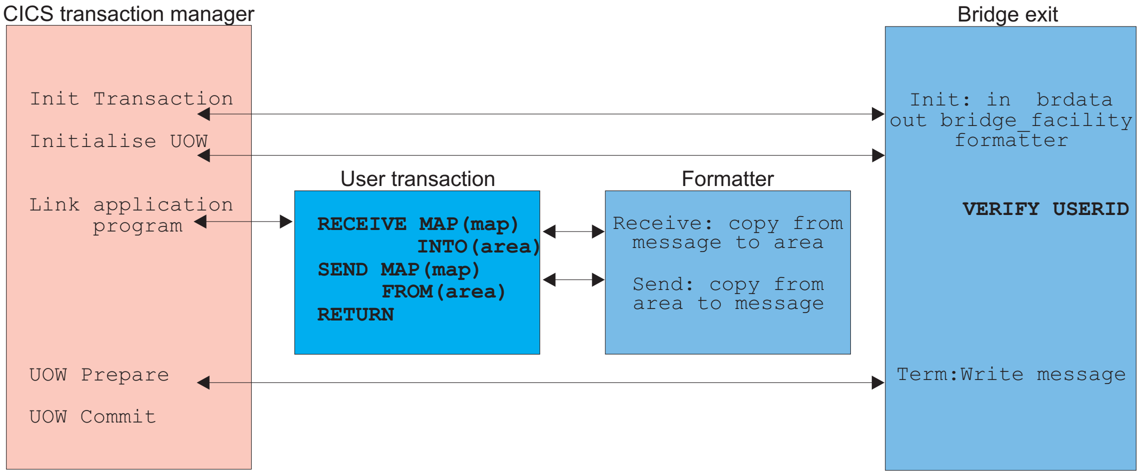


Figure 10. A simple terminal transaction using a formatter

Pseudoconversational transactions

A pseudoconversation normally involves a series of transactions, each initiated by the previous transaction, which may also pass some data. The name of the next transaction to be run can be defined by the user transaction in different ways:

1. EXEC CICS RETURN TRANSID
2. EXEC CICS RETURN TRANSID IMMEDIATE
3. EXEC CICS START TRANSID TERMID
4. EXEC CICS SET TERMINAL/NETNAME NEXTTRANSID
5. Terminal data

Note: Transactions initiated by START TERMID are not necessarily pseudoconversational. Here we are considering only those transactions initiated by a START to the principal facility (the bridge facility) where the STARTING and STARTED applications are associated in a pseudoconversation. In this case, START TERMID must specify the bridge facility.

Commands 1-4 all cause the bridge mechanism to pass the next transaction identifier and the START code in the bridge exit area (BRXA) on the termination and abend calls, with an indicator showing the source of the next TRANSID. This indicator can have 3 settings:

IMMEDIATE
The next TRANSID value came from a RETURN IMMEDIATE.

STARTED
The next TRANSID value cam from a START TERMID.

NORMAL
The next TRANSID value came from a RETURN TRANSID or SET command.

The BRXA_STARTCODE field is also set to the start code appropriate for the next transaction.

At transaction termination, the bridge exit is called and passed the BRXA containing the next TRANSID and START code, and the indicator. It can then issue an EXEC CICS START BREXIT command for the next TRANSID, or return the next transaction information to the client application.

You can design the bridge exit to provide any of the following options for the client application, when issuing a message for a subsequent transaction:

- Copy the next TRANSID in the output message to the TRANSID field in the new input message. In this case the specified next TRANSID (from whatever source) will be run.
- Put a different TRANSID in the TRANSID field of the new input message. This provides a mechanism for cancelling the existing flow; the equivalent of logging or powering the terminal off. The existing next transaction and all the start requests are discarded. An exception trace entry is written.

To preserve the pseudoconversational environment, the bridge exit requests that its bridge facility be kept by specifying a **keep time** value. A bridge facility token is returned to the client application. Subsequent transaction requests in the pseudoconversation must contain this token.

Note: The same bridge facility must be used by all transactions in the pseudoconversation.

Pseudoconversational and terminal information (the COMMAREA and TCTUA) are saved and associated with the bridge facility.

The TERMID is preserved for the lifetime of the bridge facility. This means that transactions that set up TS or TD queues using a TERMID as part of the name can be run in a bridge environment.

Note: If the next TRANSID is set (or defaulted) by the bridge exit to the same value as the next TRANSID saved in the bridge facility, CICS treats the transaction as part of a pseudoconversation. The user application can issue EXEC CICS INQUIRE TASK STARTCODE to find out if it is part of a pseudoconversation. The startcode ' TO' is returned for the first transaction and 'TP' for subsequent transactions in the pseudoconversation.

Conversational transactions

The bridge exit can handle conversational user transactions that involve multiple terminal input screens resulting from the transaction issuing multiple RECEIVEs. These transactions can by handled by the bridge mechanism in two ways:

- The client application provides the bridge exit with all of the input necessary to satisfy all of the RECEIVE requests. In effect, this turns a conversational transaction into a non-conversational transaction. This is possible only if all of the data necessary to satisfy all input requests is known before starting the transaction.
- It is more usual for the end-user to analyze information from a previous SEND before responding to a subsequent RECEIVE. To support this the bridge exit sends a message requesting more data to the client application.

Implementing a 3270 bridge environment

The bridge mechanism is very flexible, but most implementations fall into a few basic models.

This section tells you how to identify the model that fits the requirements of your system and applications. It then presents examples of each model, which you can use as checklists when preparing your own client programs, bridge exits and monitors.

It covers the following topics:

- “Determining the bridge environment model”
- “Using the long running monitor model to implement the 3270 bridge” on page 33
- “Using the two task model to implement the 3270 bridge” on page 35
- “Using a single message model to implement the 3270 bridge” on page 37
- “Using the direct model to implement the 3270 bridge” on page 39

Determining the bridge environment model

There are basically four models:

- Long running monitor
- Two-task model
- Single message monitor
- Direct (without a bridge monitor)

From the questions asked in the following sections, you can determine what kind of model is required to run the bridge in your implementation:

Is the client application a CICS transaction?

YES use the **Direct** model. The client program can issue the EXEC CICS START BREXIT command itself, and does not need a bridge monitor.

Can the bridge exit send and receive messages directly from the client application?

NO use the **Two-task** model.

Is a new transaction started when a message arrives in CICS?

YES use a **Single message** model.

Otherwise, a **Long-running monitor** model should be used.

Using the long running monitor model to implement the 3270 bridge

In this model, the client can be anywhere, but is usually outside CICS, possibly on a workstation platform. The client application sends a message to CICS, and waits for a response.

The bridge monitor:

- Identifies that there is a new message (It could be POSTed, or browse a queue)
- Browses the message to obtain the user TRANSID and FACILITYTOKEN, if present
- Issues an EXEC CICS START TRANSID() BREXIT() BRDATA()

This runs the user transaction. The bridge exit is called to read and write messages directly from/to the client application. The bridge monitor does not (usually) need any further involvement in this request, but remains waiting for new requests.

Note: The bridge monitor must not get a lock on the message as the message must be readable by the bridge exit. The messages can be recoverable.

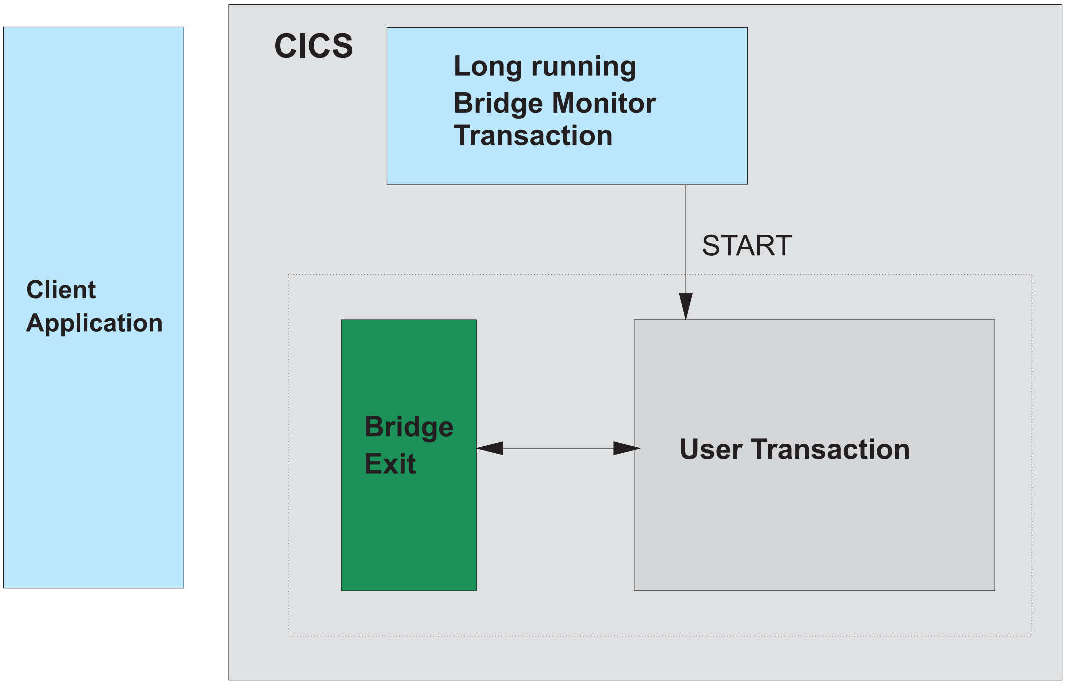


Figure 11. The long running monitor model

Client application design

1. Set initial FACILITYTOKEN to nulls
2. Create a message header containing:
   - TRANSID
   - FACILITYTOKEN
   - FACILITYLIKE
   - USERID (optional)
   - output message and/or queue identifier
3. Loop until end of transaction and NEXTTRANSID is blank:

   - Create message vectors containing information to satisfy each expected CICS API request (this is usually just a RECEIVE or RETRIEVE vector)
   - Create a message containing the message header plus zero or more vectors
   - Output message
   - Input message reply (either with a wait option, or a loop)
   - Copy the input message header to the next output message
   - Extract message vectors from the input message
   - Process these and get futher input from the user if necessary
   - If next transid is set then copy next transid to TRANSID
4. End of loop

Bridge monitor design

1. Get startup information (for example, queue identifier)
2. Initialize queues
3. Loop until shutdown request
   - Wait for new message
   - Browse new message header (without locking it)
   - Extract following from the message header
     - TRANSID
     - FACILITYTOKEN
     - FACILITYLIKE (optional)
     - USERID (optional)
     - output message and/or queue identifier
   - Create brdata containing (see DFHBRSD for example)
     - input message and/or queue identifier
     - output message and/or queue identifier
     - FACILITYTOKEN or nulls if new request
     - FACILITYLIKE if new request (default to blanks)
   - EXEC CICS START TRANSID(transid) BREXIT(bridge exit) BRDATA(brdata)
   - Check if task shutdown
4. End of loop
5. Shutdown process

Bridge exit design

The supplieds DFH0CBRE and DFH0CBRF can be used with the following changes (if necessary):

- Change the transport mechanism specific calls
- If a message header other than MQCIH is used change this in DFH0CBRF
- If message vectors other than the BRMQ vectors are used, change these in DFH0CBRF

Using the two task model to implement the 3270 bridge

In this model, the client can be anywhere, but is usually outside CICS, possibly on a workstation platform The bridge monitor is started by a message arriving in CICS. The bridge monitor then:

- Browses the message to obtain the user TRANSID and FACILITYTOKEN, if present
- Issues an EXEC CICS START TRANSID() BREXIT() BRDATA()

The bridge exit cannot read or write the messages directly using the transport mechanism, so the bridge monitor and bridge exit communicate with each other using ECBs. When the bridge exit wants to write a message, it stores the message in memory, or on a non recoverable TS queue, and posts the bridge monitor. The bridge monitor then gets the message and sends it to the client. This model is the most complex, and the messages are not recoverable.

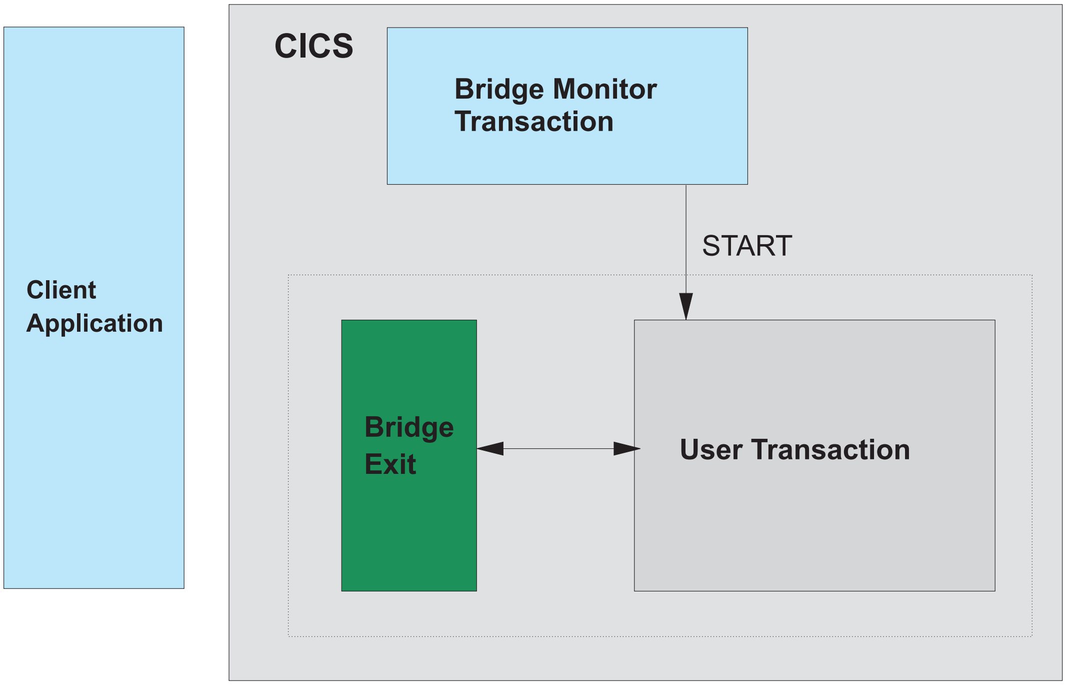


Figure 12. The two-task model

Bridge monitor design

1. Get startup instance information (for example, message and queue identifier)
2. Get shared storage for message buffers Initialise queues
3. Read new message header (if possible without locking it)
4. Extract following from the message header
   - TRANSID
   - FACILITYTOKEN
   - FACILITYLIKE (optional)
   - USERID (optional)
   - output message and/or queue identifier
5. Create brdata containing (see DFHBRSD for example)
   - input message buffer address

- output message buffer address
- FACILITYTOKEN or nulls if new request
- FACILITYLIKE if new request (default to blanks)
- ecb address
- message buffer address

6. EXEC CICS START TRANSID(transid) BREXIT(bridge exit) BRDATA(brdata)
7. Loop until message indicates end of transaction
   - Wait on ecb
   - Get message from bridge exit
   - Write message
   - If message is a request for more data:
     - Read next message (wait for response from client)
     - Send message to bridge exit
     - Post ecb

Bridge exit design

The supplieds DFH0CBRE and DFH0CBRF can be used with the following changes (if necessary):

- Change the transport mechanism specific calls to use the post/wait mechanism
- If a message header other than MQCIH is used change this in DFH0CBRF
- If message vectors other than the BRMQ vectors are used, change these in DFH0CBRF

Using a single message model to implement the 3270 bridge

In this model, the client can be anywhere, but is usually outside CICS, possibly on a workstation platform The bridge monitor is started by a message arriving in CICS. The bridge monitor may read the message recoverably, but in this case should issue a syncpoint before issuing the START request. The bridge monitor then:

- Browses the message to obtain the user TRANSID and FACILITYTOKEN, if present
- Issues an EXEC CICS START TRANSID() BREXIT() BRDATA()

After the monitor has issued the START it has no further involvement, so can terminate.

When the bridge exit is called to read and write messages, it communicates directly with the client application. The messages can be recoverable.

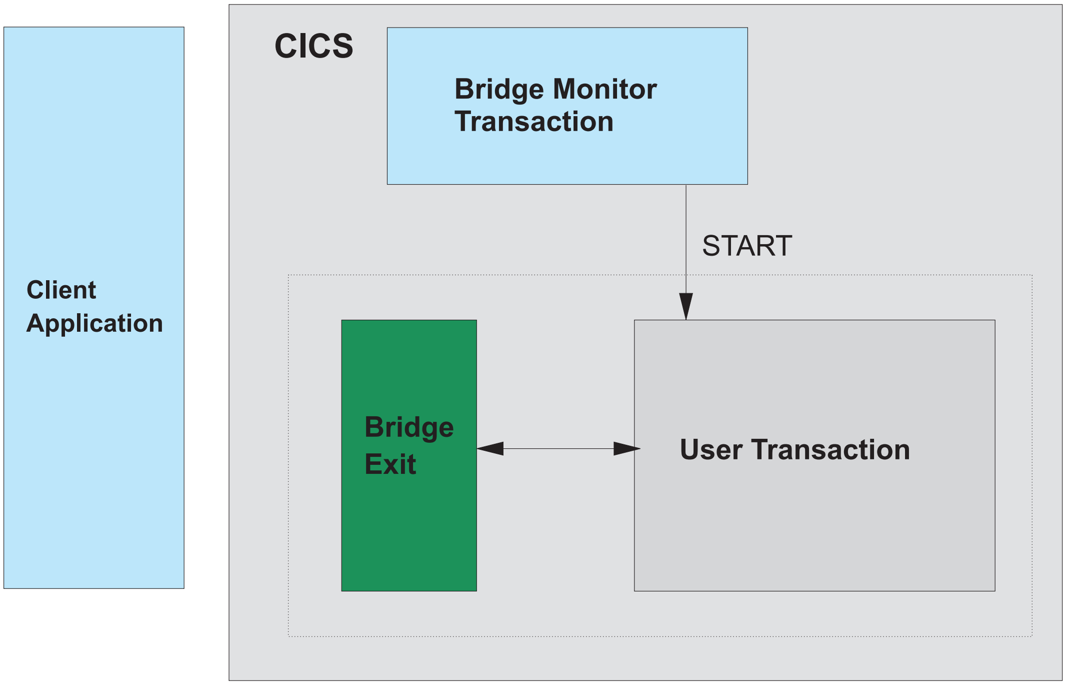


Figure 13. The single message model

Bridge monitor design

1. Get start-up instance information (for example, message and queue identifier)
2. Initialize queues
3. Read new message header (if possible without locking it)
4. Extract the following from the message header:
   - TRANSID
   - FACILITYTOKEN
   - FACILITYLIKE (optional)
   - USERID (optional)
   - output message and/or queue identifier
5. Create brdata containing (see DFHBRSD for example):
   - input message and/or queue identifier

- output message and/or queue identifier
- FACILITYTOKEN or nulls if new request
- FACILITYLIKE if new request (default to blanks)

6. EXEC CICS START TRANSID(transid) BREXIT(bridge exit) BRDATA(brdata)

Using the direct model to implement the 3270 bridge

In this model, a CICS application directly starts a user transaction running in the bridge environment. This model has been known in the past as the ’user bridge’.

The client application, which is a CICS transaction:

- Writes a message on a queue (probably a TS queue)
- Creates brdata (probably consisting of the TS queue name and a null FACILITYTOKEN)
- Issues an EXEC CICS START TRANSID() BREXIT() BRDATA()

The client application then waits for data on the output queue. When the bridge exit is called to read and write messages, it communicates directly with the client application.

If Temporary Storage (TS) queues are used for the transport mechanism, they must not be recoverable as they are used by both the client and the bridge exit. If the same exit is used, communication between the client and bridge exit is synchronized by repeated EXEC CICS DELAY commands to wait for messages on the TS queue. If this is not efficient enough, this can be changed to an ECB mechanism.

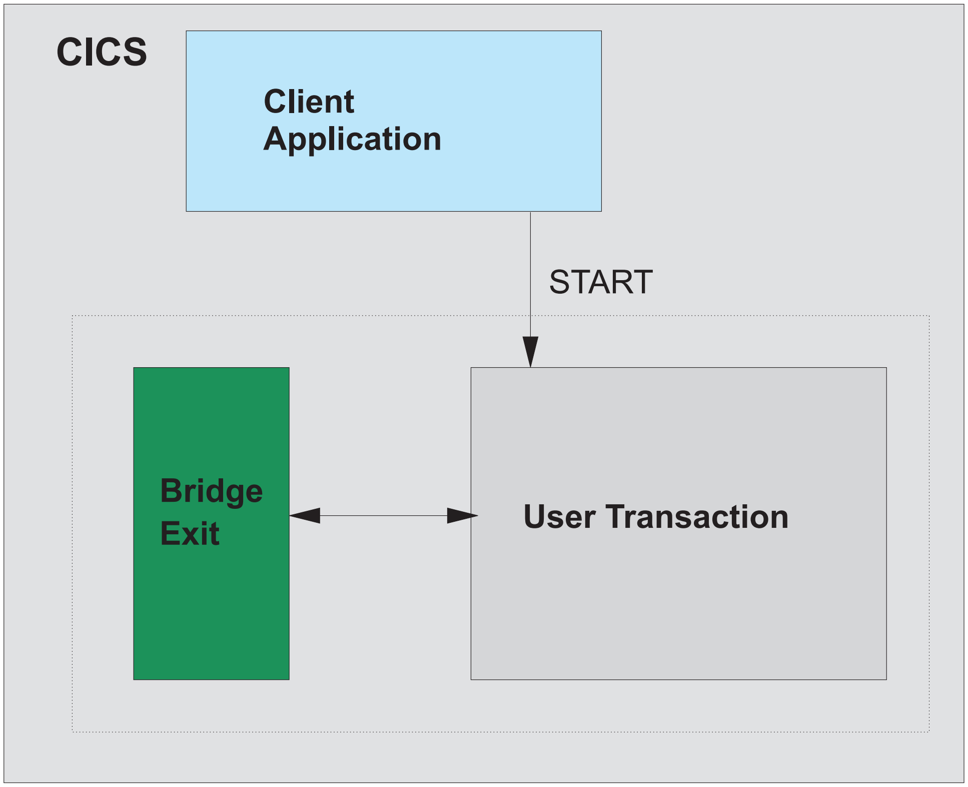


Figure 14. The Direct model

Client Application (CICS transaction) design

1. Create new message
   - First transid in pseudo conversation
   - Null FACILITYTOKEN

- FACILITYLIKE or blank

2. Loop until end of pseudo conversation:
   - Write message to TS queue
   - Create brdata (using DFHBRSD) containing:
     - Input and output queue names
     - FACILITYLIKE and FACILITYTOKEN from message
   - EXEC CICS START TRANSID(from message) BREXIT(bridge exit) BRDATA(dfhbrsd)
   - Loop until end of transaction:
     - Loop until message obtained:
       - Read message from TS queue
       - Wait a second
     - Process message (such as send response to client)
     - If request for more data
       - Create next message
       - Write message to TS queue
   - If there is a next TRANSID
     - Copy next TRANSID to TRANSID in message

Bridge exit design

DFH0CBRE and DFH0CBRF can be used unchanged.

Chapter 4. The Bridge environment

The bridge environment is established when CICS receives a START BREXIT TRANSID call. The transaction specified by TRANSID is associated with the bridge exit specified by BREXIT (or specified on the TRANSACTION resource definition), and a virtual 3270 terminal, the bridge facility, is created.

The transaction executes in the usual way, but all terminal commands are intercepted and passed to the bridge exit.

The user transaction is unchanged, but because of the way it now executes in the bridge environment, there are some restrictions on what it can do, and some limitations on how it can use the bridge facility, because it is not a real terminal.

This chapter describes these special characteristics of the user transaction. It covers the following topics:

- “User transaction programming considerations”
- “Defining the user transaction” on page 46
- “Inquiring about the bridge environment” on page 47
- “The bridge facility” on page 52

User transaction programming considerations

The user transaction runs unchanged, with the following limitations.

Abend information
The bridge facility name is not used as the TERMID in any diagnostic information produced as the result of an abend, except in a transaction dump.

ASSIGN
If the user transaction issues ASSIGN NETNAME, the value returned is the TERMID. The name is not visible outside the user transaction, and may contain characters that are not allowed in a VTAM netname definition.

You can only use ASSIGN to request information about BMS attributes such as MAPCOLUMN, MAPHEIGHT, MAPLINE, and MAPWIDTH, if an ADS descriptor is present in the mapset, and BRXA_LOAD_ADS_DESCRIPTOR is set to Y in the bridge exit area. See“Bridge exit area (BRXA)” on page 84 for details of the fields in the bridge exit area.

BMS requests
The 3270 bridge supports the following BMS commands. If other BMS functions that require a principal facility are used, they cause the user transaction to abend ABR3.

RECEIVE commands:

Command	Calls bridge exit
RECEIVE MAP TERMINAL	YES
RECEIVE MAP FROM	NO
RECEIVE MAP MAPPINGDEV	NO

© Copyright IBM Corp. 1994, 1998

Note: TERMINAL is implied if neither TERMINAL nor FROM is specified.

SEND commands:

Command	ACCUM supported	Calls bridge exit
SEND MAP TERMINAL	NO	YES
SEND TEXT TERMINAL	NO	YES
SEND TEXT NOEDIT TERMINAL	NO	YES
SEND TEXT MAPPED TERMINAL	NO	YES
SEND CONTROL TERMINAL	NO	YES
SEND MAP SET	YES	NO
SEND TEXT SET	YES	NO
SEND TEXT NOEDIT SET	YES	NO
SEND TEXT MAPPED SET	YES	NO
SEND CONTROL SET	YES	NO
SEND MAP MAPPINGDEV	NO	NO

Note: TERMINAL is implied if none of TERMINAL, SET, or PAGING is specified.

Routing:

Routing to real terminals from a transaction running on a bridge facility is supported, but it is not possible to route to a bridge facility, nor to specify a bridge facility as ERRTERM on ROUTE. If ERRTERM without a name is specified on a ROUTE request issued in a bridge environment, the INVERRTERM condition is raised.

PAGING is supported only under routing.

Partitions:

Partition related commands and options are supported, but are treated in the same way as they would be for a real terminal that does not support partitions.

SEND PARTNSET
Supported, but the bridge exit is not invoked.

RECEIVE PARTN
Supported; the bridge exit is invoked with bridge exit area command fields set up for a terminal control RECEIVE.

INPARTN
Accepted but ignored; not passed to the bridge exit.

OUTPARTN
Accepted but ignored; not passed to the bridge exit.

ACTPARTN
Accepted but ignored; not passed to the bridge exit.

CICS-supplied transactions
CEDF, CEDX, CSFE, and CSGM cannot run as user transactions.

DB2® authorization check

The RCT allows AUTHTYPE=TERMID or OPID which means that security checking is done against the corresponding name. This fails in a bridge environment, and AUTHTYPE=USERID must be used instead. This is the preferred method in all environments.

External security customization

TERMID/OPID/TCTUA information is not passed in the DFHXSID parameter list.

Global User Exits

The following global user exits (GLUEs) are **not** driven because the bridge facility is not a real terminal.

XBMIN
to intercept a RECEIVE MAP request.

XBMOUT
to intercept a SEND MAP request.

XTCATT
before a task attach (TCAM).

XTCIN after an input event (TCAM).

XTCOUT
before an output event (TCAM).

XZCATT
before a task attach (VTAM).

XZCIN after an input event (VTAM).

XZCOUT
before an output event (VTAM).

XZCOUT1
before a message is broken into RUs (VTAM).

The XALTENF and XICTENF exits can be driven if a request is made for a bridge facility. The ‘terminal-not-found’ condition is raised because the bridge facility is not a real terminal.

The standard user exit parameter list field UEPTERM that points to the TERMID are not set for exits invoked under a bridge task.

ISSUE PRINT

ISSUE PRINT is not supported and results in a no-op. A NORMAL condition is returned.

Monitoring

A 3270 bridge transaction identifier has been added to monitoring records.

Remote DLI requests

No security check of the PSB against the terminal is done for function-shipped DLI requests.

Security Processing

When a bridge facility is created, it is signed on as a preset USERID terminal. The USERID is the value returned on the bridge exit initialization call. If no value is returned, the USERID with which the transaction was started is used. As with other preset terminals, the SIGNON and SIGNOFF commands are not permitted, and INVREQ is raised.

The bridge facility is signed off when it is discarded. It remains signed on if the user transaction ends and the bridge exit specifies a keep time value.

When the bridge exit initialization call returns a bridge facility token, a check is made that the USERID is the same as that specified when the bridge facility was created. No other security check is made. If a greater level of security is required, such as validation of every message, this can be done in the bridge exit by issuing a VERIFY PASSWORD command.

START

The user transaction can issue EXEC CICS START requests for its own bridge facility. This allows existing menu-driven and pseudo-conversational applications that use this interface to work in a bridge environment. See "Pseudoconversational transactions" on page 30 for a description of START TERMID where TERMID specifies the bridge facility.

The time delay options, (INTERVAL, TIME, AFTER, AT, HOURS, MINUTES, SECONDS) are not normally used in the bridge environment. If any of these options are specified, they are saved with the other START data and passed in the BRXA to the TERM call of the bridge exit. It is then the responsibility of the bridge exit to take account of them. They could also be passed back to the end user application and used there in some way.

The INTERVAL and AFTER values are used to put the STARTs for a particular bridge facility in time order, but the exact delays requested are not implemented. TIME and AT specifications are ignored completely.

Other options on the START command are not fully supported because they are not required in the bridge environment:

TERMID

You can only specify the name of the bridge facility for this transaction. Any other name will result in a TERMIDERR.

USERID

USERID and TERMID are mutually exclusive. The CICS translator rejects START requests with both USERID and TERMID specified.

TRANSID

If the TRANSID cannot be defined as REMOTE. The TERMID will not be found if the request is shipped to a remote system.

SYSID Routing of START requests is not possible in a bridge environment. This option is not supported, unless the value of the SYSID is the local SYSID. If you specify any other value, the request will be shipped and the TERMID will not be found on the remote system.

NOCHECK

This option only applies to shipped start requests and is ignored.

PROTECT

If you specify the PROTECT option on a START request for a bridge facility, and the starting task abends before taking a syncpoint, the START request is discarded. PROTECT normally delays the starting of the new task until a SYNCPOINT has occurred. This happens automatically for a task issuing a START for its own facility because the START cannot take effect until the starting task has terminated and freed up its bridge facility.

ATTACH

This option applies only to non-terminal starts. The CICS translator rejects START ATTACH requests when TERMID is specified.

BREXIT

The BREXIT option applies only to non-terminal starts. The CICS translator rejects START BREXIT requests when TERMID is specified.

STARTed transactions

Some menu applications use START to initiate subsequent transactions.

Started transactions are identified by specifying the appropriate value in BRXA_STARTCODE the bridge exit initialization call. This value is used to return the correct response to ASSIGN STARTCODE and INQUIRE TASK STARTCODE commands issued by the user transaction.

User transactions that are initiated by START may issue one or more RETRIEVEs to obtain data passed on the START. If there is no data, CICS does not immediately return ENDDATA, but calls the bridge exit, which can then provide data to satisfy the request, or return the ENDDATA condition.

Storage violation counts

No storage violation counts will be kept in a bridge facility.

SYNCPOINT

If the user transaction issues an explicit SYNCPOINT, or an implicit SYNCPOINT occurs, as in CREATE, DISCARD CONNECTION, DISCARD TDQUEUE, DISCARD TERMINAL, or in a DLI TERM psb, the bridge exit is called (if the BRXA_CALL_FOR_SYNC parameter is set to BRXA_YES), and reissues the SYNCPOINT, or ignores it. The bridge exit is permitted to do recoverable work before and/or after the SYNCPOINT, so it can do recoverable work in either unit of work. If the bridge exit does not reissue the SYNCPOINT, the logic of the transaction could be affected.

There is no specific call to the bridge exit for the implicit syncpoint at the end of the user transaction; this is handled by the termination and/or abend calls. The exit does not need to issue a SYNCPOINT request in the termination call, and must not issue a SYNCPOINT request in the abend call.

TCTUA

The TCTUA is available after the INIT call to the bridge exit.

Transaction restart

RESTART(NO) is forced for user transactions because CICS has no way of restoring the initial input message.

Transaction Routing

Transaction Routing is not supported.

TWA

The TWA is available after the INIT call to the bridge exit.

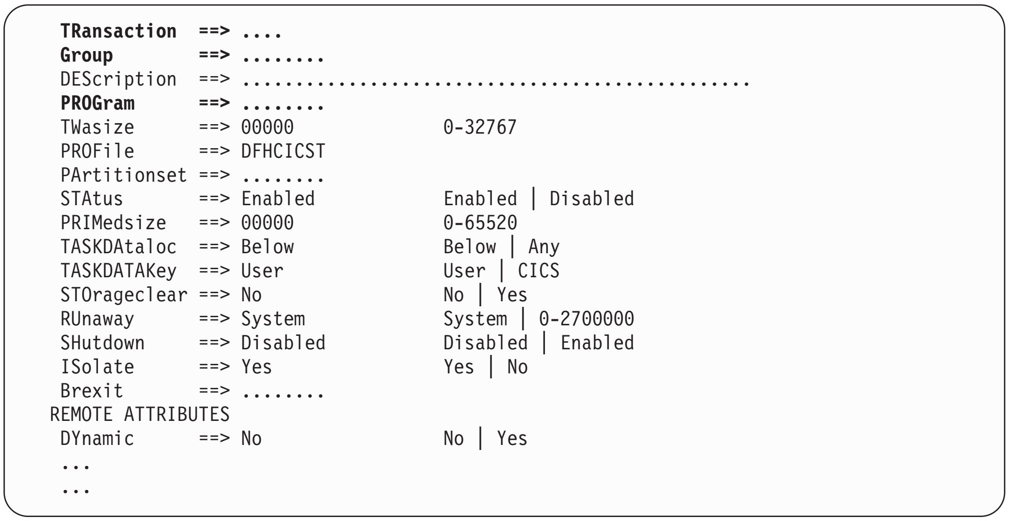
Defining the user transaction

Keywords are provided in the following resource definitions to define the default bridge exit and facility.

- TRANSACTION
- PROFILE

TRANSACTION resource definition

A user transaction definition can have an additional parameter, BREXIT, to define a default bridge exit. The named bridge exit is used when the transaction is specified in a START TRANSID BREXIT command, where the BREXIT name is blank. When the transaction is executed in the usual way, BREXIT is ignored.



```
 TRansaction  ==> ....
 Group        ==> ........
 DEScription  ==> ................................................
 PROGram      ==> ........
 TWasize      ==> 00000              0-32767
 PROFile      ==> DFHCICST
 PArtitionset ==> ........
 STAtus       ==> Enabled            Enabled | Disabled
 PRIMedsize   ==> 00000              0-65520
 TASKDAtaloc  ==> Below              Below | Any
 TASKDATAKey  ==> User               User | CICS
 STOrageclear ==> No                 No | Yes
 RUnaway      ==> System             System | 0-2700000
 SHutdown     ==> Disabled           Disabled | Enabled
 ISolate      ==> Yes                Yes | No
 Brexit       ==> ........
REMOTE ATTRIBUTES
 DYnamic      ==> No                 No | Yes
 ...
 ...
```

Figure 15. The DEFINE panel for the TRANSACTION resource definition

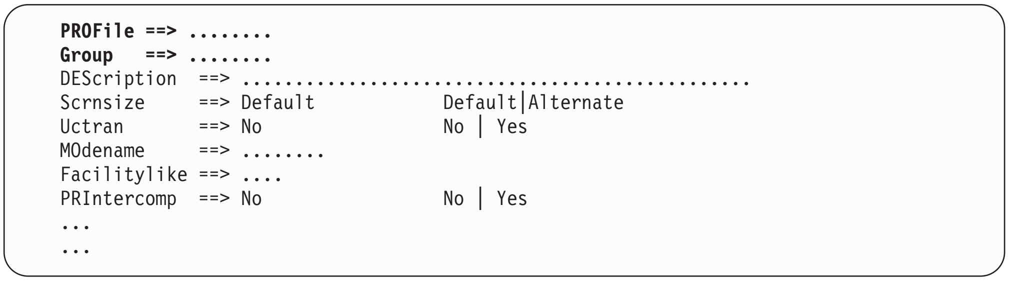
BREXIT
This is an optional parameter that defines the name of the default bridge exit to be associated with this transaction, if it is started in the 3270 bridge environment with a START BREXIT command, and BREXIT specifies no name.

The name may be up to 8 characters in length.

If BREXIT is defined, REMOTESYSTEM, REMOTENAME, DYNAMIC(YES), and RESTART(YES) should not be specified, and are ignored.

PROFILE resource definition

The PROFILE provides terminal-related information for a transaction, including the FACILITYLIKE parameter. The PROFILE of a user transaction can specify FACILITYLIKE to define the default terminal definition values to be used for the bridge facility.



```
PROFile ==> ........
Group   ==> ........
DEScription  ==> ................................................
Scrnsize     ==> Default            Default|Alternate
Uctran       ==> No                 No | Yes
MOdename     ==> ........
Facilitylike ==> ....
PRIntercomp  ==> No                 No | Yes
...
...
```

Figure 16. The DEFINE panel for the PROFILE resource definition

FACILITYLIKE

This is an optional parameter that specifies the name of an installed terminal resource definition to be used as a template for the bridge facility. It can be overridden by specifying FACILITYLIKE in the bridge exit.

There is no default value for this parameter, but if it is not defined here or in the bridge exit area, CICS uses CBRF.

If you are running in a CICS system started with the VTAM=NO System initialization (SIT) parameter, the resource definition specified by FACILITYLIKE must be defined as a remote terminal.

Inquiring about the bridge environment

You can use the following commands and interfaces to determine whether a transaction or task is executing in a bridge environment, and if so, to obtain information about the bridge monitor transaction that issued a START TRANSID BREXIT command to start the user transaction and its associated exit:

- ASSIGN
- INQUIRE TASK
- INQUIRE TRANSACTION
- CEMT
- The exit programming interface (XPI)

ASSIGN command

Function

Request values from outside the application program local environment.

Syntax

ASSIGN

```
►►──ASSIGN──┬─────────────────────────────────┬──────────────────────►◄
            └─BRIDGE(4-character data-area)───┘
```

Figure 17. ASSIGN (extract)

Options

BRIDGE

This parameter returns the transaction name (TRANSID) of the bridge monitor transaction that initiated the user transaction issuing this request.

A value of blanks is returned if :

- The user transaction was not started by a bridge monitor transaction.
- This command was issued by a program started by a distributed program link (DPL) request.

Conditions

Unchanged.

INQUIRE TASK command

Function

The INQUIRE TASK command returns information about a given task.

Syntax

INQUIRE TASK

```
                               ┌──────────────────────────┐
                               ▼                          │
►►──INQUIRE TASK(data-value)──────┬──────────────────────┬┴──────────►◄
                                  ├─BRIDGE(data-area)────┤
                                  └─IDENTIFIER(data-area)┘
```

Figure 18. INQUIRE TASK (extract)

Options

BRIDGE*(data-area)*

returns the 4-character name of the bridge monitor transaction that issued a START BREXIT TRANSID command to start this task. If this task is not currently running in the 3270 bridge environment, blanks are returned.

IDENTIFIER*(data-area)*

returns a 48-character field containing user data provided by the bridge exit, if the task was initiated in the bridge environment, or blanks, otherwise. This field is intended to assist in online problem resolution.

For example, it could contain the MQ correlator for the MQ bridge, or a Web token.

Conditions

Unchanged.

INQUIRE TRANSACTION command

Function

The INQUIRE TRANSACTION command returns information about a named transaction.

Syntax

INQUIRE TRANSACTION

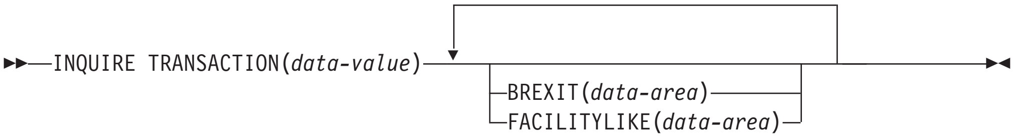


Figure 19. INQUIRE TRANSACTION (extract)

Options

BREXIT*(data-area)*
returns the 8-character name of the bridge exit defined by the BREXIT parameter of the named transaction resource definition.

If BREXIT is not defined, blanks are returned.

FACILITYLIKE*(data-area)*
returns the 4-character name of the terminal defined by the FACILITYLIKE parameter of the PROFILE associated with the named transaction resource definition.

If FACILITYLIKE is not defined, blanks are returned.

Conditions

Unchanged

CEMT INQUIRE TASK

Function

CEMT INQUIRE TASK returns information about a given task.

Syntax

CEMT INQUIRE TASK

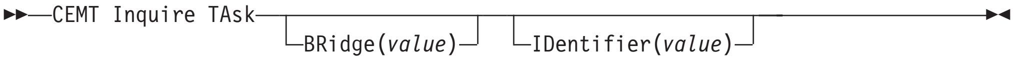


Figure 20. CEMT INQUIRE TASK (extract)

Options

BRidge*(value)*
returns the 4-character name of the bridge monitor transaction that issued a START BREXIT TRANSID command to start this task. Otherwise, blanks are returned.

IDentifier*(value)*
returns a 48-character field containing user data provided by the bridge exit, if the task was initiated in the bridge environment, or blanks, otherwise. This field is intended to assist in online problem resolution.

For example, it could contain the MQ correlator for the MQ bridge, or a Web token.

This field can contain hexadecimal values. The *CICS Supplied Transactions* manual tells you how to display these fields and provides more information about CEMT.

CEMT INQUIRE TRANSACTION

Function

CEMT INQUIRE TRANSACTION returns information about a named transaction.

Syntax

CEMT INQUIRE TRANSACTION

```
►►──CEMT Inquire TRAnsaction──┬────────────────┬──┬──────────────────────┬──────────►◄
                              └─BRexit(value)──┘  └─FAcilitylike(value)──┘
```

Figure 21. CEMT INQUIRE TRANSACTION (extract)

Options

BRexit*(value)*
returns the 8-character name of the bridge exit defined by the BREXIT parameter of the named transaction resource definition.

If BREXIT is not defined, blanks are returned.

FAcilitylike*(value)*
returns the 4-character name of the terminal defined by the FACILITYLIKE parameter of the PROFILE associated with the named transaction resource definition.

If FACILITYLIKE is not defined, blanks are returned.

XPI commands

INQUIRE_TRANDEF

The parameter BREXIT is provided on the INQUIRE_TRANDEF function, returning the following value:

BREXIT(name8)
returns the name of the bridge exit program. If there is no bridge exit, blanks are returned.

name8
The name of an 8-byte location to receive the name of the bridge exit program.

INQUIRE_CONTEXT

A new function, INQUIRE_CONTEXT has been created, returning the following values:

BRIDGE_EXIT_PROGRAM(name8)
returns the name of the bridge exit program used by this task. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name8
The name of an 8-byte location to receive the name of the bridge exit program.

BRIDGE_FACILITY_TOKEN(name4)
returns a token that contains the address of the bridge facility used by this task. This has the same format as a TCTTE and can be mapped using the DSECT DFHTCTTE. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4
The name of a 4-byte location to receive the token.

BRIDGE_TRANSACTION_ID(name4)
returns the name of the bridge monitor transaction used to start this user transaction. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4
The name of a 4-byte location to receive the name of the bridge transaction.

BRXA_TOKEN(name4)
returns a token that contains the address of the bridge exit area (BRXA) used by this task. The format of BRXA is defined by the DFHBRARx copy book. If CONTEXT returns NORMAL, the contents of this field are meaningless.

name4
The name of a 4-byte location to receive the token.

CONTEXT(byte1)
returns, in a 1-byte location (*byte1*), the type of environment in which the transaction is running.

NORMAL
A transaction that is not running in a bridge environment.

BRIDGE
A user transaction that was started using a bridge.

BREXIT
A bridge exit program.

See the *CICS Customization Guide* for more information about the XPI.

The bridge facility

The user transaction can retrieve information about the bridge facility using INQUIRE and ASSIGN.

A user transaction can issue INQUIRE TERMINAL or INQUIRE NETNAME for its bridge facility, or can issue INQUIRE TASK for itself. The TERMID can be obtained from EIBTRMID or from ASSIGN FACILITY, and the NETNAME can be obtained from ASSIGN NETNAME. Any other task issuing these commands for the bridge transaction facility receives TERMIDERR.

Bridge facilities do not appear in response to INQUIRE TERMINAL browses.

All keywords of ASSIGN and INQUIRE are supported and return the values that have been set for the bridge facility from the FACILITYLIKE terminal definition, or that have been set during the execution of the transaction.

Some keywords return values fixed by CICS for the bridge environment. These are:

Table 2. INQUIRE TERMINAL values

Keyword	Returned value
ACQSTATUS	ACQUIRED
ACCESSMETHOD	VTAM
CORRELID	blanks
EXITTRACING	NOTAPPLIC
LINKSYSTEM	blanks
MODENAME	blanks
REMOTENAME	blanks
REMOTESYSTEM	blanks
REMOTESYSNET	blanks
SERVSTATUS	INSERVICE
TCAMCONTROL	X’FF’
TERMSTATUS	ACQUIRED
TTISTATUS	YES
ZCPTRACING	NOZCPTRACE

Table 3. INQUIRE TASK values

Keyword	Returned value
FACILITY	the bridge facility
FACILITYTYPE	TERM or TASK
STARTCODE	S,SD,TO,TP

QUERY

The keywords listed below represent terminal attributes that can be set by the 3270 Query function at logon time for a real device:

ALTSCRNHT
BACKTRANSST
ALTSCRNWD
COLORST
APLKYBDST
EXTENDEDDSST
APLTEXTST
GCHARS

GCODES HILIGHTST MSRCONTROLST OUTLINEST
PARTITIONSST PROGSYMBOLST SOSIST VALIDATIONST

If the real FACILITYLIKE terminal is logged on when the bridge facility is created, the QUERY will have been performed and the values returned will apply to the bridge facility.

If the real FACILITYLIKE terminal is not logged on at the time that the bridge facility is created, the QUERY will not have been performed and the bridge facility will be created using values from the FACILITYLIKE resource definition.

SET TERMINAL/NETNAME

The following table shows the effect of each of the SET TERMINAL/NETNAME keywords when issued by a user transaction for its bridge facility. Unless otherwise specified, the response is DFHRESP(NORMAL).

KEYWORD	EFFECT
ACQSTATUS	Ignored.
ALTPRINTER	Value is SET, and is returned on INQUIRE, but is never used by CICS.
ALTPRTCOPYST	Value is SET, and is returned on INQUIRE, but is never used by CICS.
ATISTATUS	Works as for normal 3270.
CANCEL	Ignored
CREATESESS	Ignored.
DISCREQST	Value is SET, and is returned on INQUIRE, but is never used by CICS.
EXITTRACING	Ignored.
FORCE	Ignored.
MAPNAME	Works as for normal 3270.
MAPSETNAME	Works as for normal 3270.
NEXTTRANSID	Works as for normal 3270.
OBFORMATST	Works as for normal 3270.
PAGESTATUS	Ignored.
PRINTER	Value is SET, and is returned on INQUIRE, but is never used by CICS.
PRTCOPYST	Value is SET, and is returned on INQUIRE, but is never used by CICS.
PURGE	Ignored.
PURGETYPE	Ignored.
RELREQST	Value is SET, and is returned on INQUIRE, but is never used by CICS.
SERVSTATUS	Works as for normal 3270.
TCAMCONTROL	Returns INVREQ, as for normal 3270.
TERMPRIORITY	Value is SET, and is returned on INQUIRE, but is never used by CICS.
TERMSTATUS	Ignored.

KEYWORD	EFFECT
TRACING	Value is SET, and is returned on INQUIRE, but is never used by CICS.
TTISTATUS	Ignored.
UCTRANST	Works as for normal 3270.
ZCPTRACING	Ignored.

Bridge facility global user exit

When enabled, XFAINTU (FAcility INitialization and Tidy Up) is called when a bridge facility is created or deleted:

- Just after a new bridge facility has been built.
- Just before a bridge facility is deleted. This may be at the end of a task when zero keep time is specified, or when a keep time expires before the facility is reused.

XFAINTU is needed to carry out any auditing or initialization that is normally done at LOGON/LOGOFF or AUTOINSTALL/DELETE. This could be TCTUA setup or data collection that the 3270 user transaction relies upon.

| The initialization call to XFAINTU is made outside the unit-of-work environment. You should not invoke any services from the bridge exit that would require a unit-of-work environment. (Such as Recovery services).

See the *CICS Customization Guide* for more information about global user exits.

XFAINTU

When invoked

Just after a bridge facility is created and just before it is freed.

Exit-specific parameters

UEPFAREQ

Address of a 1-byte field that indicates why the exit has been called. Possible values are:

UEPFAIN

Initialization.

UEPFATU

Tidy-up.

UEPFATUT

Address of a 1-byte field that indicates the type of tidy-up required. Possible values are:

UEPFANTU

Normal tidy-up.

UEPFAETU

Expired tidy-up.

UEPFANAM

Address of the bridge facility name.

UEPFATYP

Address of a 1-byte field that indicates the facility type. The value is always:

UEPFABR
3270 bridge facility.

UEPFAUAA
Address of the bridge facility user area (TCTUA).

UEPFAUAL
Address of a one-byte field containing the length of the bridge facility user area.

Return codes

UERCNORM
Continue processing.

XPI calls
| All can be used, except those that use Recovery Manager services.

API calls
| All can be used except those that invoke task-related user exits, or use
| Recovery Manager services.

Chapter 5. Supplied 3270 bridge exits

This section tells you aboutsome of the IBM supplied bridge solutions listed in “Bridge implementations provided” on page 27. It covers the following topics:

- “The TS/TD supplied bridge exit”
- “The Web bridge exit” on page 58
- “Data formats for the supplied bridge exits” on page 59

The TS/TD supplied bridge exit

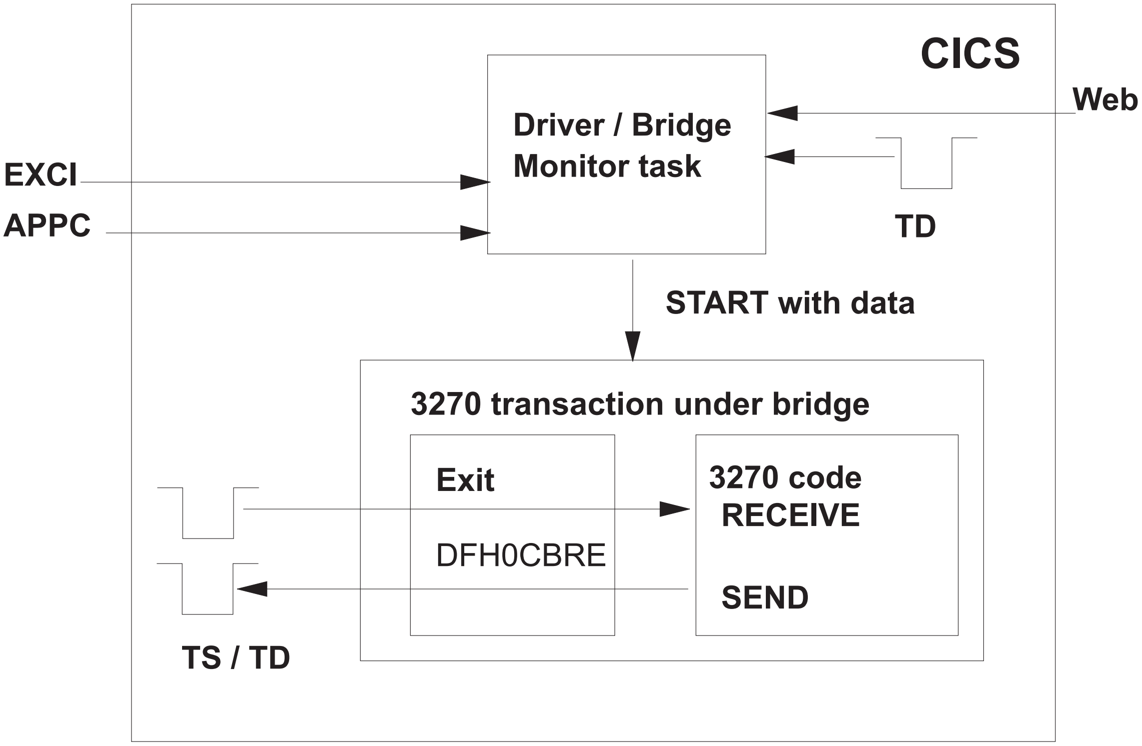


Figure 22. The supplied TS/TD bridge exit

This exit, DFH0CBRE (and its associated formatter, DFH0CBRF),is supplied in COBOL source. It uses CICS temporary storage (TS) or transient data (TD) queues to pass input and output from and to the client application (another CICS application).

You can modify this exit to support other transport mechanisms.

DFH0CBRE is the most general of the supplied exits. To run a transaction using this exit, you simply issue a START TRANSID() BRDATA() BREXIT(DFH0CBRE) command. The formats of the interfaces and messages used in this bridge exit are described in“Data formats for the supplied bridge exits” on page 59. The TS/TD supplied bridge exit is an example of the Direct model.

© Copyright IBM Corp. 1994, 1998

Using the DFH0CBRE exit

Before using the DFH0CBRE bridge exit to run an existing CICS transaction, you need to perform the following steps:

1. Translate and compile DFH0CBRE and DFH0CBRE using an appropriate COBOL compiler.
2. Link DFH0CBRE and DFH0CBRF as standard CICS application programs into a CICS load library used by the CICS system on which you will run the user transaction.
3. Define DFH0CBRE and DFH0CBRF as programs to CICS. Sample definitions are supplied in the DFH$BR group.
4. Define and install transient data queues if required.
5. Write a client requester program. This is an application program, written by you in any of the supported languages (Assembler, COBOL, PLI, C).

   It should issue a START TRANSID BREXIT BRDATA command to start the required user transaction, passing BRDATA data formatted as described in "BRDATA format" on page 60.

   It should write data to the TS/TD queue named, to provide 3270 terminal data for the 3270 user transaction, formatted as described in "Message data format" on page 61, and read responses back from the queue.
6. Define and install the end-user requester program and transaction.
7. Run the end-user requester transaction.

The Web bridge exit

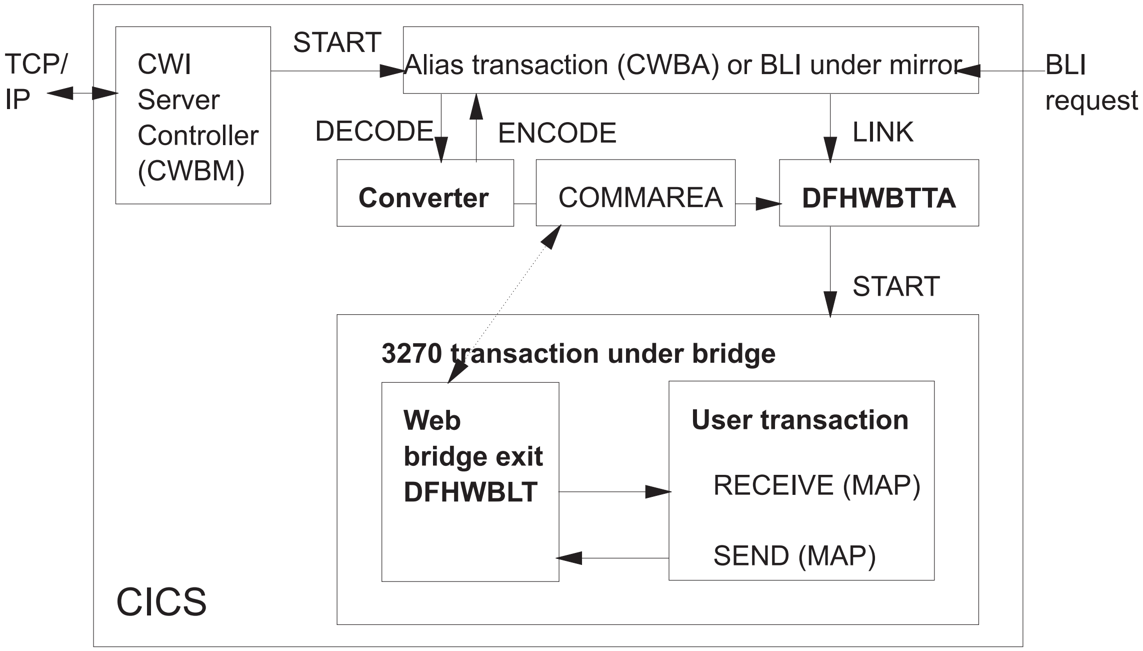


Figure 23. The Web supplied bridge exit

This exit, DFHWBLT, is an object code exit that allows you to access a CICS transaction from the World Wide Web. It uses the CICS Web support described in the *CICS Internet Guide*.

DFHWBLT works in conjunction with a long running server transaction, CWBM, which monitors the TCP/IP interface for incoming requests, and a bridge monitor, DFHWBTTA. The formats of the interfaces and messages used in this bridge exit are the same as in the TS/TD bridge and are described in "Data formats for the supplied bridge exits". The Web bridge exit is an example of the two task monitor model.

Using the Web bridge exit

Use of this exit is fully described in the *CICS Internet Guide*. After installation, you do not need to provide any user code to use this solution, but it can be customized.

A more detailed implementation description can be found in the Redbook *CICS Transaction server for OS/390: Web Interface and 3270 Bridge*, order number SG24-5243.

Data formats for the supplied bridge exits

There are two interfaces between the supplied bridge exits and the client application. They are:

BRDATA

BRDATA is passed to the bridge exit when the user transaction and its associated bridge exit are started with a START TRANSID BREXIT BRDATA command. The main purpose of BRDATA is to pass information required for bridge exit initialization (facility token and facilitylike) and data required to identify the message queue and message. The names of the parameters and constants in the BRDATA data, translated into appropriate forms for the different programming languages supported, are defined in the following copybook files supplied as part of the 3270 bridge.

Table 4. START data copybooks

Language	Definition	Constants
Assembler	DFHBRSDD	DFHBRSCD
C	DFHBRSDH	DFHBRSCH
PL/I	DFHBRSDL	DFHBRSCL
COBOL	DFHBRSDO	DFHBRSCO

Messages

The messages passed between the bridge exit and the client application via TS and TD queues, to satisfy SEND and RECEIVE requests from the 3270 user application. The names of the parameters and constants in the message data, translated into appropriate forms for the different programming languages supported, are defined in the following copybook files supplied as part of the 3270 bridge.

Table 5. Message data copybooks

Language	Definition
Assembler	DFHBRMQD
C	DFHBRMQH
PL/I	DFHBRMQL
COBOL	DFHBRMQO

All user messages must begin with a standard message header, MQCIH. The names of the parameters and constants in MQCIH, translated into appropriate forms for the different programming languages supported, are defined in the following copybook files supplied as part of the 3270 bridge.

Table 6. MQCIH message header copybooks

Language	Definition	Constants
Assembler	DFHBRMHD	DFHBRMCD
C	DFHBRMHH	DFHBRMCH
PL/I	DFHBRMHL	DFHBRMCL
COBOL	DFHBRMHO	DFHBRMCO

BRDATA format

An EXEC CICS START BREXIT() BRDATA() command is issued to run a transaction in the bridge environment. To use the supplied bridge exits, the BRDATA data must conform to the following format.

Note that the parameter names shown are in COBOL format with a dash '-' name separator, rather than the underscore '_' required for other languages.

Offset Hex	Type	Len	Name
(0)	STRUCTURE	64	DFHBRSD
(0)	STRUCTURE	16	BRSD-HEADER-DATA
(0)	CHARACTER	4	BRSD-STRUCID
(4)	FULLWORD	4	BRSD-VERSION
(8)	FULLWORD	4	BRSD-STRUCLENGTH
(C)	CHARACTER	4	reserved
(10)	STRUCTURE	40	BRSD-QUEUE-NAMES
(10)	STRUCTURE	20	BRSD-OUTPUT-QUEUE
(10)	CHARACTER	2	BRSD-OUTPUT-TYPE
(12)	CHARACTER	2	reserved
(14)	STRUCTURE	16	BRSD-TS-OUTPUT-QUEUE
(14)	CHARACTER	4	BRSD-TD-OUTPUT-QUEUE
(18)	CHARACTER	12	reserved
(24)	STRUCTURE	20	BRSD-INPUT-QUEUE
(24)	CHARACTER	2	BRSD-INPUT-TYPE
(26)	HALFWORD	2	BRSD-INPUT-ITEM
(28)	STRUCTURE	16	BRSD-TS-INPUT-QUEUE
(28)	CHARACTER	4	BRSD-TD-INPUT-QUEUE
(2C)	CHARACTER	4	reserved
(30)	CHARACTER	8	reserved
(38)	CHARACTER	8	BRSD-FACILITY-TOKEN
(40)	CHARACTER	4	BRSD-FACILITYLIKE
(44)	CHARACTER	4	reserved

BRSD-STRUCID
An eye-catcher. This must be set to the constant brsd-struc-id in the DFHBRSCx copy book.

BRSD-VERSION
The version number of the start data. This must be set to the constant brsd-version-2 in the DFHBRSCx copy book.

BRSD-STRUCLENGTH
The length of the start data. This must be set to the constant brsd-length-1

BRSD-OUTPUT-TYPE
An indicator showing whether the output queue is temporary storage (TS) or Transient data (TD). This must be set to either the constant BRSD-TS or BRSD-TD in the DFHBRSCx copy book.

BRSD-TS-OUTPUT-QUEUE
The 8 or 16-byte name of the queue, if the output queue is a TS queue.

BRSD-TD-OUTPUT-QUEUE
The 4 byte name of the queue, if the output queue is a TD queue.

BRSD-INPUT-TYPE
The queue type, either TS or TD. This must be set to the constant BRSD-TS or BRSD in the DFHBRSCx copy book.

BRSD-INPUT-ITEM
The first item number in the TS queue. A value of binary zeroes means ’next’.

BRSD-TS-INPUT-QUEUE
The 8 or 16-byte name of the queue, if the input queue is a TS queue.

BRSD-TD-INPUT-QUEUE
The 4 byte name of the queue, if the input queue is a TD queue.

BRSD-FACILITY-TOKEN
The 8-byte bridge facility token. This value is nulls for a new facility.

BRSD-FACILITYLIKE
The name of an installed terminal that is to be used as a model for the bridge facility, when BRSD-FACILITY-TOKEN is nulls.

Message data format

Messages sent to the supplied bridge exits by the client application must all conform to the following format:

All messages must begin with a message header, MQCIH, followed by a variable number of self defining *vectors*. Each vector contains a vector header that identifies the vector type and length so that the bridge exit can interpret the content of each message.

There are three types of vector:

OUTBOUND REPLY
This type of vector flows from the user transaction to the end-user transaction carrying a reply. No further input is requested.

OUTBOUND REQUEST
This type of vector flows from the user transaction to the end-user transaction requesting further input. There is only one in each message and it must be last.

INBOUND
This type of vector flows from the end-user transaction to the user transaction carrying data to satisfy a user transaction RECEIVE or RETRIEVE. Processing is more efficient if the order of RETRIEVE and RECEIVE vectors in the first message matches the order of CICS commands in the user transaction, but this is not essential.

RETRIEVE vectors can only flow in the first message. If there are more RETRIEVEs than can fit in a single message, they can be sent in multiple messages , provided that:

- All of the RETRIEVE vectors are sent before any RECEIVE vectors
- All of the messages are written to the queue before the START BREXIT is issued.

Note: All fields have a minimum length of four bytes. Data is left justified and padded with blanks if necessary. Field names are shown with '-' (dash) separators, as used in COBOL. Other languages require '_' (underscore) separators.

Header MQCIH	Vector#1 BRMQ	...	Vector#n BRMQ

Figure 24. Message format

MQCIH Message header:

Offset	Type	Len	Name
Hex			
(0)	STRUCTURE	180	MQCIH
(0)	CHARACTER	4	MQCIH-STRUCID
(4)	FULLWORD	4	MQCIH-VERSION
(8)	FULLWORD	4	MQCIH-STRUCLENGTH
(C)	BINARY	8	reserved
(14)	CHARACTER	8	reserved
(1C)	BINARY	20	reserved
(30)	FULLWORD	4	MQCIH-GETWAITINTERVAL
(34)	BINARY	8	reserved
(3C)	FULLWORD	4	MQCIH-FACILITYKEEPTIME
(40)	FULLWORD	4	MQCIH-ADSDESCRIPTOR
(44)	FULLWORD	4	MQCIH-CONVERSATIONALTASK
(48)	FULLWORD	4	MQCIH-TASKENDSTATUS
(4C)	CHARACTER	8	MQCIH-FACILITY
(54)	CHARACTER	4	reserved
(58)	CHARACTER	4	MQCIH-ABENDCODE
(5C)	CHARACTER	8	MQCIH-AUTHENTICATOR
(64)	CHARACTER	24	reserved
(7C)	CHARACTER	4	MQCIH-TRANSACTIONID
(80)	CHARACTER	4	MQCIH-FACILITYLIKE
(84)	CHARACTER	4	MQCIH-ATTENTIONID
(88)	CHARACTER	4	MQCIH-STARTCODE
(8C)	CHARACTER	4	MQCIH-CANCELCODE
(90)	CHARACTER	4	MQCIH-NEXTTRANSACTIONID
(94)	CHARACTER	16	reserved
(A4)	FULLWORD	4	MQCIH-CURSORPOSITION
(A8)	FULLWORD	4	MQCIH-ERROROFFSET
(AC)	FULLWORD	4	MQCIH-INPUTITEM
(B0)	BINARY	4	reserved

MQCIH-STRUCID

The identifier for the CICS information header structure. This is an input field.

MQCIH-VERSION

The version number for the CICS information header structure. This must be MQCIH-VERSION-2. This is an input field.

MQCIH-STRUCLENGTH

The length of the CICS information header structure. This must be MQCIH-LENGTH-2. This is an input field.

MQCIH-GETWAITINTERVAL

The maximum wait interval for message input (in milliseconds). This is an input field.

MQCIH-FACILITYKEEPTIME

The length of time that the bridge facility will be kept after the user transaction has ended (in seconds). This is an input field.

MQCIH-ADSDESCRIPTOR

An indicator specifying whether ADS descriptors should be sent on SEND and RECEIVE BMS requests. The MQCADSD-MSGFORMAT value indicates that the ***long***form of the ADSD is used. Valid values are:

MQCADSD-NONE

MQCADSD-SEND

MQCADSD-RECV

MQCADSD-SEND+MQCADSD-RECV

MQCADSD-SEND+MQCADSD-RECV+MQCADSD-MSGFORMAT

This is an input field.

MQCIH-CONVERSATIONALTASK

An indicator specifying whether the task should be allowed to issue requests for more information, or should abend. Valid values are:

MQCCT-YES

MQCCT-NO

This is an input field.

MQCIH-TASKENDSTATUS

The value returned on output messages showing the status of the user transaction. One of the following values will be returned:

MQCTES-NOSYNC

The user transaction has not yet completed, and has not syncpointed.

MQCTES-COMMIT

The user transaction has not yet completed, but has syncpointed the first unit of work.

MQCTES-BACKOUT

The user transaction has not yet completed. The current unit of work will be backed out.

MQCTES–ENDTASK

The user transaction has ended (or abended).

This is an output field.

MQCIH-FACILITY

The 8-byte bridge facility token. This value is returned on output messages when a keep time is specified. This is an input/output field.

MQCIH-ABENDCODE
The abend code returned if the transaction abends, otherwise this field is set to blanks. This is an output field.

MQCIH-AUTHENTICATOR
The password or passticket for the specified USERID. This is an input field.

MQCIH-TRANSACTIONID
The transaction identifier of the user transaction.

MQCIH-FACILITYLIKE
The name of an installed terminal that is to be used as a model for the bridge facility. A value of blanks means that the FACILITYLIKE is taken from the bridge transaction profile definition, or a default value is used. This is an input field.

MQCIH-ATTENTIONID
The initial value of the AID key when the transaction is started. This is a 1-byte value, left justified. It is an input field.

MQCIH-STARTCODE
An indicator set on output from CICS with the start code that is appropriate for the next transaction. Valid values are:

MQCSC-START

MQCSC-STARTDATA

MQCSC-TERMINPUT

MQCSC-NONE

This is an input field.

MQCIH-CANCELCODE
The abend code to be used to terminate the transaction (normally a conversational transaction that is requesting more data). Otherwise this field is set to blanks. This is an input field.

MQCIH-NEXTTRANSACTIONID
The name of the next transaction returned by the user transaction (usually by EXEC CICS RETURN TRANSID). If there is no next transaction, this field is set to blanks. This is an output field.

MQCIH-CURSORPOSITION
The initial cursor position when the transaction is started. Subsequently, for conversational transactions, the cursor position is in the RECEIVE vector. This is an input field.

MQCIH-ERROROFFSET
The position of invalid data detected by the bridge exit. This field provides the offset from the start of the message to the location of the invalid data.

MQCIH-INPUTITEM
The current TS queue item number being processed by the bridge exit.

Standard header for all vectors:

Offset	**Type**	**Len**	**Name**
Hex			
(0)	STRUCTURE	16	BRMQ-VECTOR-HEADER
(0)	FULLWORD	4	BRMQ-VECTOR-LENGTH
(4)	CHARACTER	4	BRMQ-VECTOR-DESCRIPTOR
(8)	CHARACTER	4	BRMQ-VECTOR-TYPE
(C)	CHARACTER	4	BRMQ-VECTOR-VERSION

BRMQ-VECTOR-LENGTH

The length of the vector. On output, this is always rounded up to the next multiple of 4, to facilitate full word alignment of subsequent vectors in the message. On input to the bridge exit, it is advisable to round up to the next multiple of 4 for the same reason.

BRMQ-VECTOR-DESCRIPTOR

An indicator to define the CICS command associated with this vector. Valid values are:

0402	RECEIVE
0404	SEND
0406	CONVERSE
0418	ISSUE ERASEAUP
100A	RETRIEVE
1802	RECEIVE MAP
1804	SEND MAP
1806	SEND TEXT
1812	SEND CONTROL

BRMQ-VECTOR-TYPE

The vector type. Valid values are:

I	Inbound to the bridge exit.
O	Outbound from the bridge exit.

BRMQ-VECTOR-VERSION

The vector version number. Valid values are:

X’00000000’

The first version.

Outbound reply vectors

Outbound vectors carry reply messages flowing from the user transaction to the end-user transaction.

SEND:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	48	BRMQ-SEND
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-SE-ERASE-INDICATOR
(14)	CHARACTER	4	BRMQ-SE-CTLCHAR
(18)	CHARACTER	4	BRMQ-SE-STRFIELD-INDICATOR
(1C)	CHARACTER	4	BRMQ-SE-DEFRESP-INDICATOR
(20)	CHARACTER	4	BRMQ-SE-INVITE-INDICATOR
(24)	CHARACTER	4	BRMQ-SE-LAST-INDICATOR
(28)	CHARACTER	4	BRMQ-SE-WAIT-INDICATOR
(2C)	FULLWORD	4	BRMQ-SE-DATA-LEN
(30)	CHARACTER		variable length data

BRMQ-SE-ERASE-INDICATOR

The type of ERASE specified by the CICS SEND command that caused the exit to be called. Valid character values, left justified, are:

N No ERASE.

E ERASE.

A ERASE ALTERNATE.

D ERASE DEFAULT.

BRMQ-SE-CTLCHAR
The CTLCHAR value specified by the SEND command that caused the exit to be called. If CTLCHAR is not specified, the default X’C3’ is sent.

BRMQ-SE-STRFIELD-INDICATOR
The presence of STRFIELD on the SEND command. Valid character values, left justified, are:

Y STRFIELD specified.

N STRFIELD not specified.

BRMQ-SE-DEFRESP-INDICATOR
The presence of DEFRESP on the SEND command that caused the exit to be called. Valid character values, left justified, are:

Y DEFRESP specified.

N DEFRESP not specified.

BRMQ-SE-INVITE-INDICATOR
The presence of INVITE on the send command that caused the exit to be called. Valid character values, left justified, are:

Y INVITE specified.

N INVITE not specified.

BRMQ-SE-LAST-INDICATOR
The presence of LAST on the SEND command that caused the exit to be called. Valid character values, left justified, are:

Y LAST specified.

N LAST not specified.

BRMQ-SE-WAIT-INDICATOR
The presence of WAIT on the SEND command that caused the exit to be called. Valid character values, left justified, are:

Y WAIT specified.

N WAIT not specified.

BRMQ-SE-DATA-LEN
The length of the data associated with the FROM option of the SEND command that caused the exit to be called. This is explicitly defined in the LENGTH or FLENGTH option, or derived from the length of the field.

data
Character field of length BRMQ-SE-DATA-LEN containing the data addressed by the FROM option of the SEND command.

SEND CONTROL: The fields in this vector are included also in SEND MAP and SEND TEXT.

Offset	**Type**	**Len**	**Name**
Hex			
(0)	STRUCTURE	52	BRMQ-SEND-CONTROL

Offset Hex	Type	Len	Name
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-SC-ERASE-INDICATOR
(14)	CHARACTER	4	BRMQ-SC-ERASEAUP-INDICATOR
(18)	CHARACTER	4	BRMQ-SC-FREEKB-INDICATOR
(1C)	CHARACTER	4	BRMQ-SC-ALARM-INDICATOR
(20)	CHARACTER	4	BRMQ-SC-FRSET-INDICATOR
(24)	CHARACTER	4	BRMQ-SC-LAST-INDICATOR
(28)	CHARACTER	4	BRMQ-SC-WAIT-INDICATOR
(2C)	FULLWORD	4	BRMQ-SC-CURSOR
(30)	CHARACTER	4	BRMQ-SC-MSR-DATA

BRMQ-SC-ERASE-INDICATOR
The type of ERASE specified by the CICS BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

N No ERASE

E ERASE

A ERASE ALTERNATE

D ERASE DEFAULT

BRMQ-SC-ERASEAUP-INDICATOR
The presence of ERASEAUP on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

Y ERASEAUP specified.

N ERASEAUP not specified.

BRMQ-SC-FREEKB-INDICATOR
The presence of FREEKB on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

Y FREEKB specified.

N FREEKB not specified.

BRMQ-SC-ALARM-INDICATOR
The presence of ALARM on the BMS SEND command that caused the exit to be called. Valid values are:

Y ALARM specified.

N ALARM not specified.

BRMQ-SC-FRSET-INDICATOR
The presence of FRSET on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

Y FRSET specified.

N FRSET not specified.

BRMQ-SC-LAST-INDICATOR
The presence of LAST on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

Y LAST specified.

N LAST not specified.

BRMQ-SC-WAIT-INDICATOR
The presence of WAIT on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

Y WAIT specified.

N WAIT not specified.

BRMQ-SC-CURSOR
The presence of CURSOR or CURSOR(*data-value*) on the BMS SEND command that caused the exit to be called. Valid character values, left justified, are:

-1 CURSOR specified with dynamic cursor positioning.

-2 Neither CURSOR nor CURSOR(*data-value*) specified.

other The value of CURSOR(*data-value*) specified.

BRMQ-SC-MSR-DATA
The value of the MSR option specified on the BMS SEND command that caused the exit to be called. Valid values are:

X’00000000’
MSR option not specified.

other The value of the MSR option specified.

SEND MAP:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	88	BRMQ-SEND-MAP
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-SC-ERASE-INDICATOR
(14)	CHARACTER	4	BRMQ-SC-ERASEAUP-INDICATOR
(18)	CHARACTER	4	BRMQ-SC-FREEKB-INDICATOR
(1C)	CHARACTER	4	BRMQ-SC-ALARM-INDICATOR
(20)	CHARACTER	4	BRMQ-SC-FRSET-INDICATOR
(24)	CHARACTER	4	BRMQ-SC-LAST-INDICATOR
(28)	CHARACTER	4	BRMQ-SC-WAIT-INDICATOR
(2C)	FULLWORD	4	BRMQ-SC-CURSOR
(30)	CHARACTER	4	BRMQ-SC-MSR-DATA
(34)	CHARACTER	8	BRMQ-SM-MAPSET
(3C)	CHARACTER	8	BRMQ-SM-MAP
(44)	CHARACTER	4	BRMQ-SM-DATA-INDICATOR
(48)	FULLWORD	4	BRMQ-SM-DATA-LEN
(4C)	FULLWORD	4	BRMQ-SM-DATA-OFFSET
(50)	FULLWORD	4	BRMQ-SM-ADSD-LEN
(54)	FULLWORD	4	BRMQ-SM-ADSD-OFFSET
(58)	CHARACTER		variable length data

Fields **BRMQ-SC-ERASE-INDICATOR** to **BRMQ-SC-MSR-DATA** are defined in “SEND CONTROL” on page 66.

BRMQ-SM-MAPSET
The value of the MAPSET option specified by the SEND MAP command that caused the exit to be called.

BRMQ-SM-MAP
The value of the MAP option specified by the SEND MAP command that caused the exit to be called.

BRMQ-SM-DATA-INDICATOR

The presence of MAPONLY and DATAONLY options on the SEND MAP command that caused the exit to be called. Valid character values, left justified, are:

D DATAONLY specified.

M MAPONLY specified.

N Neither DATAONLY nor MAPONLY specified.

BRMQ-SM-DATA-LEN

The length of the data associated with the FROM option on the SEND MAP command that caused the exit to be called. This is the length of the symbolic map or ADS (application data structure).

BRMQ-SM-DATA-OFFSET

The offset from the beginning of the SEND MAP vector to the data associated with the FROM option of the SEND MAP command that caused the exit to be called.

BRMQ-SM-ADSD-LEN

The length of the ADS descriptor associated with this map. This length is zero if the ADSD is not available, or was not requested. See “ADS descriptor area” on page 100 for a description of the ADS.

BRMQ-SM-ADSD-OFFSET

The offset from the beginning of the SEND MAP vector to the ADSD. This is zero if the ADSD is not available, or was not requested.

data

Character field of length BRMQ-SM-DATA-LEN containing the data specified by the FROM option of the SEND MAP command, in ADS (Application Data Structure) format. This is followed by an ADS descriptor for this data, of length BRMQ-SM-ADSD-LEN, if an ADS descriptor was requested.

SEND TEXT:

Offset	Type	Len	Name
Hex			
(0)	STRUCTURE	60	BRMQ-SEND-TEXT
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-SC-ERASE-INDICATOR
(14)	CHARACTER	4	BRMQ-SC-ERASEAUP-INDICATOR
(18)	CHARACTER	4	BRMQ-SC-FREEKB-INDICATOR
(1C)	CHARACTER	4	BRMQ-SC-ALARM-INDICATOR
(20)	CHARACTER	4	BRMQ-SC-FRSET-INDICATOR
(24)	CHARACTER	4	BRMQ-SC-LAST-INDICATOR
(28)	CHARACTER	4	BRMQ-SC-WAIT-INDICATOR
(2C)	FULLWORD	4	BRMQ-SC-CURSOR
(30)	CHARACTER	4	BRMQ-SC-MSR-DATA
(34)	CHARACTER	4	BRMQ-ST-TEXT-TYPE
(38)	FULLWORD	4	BRMQ-ST-DATA-LEN
(3C)	CHARACTER		variable length data

Fields **BRMQ-SC-ERASE-INDICATOR** to **BRMQ-SC-MSR-DATA** are defined in “SEND CONTROL” on page 66.

BRMQ-ST-TEXT-TYPE

The presence of MAPPED or NOEDIT options on the SEND TEXT command that caused the exit to be called. Valid character values, left justified, are:

M MAPPED specified.

N NOEDIT specified.

blank Neither MAPPED nor NOEDIT specified.

BRMQ-ST-DATA-LEN
The length of the text associated with the FROM option of the SEND TEXT command that caused the exit to be called.

data
Character field of length BRMQ-ST-DATA-LEN containing the data specified by the FROM option of the SEND TEXT command that caused the exit to be called. For SEND TEXT MAPPED, the additional 4 bytes created by the SEND MAP command, are included in this field, but the additional length is not included in BRMQ-ST-DATA-LEN.

ISSUE ERASEAUP:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	20	BRMQ-ISSUE-ERASEAUP
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-IE-WAIT-INDICATOR

BRMQ-IE-WAIT-INDICATOR
The presence of the WAIT option on the ISSUE ERASEAUP command that caused the exit to be called. Valid character values, left justified, are:

Y WAIT specified.

N WAIT not specified.

Outbound request vectors

Outbound request vectors flow from the user transaction to the end-user transaction requesting further input. There is only one in each message and it must be last.

RECEIVE REQUEST:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	20	BRMQ-RECEIVE-REQUEST
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-RER-BUFFER-INDICATOR

BRMQ-RER-BUFFER-INDICATOR
The presence of the BUFFER option on the RECEIVE request that caused the exit to be called. Valid character values, left justified, are:

Y BUFFER specified.

N BUFFER not specified.

RECEIVE MAP REQUEST:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	36	BRMQ-RECEIVE-MAP-REQUEST
(0)	CHARACTER	16	Header
(10)	CHARACTER	8	BRMQ-RMR-MAPSET
(18)	CHARACTER	8	BRMQ-RMR-MAP

Offset Hex	Type	Len	Name
(20)	FULLWORD	4	BRMQ-RMR-ADSD-LEN
(24)	CHARACTER		variable length data

BRMQ-RMR-MAPSET
The value of the MAPSET option on the RECEIVE MAP command that caused the exit to be called.

BRMQ-RMR-MAP
THE value of the MAP option on the RECEIVE MAP command that caused the exit to be called.

BRMQ-RMR-ADSD-LEN
The length of the ADS descriptor associated with this map. This length is zero if the ADSD is not available, or was not requested (MQCIH-ADSDESCRIPTOR set to MQCADSD-NONE).

data
The ADS descriptor associated with the requested map. No data is sent if BRMQ-RMR-ADSD-LEN is zero.

CONVERSE REQUEST:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	48	BRMQ-CONVERSE-REQUEST
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-COR-ERASE-INDICATOR
(14)	CHARACTER	4	BRMQ-COR-CTLCHAR
(18)	CHARACTER	4	BRMQ-COR-STRFIELD-INDICATOR
(1C)	CHARACTER	4	BRMQ-COR-DEFRESP-INDICATOR
(20)	CHARACTER	12	(reserved)
(2C)	FULLWORD	4	BRMQ-COR-DATA-LEN
(30)	CHARACTER		variable length data

BRMQ-COR-ERASE-INDICATOR
The type of ERASE specified by the CICS SEND command that caused the exit to be called. Valid character values, left justified, are:

N No ERASE.

E ERASE.

A ERASE ALTERNATE.

D ERASE DEFAULT.

BRMQ-COR-CTLCHAR
The CTLCHAR value specified by the SEND command that caused the exit to be called. If CTLCHAR is not specified, the default X’C3’ is sent.

BRMQ-COR-STRFIELD-INDICATOR
The presence of STRFIELD on the SEND command. Valid character values, left justified, are:

Y STRFIELD specified.

N STRFIELD not specified.

BRMQ-COR-DEFRESP-INDICATOR
The presence of DEFRESP on the SEND command that caused the exit to be called. Valid character values, left justified, are:

Y DEFRESP specified.

N DEFRESP not specified.

BRMQ-COR-DATA-LEN
The length of the data associated with the FROM option of the SEND command that caused the exit to be called. This is explicitly defined in the LENGTH or FLENGTH option, or derived from the length of the field.

data
Character field of length BRMQ-COR-DATA-LEN containing the data addressed by the FROM option of the CONVERSE command.

Inbound vectors

Inbound vectors flow from the end-user transaction to the user transaction carrying data to satisfy a user transaction RECEIVE, CONVERSE, or RETRIEVE.

RECEIVE:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	36	BRMQ-RECEIVE
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-RE-TRANSMIT-SEND-AREAS
(14)	CHARACTER	4	BRMQ-RE-BUFFER-INDICATOR
(18)	CHARACTER	4	BRMQ-RE-AID
(1C)	FULLWORD	4	BRMQ-RE-CPOSN
(20)	FULLWORD	4	BRMQ-RE-DATA-LEN
(24)	CHARACTER		variable length data

BRMQ-RE-TRANSMIT-SEND-AREAS
A flag indicating whether previously generated, but not yet transmitted, SEND areas are to be preserved. Valid character values, left justified, are:

Y Preserve untransmitted SEND areas.

N Delete untransmitted SEND areas.

BRMQ-RE-BUFFER-INDICATOR
A flag indicating whether the data provided in the inbound vector is in a format to be received by a CICS RECEIVE command with the BUFFER option. Valid character values, left justified, are:

Y Data in BUFFER format.

N Data not in BUFFER format.

BRMQ-RE-AID
The AID key that was simulated to generate data in response to the RECEIVE command. The first byte of this field contains equivalent values to EIBAID, as defined by DFHAID. The remaining three bytes are padded with blanks. This value is inserted into the EIBAID field.

BRMQ-RE-CPOSN
The offset of the cursor at the time the RECEIVE data was generated. This value is inserted in EIBCPOSN for the transaction issuing the EXEC CICS RECEIVE.

BRMQ-RE-DATA-LEN
The length of the data provided in this vector in response to the RECEIVE command. This value is copied into the LENGTH or FLENGTH field.

data
Character field of length BRMQ-RE-DATA-LEN to be copied into the INTO area, or referenced by the SET option, of the RECEIVE command that caused the exit to be called.

RECEIVE MAP:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	48	BRMQ-RECEIVE-MAP
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-RM-TRANSMIT-SEND-AREAS
(14)	CHARACTER	8	BRMQ-RM-MAPSET
(1C)	CHARACTER	8	BRMQ-RM-MAP
(24)	CHARACTER	4	BRMQ-RM-AID
(28)	FULLWORD	4	BRMQ-RM-CPOSN
(2C)	FULLWORD	4	BRMQ-RM-DATA-LEN
(30)	CHARACTER		variable length data

BRMQ-RM-TRANSMIT-SEND-AREAS
A flag indicating whether previously generated, but not yet transmitted, SEND areas are to be preserved. Valid character values, left justified, are:

Y Preserve untransmitted SEND areas.

N Delete untransmitted SEND areas.

BRMQ-RM-MAPSET
The name of the MAPSET to be used to present the data. If this is blank, the data is to be presented to the next RECEIVE MAP command, regardless of the MAPSET value specified by the command.

BRMQ-RM-MAP
The name of the MAP to be used to present the data. If this is blank, the data is to be presented to the next RECEIVE MAP command, regardless of the MAP value specified by the command.

BRMQ-RM-AID
The AID key that was simulated to generate data in response to the RECEIVE command. The first byte of this field contains equivalent values to EIBAID, as defined by DFHAID. The remaining three bytes are padded with blanks. This value will be inserted into the EIBAID field for the transaction issuing the EXEC CICS RECEIVE MAP.

BRMQ-RM-CPOSN
The offset of the cursor at the time the RECEIVE data was generated. This value will be inserted into the EIBCPOSN field for the transaction issuing the EXEC CICS RECEIVE MAP.

BRMQ-RM-DATA-LEN
The length of the data provided in this vector in response to the RECEIVE MAP command that caused the exit to be called. This value is copied into the LENGTH or FLENGTH field.

data
Character field of length BRMQ-RM-DATA-LEN, in ADS (Application Data Structure) format equivalent to the MAP and MAPSET specified by the RECEIVE command.

CONVERSE:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	36	BRMQ-CONVERSE
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-CO-TRANSMIT-SEND-AREAS
(14)	CHARACTER	4	reserved
(18)	CHARACTER	4	BRMQ-CO-AID
(1C)	FULLWORD	4	BRMQ-CO-CPOSN
(20)	FULLWORD	4	BRMQ-CO-DATA-LEN
(24)	CHARACTER		variable length data

BRMQ-CO-TRANSMIT-SEND-AREAS
A flag indicating whether previously generated, but not yet transmitted, SEND areas are to be preserved. Valid character values, left justified, are:

Y Preserve untransmitted SEND areas.

N Delete untransmitted SEND areas.

BRMQ-CO-AID
The AID key that was simulated to generate data in response to the RECEIVE command. The first byte of this field contains equivalent values to EIBAID, as defined by DFHAID. The remaining three bytes are padded with blanks. This value is inserted into the EIBAID field.

BRMQ-CO-CPOSN
The offset of the cursor at the time the RECEIVE data was generated. This value is inserted in EIBCPOSN for the transaction issuing the EXEC CICS RECEIVE.

BRMQ-CO-DATA-LEN
The length of the data provided in this vector in response to the RECEIVE command. This value is copied into the LENGTH or FLENGTH field.

data
Character field of length BRMQ-CO-DATA-LEN to be copied into the INTO area, or referenced by the SET option, of the CONVERSE command that caused the exit to be called.

RETRIEVE:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	36	BRMQ-RETRIEVE
(0)	CHARACTER	16	Header
(10)	CHARACTER	4	BRMQ-RT-RTRANSID
(14)	CHARACTER	4	BRMQ-RT-RTERMID
(18)	CHARACTER	8	BRMQ-RT-QUEUE
(20)	FULLWORD	4	BRMQ-RT-DATA-LEN
(24)	CHARACTER		variable length data

BRMQ-RT-RTRANSID
The value to be returned in the RTRANSID field, to the program that issued the RETRIEVE. A blank indicates that there is no RTRANSID.

BRMQ-RT-RTERMID
The value to be returned in the RTERMID field, to the program that issued the RETRIEVE. A blank indicates that there is no RTERMID.

BRMQ-RT-QUEUE
The value to be returned in the QUEUE field, to the program that issued the RETRIEVE. A blank indicates that there is no QUEUE.

BRMQ-RT-DATA-LEN
The length of the data provided in this vector in response to the RETRIEVE command that caused the exit to be called. This value is copied into the LENGTH or FLENGTH field.

data
Character field of length BRMQ-RT-DATA-LEN to be copied into the INTO area, or referenced by the SET option of the RETRIEVE command.

Note: The RETRIEVE vector is only valid in the first inbound message. It is ignored in other messages.

Chapter 6. Writing your own bridge programs

If you do not want to use the IBM supplied bridge exits and monitors, you can modify them, or write your own. This chapter tells you how you can modify the supplied exits and about the interfaces that are defined by CICS that you must support in your own programs. It covers the following topics:

Designing your own bridge solution

The bridge mechanism is very flexible, but most implementations fall into a few basic models.

"Implementing a 3270 bridge environment" on page 32 tells you how to identify the model that fits the requirements of your system and applications. It then presents examples of each model, which you can use as checklists when preparing your own client programs, bridge exits and monitors.

Is a new bridge exit needed?

If you want to use a transport mechanism other than TS or TD, or you want to use a message format that doesn't have MQCIH as the message header, it may be simpler to modify DFH0CBRE, rather than write your own complete solution.

Changing the transport mechanism
The supplied exit has three sections marked ″transport mechanism specific″. These should be changed to replace the TS/TD code with transport specific code. The brdata may also need to change. This is contained in the DFHBRSCO and DFHBRSDO copybooks, and in the INIT routine.

Changing the message header
The only user of the MQCIH message header is the DFH0CBRE bridge exit. It is not used in the formatter, or the common code. Therefore if you want to simplify the MQCIH, for example to hard code most of the values in the exit, and have a reduced header, this can be done by changing the DFHMQMHO and DFHMQMCO copy books in the ″message header″ sections and routines marked as ″MSG-HDR SPECIFIC ROUTINES″.

If you want a much simpler interface which is tailored for a specific(type) of application, then you should consider writing your own bridge exit that is specifically designed for a single transaction (or a number of transactions with identical interfaces).

An example of this approach is described in the Dallas System Center Redbook.

© Copyright IBM Corp. 1994, 1998

Is a new formatter needed?

If you want to use a message format that doesn’t use BRMQ message vectors, it may be simpler to modify DFH0CBRF, rather than write your own complete solution.

Changing the message vectors
The only user of the BRMQ message vectors is the DFH0CBRF formatter. It is not used in the bridge exit, or the common code. Therefore if you want to simplify the BRMQ vectors, for example to hard code values in the exit, and have a reduced header, this can be done by changing the DFHBRMQ copy book in the ″message structures″ section, and occurances of the BRMQ field

If you want a much simpler interface which is tailored for a specific(type) of application, then you should consider writing your own formatter that is specifically designed for a single transaction (or a number of transactions with identical interfaces). The data can be read by the bridge exit as normal, but the formatter can be tailored for only those API interfaces that are specifically required.

Writing your own bridge exit

Your bridge exit is always called for the following requests:

- User transaction initialization
- User transaction bind
- User transaction termination
- User transaction abnormal termination
- Syncpoint (optional)

If a formatter is specified in the INIT call, then the bridge exit is also called for Read and Write message requests.

If a formatter is specified in the INIT call, it is called for the following requests. Otherwise, if a formatter is not used, the bridge exit is also called for these requests:

- SEND (Terminal Control and BMS)
- RECEIVE (Terminal Control and BMS)
- CONVERSE
- FREE
- ISSUE DISCONNECT
- ISSUE ERASEAUP
- RETRIEVE (in some cases)

Transaction calls to the bridge exit

The following calls are made at transaction initialization, termination, syncpoint and abend:

INIT (Initialization) call
This call is made by the CICS transaction manager during the establishment of the bridge environment for the user transaction. The bridge facility is identified and a FACILITY_TOKEN created. If a null FACILITY_TOKEN is supplied, the bridge exit establishes a new bridge facility and creates a new token; if a valid FACILITY_TOKEN is supplied, the bridge exit uses the existing bridge facility, and the parameters USERID, STARTCODE and FACILITYLIKE are ignored.

This call also processes the BRDATA passed on the START command, storing the passed data in the Bridge Exit Area (BRXA) for subsequent use during the execution of the user transaction.

The following values can be set in the transaction and common area of the BRXA on this call. Any other values are ignored.

Table 7. Init call parameters

Field Name	Default
BRXA_FACILITY_TOKEN	0
BRXA_STARTCODE	Terminal
BRXA_FACILITYLIKE	blank
BRXA_ABEND_CODE	blank
BRXA_USER_ABEND_CODE	blank
BRXA_LOAD_ADS_DESCRIPTOR	NO
BRXA_IDENTIFIER	nulls
BRXA_FORMATTER	nulls
BRXA_CALL_EXIT_FOR_SYNCPOINT	YES

The INIT call can only issue CICS API commands that do not explicitly or implicity use resources that could be recoverable. Invalid commands are:

- File commands
- Temporary storage commands
- Transient data commands
- Task related user exit requests
- CREATE commands
- SYNCPOINT commands
- ENQ commands
- CICS Business Transaction Services (BTS) commands

BIND call

This call is made by the CICS transaction manager during task initalization, when the Unit of Work (UOW) is created.

It is used to open queues and possibly obtain the message to run the user transaction.

You can also use the EXEC CICS VERIFY command in this call to validate a password or pass-ticket in the message.

The following BRXA parameters can be set in the BIND call:

Table 8. Init call parameters

Field Name	Default
BRXA_ABEND_CODE	blank
BRXA_USER_ABEND_CODE	blank
BRXA_LOAD_ADS_DESCRIPTOR	NO
BRXA_IDENTIFIER	nulls

SYNCPOINT call

This is only called if the field BRXA_CALL_EXIT_FOR_SYNCPOINT is set.

BRXA_SYNC_COMMAND is set to indicate whether the SYNCPOINT is a rollback. This call allows the exit to write a request to the client before and/or after the syncpoint call. Note that if the call is a rollback, the write request should not be recoverable. The SYNCPOINT call must be reissued exactly the same as the original call.

TERM (termination) call

This call is made by the CICS transaction manager when the user transaction issues a RETURN command. On this call the bridge exit sends the response message back to the client application. This can be done by a direct interface to the transport mechanism, for example, by issuing an MQPUT command to an MQ response queue, or indirectly by passing the response message to the bridge monitor for transmission.

This call also identifies the next transaction to be run if this has been specified and can then issue an EXEC CICS START BREXIT command for the next TRANSID, or return the next transaction information to the client application.

The following values can be set in the transaction and common area of the BRXA on this call. Any other values are ignored.

- BRXA_FACILITY_KEEP_TIME
- BRXA_USER_ABEND_CODE

ABEND call

This call is made if the user transaction abends, so that the bridge exit can send non-recoverable messages to the client application. For example, a non-syncpointing MQPUT can be issued for the MQ bridge.

Recoverable requests cannot be made in this call.

BRXA_USER_ABEND_CODE can not be set in this call.

The following values can be set in the transaction and common areas on this call. Any other values are ignored.

- BRXA_FACILITY_KEEP_TIME

Message calls to the bridge exit

The following calls are made if a formatter is specified in the INIT call:

brxa-read-message-nowait

The purpose of this call is to read the next message if there already is one available. It is only used if the client application writes more than one message at a time. When a message is read, the bridge exit could check the password or pass-ticket in the message using an EXEC CICS VERIFY PASSWORD to ensure that the message came from an authorized source.

brxa-read-message-wait

The purpose of this call to to get the next message. This is usually done by sending a message to the client, requesting the next message, and waiting for a reply. When a message is read, the bridge exit could check the password or pass-ticket in the message using an EXEC CICS VERIFY PASSWORD to ensure that the message came from an authorized source.

brxa-write-message

The purpose of this call is to write a message. This is an intermediate message due to either a flush request, or the message buffer being full.

API calls to the bridge exit

These calls are only made if a formatter is not specified in the INIT call:

SEND call
This call is made to the bridge exit when the user transaction issues a SEND command. This is the reverse of what happens in a RECEIVE command. In this case the bridge exit copies data from the command area to a message to be returned to the client application.

RECEIVE call
This call is made to the bridge exit when the user transaction issues a RECEIVE command. Using the input obtained from BRDATA (or from a client application message), data is copied to the INTO or SET storage for the user transaction RECEIVE command. For RECEIVE MAP calls, this is the ADS. The bridge exit can set the following fields in the command area:

- The EIBRESP/EIBRESP2 values (defaults to NORMAL)
- The EIBAID value (defaults to ENTER)
- The EIBCPOSN value (defaults to 0)

CONVERSE
This call combines the function of SEND and RECEIVE.

FREE
After this call has been made, no further API calls should be made.

ISSUE DISCONNECT
After this call has been made, no further API calls should be made.

ISSUE ERASEAUP

RETRIEVE
RETRIEVE calls are normally only used in the first leg of a pseudoconversation, when a startcode of SD is returned. All other RETRIEVE requests should return ENDDATA.

Writing your own formatter

If the bridge exit specifies a formatter, the formatter is called for the following requests:

- SEND (Terminal Control and BMS)
- RECEIVE (Terminal Control and BMS)
- CONVERSE
- FREE
- ISSUE DISCONNECT
- ISSUE ERASEAUP
- RETRIEVE (in some cases)

Calls to the formatter

Your formatter must handle the following calls:

RECEIVE call
This call is made to the formatter when the user transaction issues a RECEIVE command. Using the input obtained from BRDATA (or from a client application message), data is copied to the INTO or SET storage for the user transaction

RECEIVE command. For RECEIVE MAP calls, this is the ADS. The bridge exit can set the following fields in the command area:

- The EIBRESP/EIBRESP2 values (defaults to NORMAL)
- The EIBAID value (defaults to ENTER)
- The EIBCPOSN value (defaults to 0)

If the RECEIVE does not have data to answer a receive request (for a conversational transaction), then if the client can send more than one message at a time, this call can return a response of BRXA-FMT-READ-MESSAGE-NOWAIT to CICS to ask if the next message has already arrived. CICS then calls the formatter again. If a message was available the formatter will be called again for the same request. This time there should be a information to process the command. If there is (still) nothing in the message to answer the receive request, this call can add a vector to the end of the message, requesting more data, and return a response of BRXA-FMT-REQUEST-NEXT-MESSAGE to CICS. The bridge exit will write the current message, and read the next message when it arrives. When the message is read, the formatter will be called again. This time there should be information to process the command.

SEND call
This call is made to the bridge exit when the user transaction issues a SEND command. This is the reverse of what happens in a RECEIVE command. In this case the bridge exit copies data from the command area to a message to be returned to the client application. If the message is too large to process, the send call returns a response of BRXA-FMT-OUTPUT-BUFFER-FULL to CICS. This results in the bridge exit being called to send the message and free the buffer space. The formatter will be called again for the same request. If the SEND results in the current message being flushed, this call should return a response of BRXA-FMT-WRITE-MESSAGE. The formatter will not be called again for the same request.

CONVERSE
This call combines the function of SEND and RECEIVE.

FREE
After this call has been made, no further API calls should be made.

ISSUE DISCONNECT
After this call has been made, no further API calls should be made.

ISSUE ERASEAUP

RETRIEVE
RETRIEVE calls are normally only used in the first leg of a pseudoconversation, when a startcode of SD is returned. All other RETRIEVE requests should return ENDDATA.

Return codes from the formatter

If the formatter cannot fully process a command, it passes a return code back to CICS in the field BRXA_FMT_RESPONSE. This gives the formatter the option of calling the bridge exit. CICS also passes state information to the formatter in fields BRXA_READ_NOWAIT_ISSUED and BRXA_REQUEST_NEXT_ISSUED.

BRXA_FMT_RESPONSE

The following output values can be returned to CICS in BRXA_FMT_RESPONSE

BRXA_FMT_NONE
(default) No action. The formatter has processed the request.

BRXA_FMT_OUTPUT_BUFFER_FULL
There is no room to add the next vector. Call the bridge exit to write the message, clear the buffer, then call the formatter again.

BRXA_FMT_WRITE_MESSAGE
The request required data to be flushed. Call the bridge exit to write the message.

BRXA_FMT_REQUEST_NEXT_MESSAGE
The formatter has processed all the data in the message. Call the bridge exit to read another message, then call the formatter again.

BRXA_FMT_READ_MESSAGE_NOWAIT
The formatter has processad all the data in the message. Check to see if there is a new message before requesting any further input. Call the bridge exit to read a message, then call the formatter again.

BRXA_READ_NOWAIT_ISSUED

The following values are passed to the formatter in BRXA_READ_NOWAIT_ISSUED. This field is used by the formatter to check if it has already returned a BRXA_FMT_READ_MESSAGE_NOWAIT for this command.

BRXA_NO
A BRXA_FMT_READ_MESSAGE_NOWAIT has not been returned for this command.

BRXA_YES
A BRXA_FMT_READ_MESSAGE_NOWAIT has been returned for this command.

BRXA_REQUEST_NEXT_ISSUED

The following values are passed to the formatter in BRXA_REQUEST_NEXT_ISSUED. This field is used by the formatter to check if it has already returned a BRXA_FMT_REQUEST_NEXT_MESSAGE for this command.

BRXA_NO
A BRXA_FMT_REQUEST_NEXT_MESSAGE has not been returned for this command.

BRXA_YES
A BRXA_FMT_REQUEST_NEXT_MESSAGE has been returned for this command.

Bridge exit and formatter programming considerations

Before you plan to write a 3270 bridge exit, you need to know about the following programming constraints:

Language support

A bridge exit can be written in Assembler, C, COBOL, or PL/I. Copybooks files to support these languages are listed in “Supplied copybooks” on page 105.

The bridge exit area parameters described in this book are shown with an underscore, ’_’, as a name separator. In COBOL programs, this underscore is replaced by a dash, ’-’.

TWA

The bridge exit and the user transaction access the same Transaction Work Area (TWA), so you should take care if making any changes to it. The TWA is created after the user transaction initialization call to the exit. Before that, it has zero length.

Storage

The bridge exit is a user replaceable program. This means that all local and SET storage will not be accessible after the exit call has completed. You should store any state data in storage obtained using GETMAIN.

Recoverable resources

All requests made in a bridge exit are run as part of the same unit of work as the user transaction. Therefore, any recoverable requests made by the bridge exit are committed or rolled back at the same time as the user transaction’s resources.

Resource sharing

The bridge exit should not attempt to share resources such as VSAM records or ENQ names with the user transaction.

Abend

The exit should not issue an EXEC CICS ABEND to simulate existing transaction abends because abend handling is modified in the user replaceable program environment. Abends occurring in the bridge exit are trapped by CICS and the user transaction is abnormally terminated with ABRQ. If the bridge exit needs to force an abend, it should set the field BRXA_USER_ABEND.

Bridge exit area (BRXA)

This section contains Product-sensitive Programming Interface and Associated Guidance Information.

The bridge exit area (BRXA) is the interface between the bridge exit or the formatter and CICS.

This is an interface defined by CICS, and must be used by all bridge exits and formatters. It is a CICS COMMAREA, accessed by ADDRESS COMMAREA, which is passed to the bridge exit or formatter by CICS whenever it is called.

The names of the parameters and constants, translated into appropriate forms for the different programming languages supported, are supplied in the copybook files listed in “Supplied copybooks” on page 105.

The BRXA contains a number of sub-areas that are used by the bridge exit and formatter to process each call, and retain information between calls. It consists of the following sub-areas:

Header
This area contains version information and pointers to some of the following areas.

Transaction area
This area is used by bridge exit initialization processing. It contains information about the user transaction that CICS will run, and the real 3270 that it expects to use.

Command area
This area provides details of the command request. For CICS API requests it provides a simplified description of the command and response fields.

User area
This area is used to store data between calls to the bridge exit. It acts as a user input area to store the messages needed to satisfy RECEIVE and RETRIEVE requests, and also as a user output area to store the messages from SEND requests so that they can all be sent together when the user transaction terminates.

ADS descriptor
This area contains an ADS descriptor for BMS SEND MAP and RECEIVE MAP requests, if the mapset has been assembled with CICS Transaction Server for OS/390 Release 2 or later release and an ADS descriptor has been created.

The ADS descriptor is created by the BMS macros in either a ***long*** or a ***short*** form. Long data has the same content as short data, but the fields are word aligned to support those transport mechanisms, such as MQSeries, that require all message fields to be word aligned.

BRXA header area

Offset Hex	Type	Len	Name
(0)	STRUCTURE	56	BRXA_HEADER
(0)	CHARACTER	8	BRXA_HEADER_EYECATCHER
(8)	FULLWORD	4	BRXA_HEADER_LENGTH
(C)	FULLWORD	4	BRXA_HEADER_VERSION_NO
(10)	ADDRESS	4	BRXA_TRANSACTION_AREA_PTR
(14)	FULLWORD	4	BRXA_TRANSACTION_AREA_LEN
(18)	ADDRESS	4	BRXA_COMMAND_AREA_PTR
(1C)	FULLWORD	4	BRXA_COMMAND_AREA_LEN
(20)	ADDRESS	4	BRXA_USER_AREA_PTR
(24)	FULLWORD	4	BRXA_USER_AREA_LEN
(28)	ADDRESS	4	BRXA_INPUT_MSG_PTR
(2C)	FULLWORD	4	BRXA_INPUT_MSG_LEN
(30)	ADDRESS	4	BRXA_OUTPUT_MSG_PTR
(34)	FULLWORD	4	BRXA_OUTPUT_MSG_LEN

The BRXA header contains the following fields:

BRXA_HEADER_EYECATCHER
An eye-catcher to identify the area as an BRXA. This is initialized by CICS to the value BRXA_HEADER_EYE ('>BRAREA '), which is defined in the DFHBRACx copy books.

BRXA_HEADER_LENGTH
The length of the header.

BRXA_HEADER_VERSION_NO
The version number of the BRXA. This allows future releases to extend the BRXA. This is initialized by CICS to the value of BRXA_CURRENT_VERSION_NO in the DFHBRACx copybook.

BRXA_TRANSACTION_AREA_PTR
The address of the transaction subarea, BRXA_TRANSACTION_AREA. This is set by CICS, and should not be modified by the bridge exit code.

BRXA_TRANSACTION_AREA_LEN
The length of the transaction subarea, BRXA_TRANSACTION_AREA. This is set by CICS, and should not be modified by the bridge exit code.

BRXA_COMMAND_AREA_PTR
The address of the command subarea, BRXA_COMMAND_AREA, This is set by CICS, and should not be modified by the bridge exit code.

BRXA_COMMAND_AREA_LEN
The length of the command subarea, BRXA_COMMAND_AREA. This is set by CICS, and should not be modified by the bridge exit code.

BRXA_USER_AREA_PTR
A field that allows the address of a user area to be saved across bridge exit calls within a task. The user area should be obtained using an EXEC CICS GETMAIN.

BRXA_USER_AREA_LEN
A field in which the exit can save the length of the user area.

BRXA_INPUT_MSG_PTR
A field used to save the address of an input message. This field is intended to be used in conjunction with a formatter.

BRXA_INPUT_MSG_LEN
A field used to save the current length of the input message.

BRXA_OUTPUT_MSG_PTR
A field used to save the address of an output message. This field is intended to be used in conjunction with a formatter.

BRXA_OUTPUT_MSG_LEN
A field used to save the current length of the output message.

BRXA transaction area

Offset	Type	Len	Name
Hex			
(0)	STRUCTURE	180	BRXA_TRANSACTION_AREA
(0)	CHARACTER	8	BRXA_TRAN_AREA_EYECATCHER
(8)	CHARACTER	4	BRXA_BRIDGE_TRANID
(C)	CHARACTER	4	BRXA_TRANID
(10)	CHARACTER	4	BRXA_NEXTTRANID
(14)	CHARACTER	4	BRXA_ABEND_CODE
(18)	CHARACTER	8	BRXA_CALLING_PROG

Offset Hex	Type	Len	Name
(20)	CHARACTER	8	BRXA_USERID
(28)	CHARACTER	8	reserved
(30)	CHARACTER	2	BRXA_STARTCODE
(32)	CHARACTER	1	BRXA_LOAD_ADS_DESCRIPTOR
(33)	CHARACTER	1	BRXA_TRACE
(34)	CHARACTER	4	BRXA_FACILITYLIKE
(38)	UNSIGNED	4	BRXA_FACILITY_KEEP_TIME
(3C)	CHARACTER	8	BRXA_FACILITY_TOKEN
(44)	HALFWORD	2	BRXA_SCREEN_HEIGHT
(46)	HALFWORD	2	BRXA_SCREEN_WIDTH
(48)	HALFWORD	2	BRXA_ALTERNATE_SCREEN_HEIGHT
(4A)	HALFWORD	2	BRXA_ALTERNATE_SCREEN_WIDTH
(4C)	CHARACTER	48	BRXA_IDENTIFIER
(7C)	CHARACTER	8	BRXA_FORMATTER
(84)	CHARACTER	1	BRXA_CALL_EXIT_FOR_SYNC
(85)	CHARACTER	1	BRXA_NEXTTRANID_SOURCE
(86)	CHARACTER	6	reserved
(8C)	CHARACTER	8	reserved
(94)	FULLWORD	4	BRXA_BRDATA_PTR
(98)	FULLWORD	4	BRXA_BRDATA_LEN
(9C)	CHARACTER	4	BRXA_INTERVAL
(A0)	CHARACTER	4	BRXA_TIME
(A4)	FULLWORD	4	BRXA_HOURS
(A8)	FULLWORD	4	BRXA_MINUTES
(AC)	FULLWORD	4	BRXA_SECONDS
(B0)	CHARACTER	1	BRXA_START_AFTER
(B1)	CHARACTER	1	BRXA_START_AT
(B2)	CHARACTER	2	reserved

The transaction area contains the following fields:

BRXA_TRAN_AREA_EYECATCHER
An eye-catcher to identify the area as a BRXA transaction area. This is set by CICS, before passing control to the bridge exit, to the value BRXA_TRAN_AREA_EYE ('>BRTRANA'), defined in the DFHBRACx copy books.

BRXA_BRIDGE_TRANID
The transaction identifier of the bridge monitor transaction that issued a START TRANSID BREXIT command to start this bridge exit and its associated user transaction.

BRXA_TRANID
The transaction identifier of the user transaction.

BRXA_NEXTTRANID
The transaction identifier of the next transaction. This is set by CICS from the TRANSID value provided in the final RETURN command of the user transaction; from the value provided by a SET TERMINAL NEXTTRANSID command, or from the TRANSID of the first START issued by the user transaction for the bridge facility. This field contains blanks (X'40') if no next transaction has been specified.

BRXA_ABEND_CODE
The abend code if the bridge transaction abends before initializing the user transaction, or if the user transaction abends. If the transaction has not abended, this field is blanks.

BRXA_CALLING_PROG
The name of the program in the user transaction which issued the command causing the bridge exit to be invoked. For the initialization, termination, and abend calls, this field is set to blanks.

BRXA_USERID
The USERID under which the user transaction is running.

BRXA_STARTCODE
This field is set to the start code appropriate to the next transaction returned in BRXA_NEXTTRANID. The following start codes are possible:

brxa_start
START command without data.

brxa_startdata
START command with data.

brxa_terminput
Terminal input (default).

The initial value of this field is blanks.

BRXA_LOAD_ADS_DESCRIPTOR
A 1-character field that tells CICS whether or not to provide the ADS descriptor on subsequent SEND MAP and RECEIVE MAP commands.

If this field is set to 'Y' (BRXA_YES) when the user transaction issues a SEND MAP and RECEIVE MAP command, CICS loads the mapset, locates the ADS descriptor for the map, and provides its address in BRXA_ADS_DESCRIPTOR_PTR in the command subarea.

The ADS descriptor format is explained in "ADS descriptor area" on page 100.

If this field has any value other than 'Y', then CICS does not attempt to load the mapset and locate the descriptor, and BRXA_ADS_DESCRIPTOR_PTR is set to null.

BRXA_TRACE
A 1-character field that is set to 'Y' (BRXA_YES) if level-2 tracing is set on for the bridge. The bridge can use this flag to trace input and output data, for example, for diagnostic purposes.

Note that for BR level tracing, the BRXA is already traced by CICS on input and output.

BRXA_FACILITYLIKE
The name of an installed 3270 terminal to be used as a template terminal definition for the bridge facility.

The exit sets this value during the initialization call. If the exit does not provide a value, CICS looks for a value specified as FACILITYLIKE in the user transaction's profile. If this value is also blanks, CICS uses the CICS-supplied definition CBRF (based on model DFHLU2).

If the specified FACILITYLIKE does not exist, CICS abends the transaction ABRJ.

It is not possible to change the FACILITYLIKE definition after the terminal has been created, so this parameter is ignored if BRXA_FACILITY_TOKEN is specified.

If the real FACILITYLIKE terminal is logged on when the bridge facility is created, any values returned by QUERY will apply also to the bridge facility.

BRXA_FACILITY_KEEP_TIME

The time (in seconds) that the bridge facility is kept after the user transaction terminates. If a non-zero value is set in this field the bridge facility and its pseudo-conversational data are retained.

CICS sets this value to zero before the bridge exit initialization call. The exit can set it at any time; CICS does not use the value until the exit returns from the task termination call. If the value is zero, CICS discards the bridge facility; if non-zero, CICS retains the facility and associated data.

The maximum value is one week (604800 seconds). If a value larger than this is specified, CICS retains the bridge facility for one week.

BRXA_FACILITY_TOKEN

A token representing the bridge facility to be used. CICS initializes this value to nulls but the exit can set it in the initialization call.

Specifying a value implies reusing a bridge facility kept from a previous transaction.

The default value of nulls results in CICS dynamically allocating a new bridge facility.

The name of the bridge facility is accessible to the user transaction in the EIBTRMID field of the EIB. No other TERMIDs in the system are the same, although the name may be reused almost immediately when the user transaction finishes, if BRXA_FACILITY_KEEP_TIME is set to zero.

BRXA_SCREEN_HEIGHT

The current screen height.

BRXA_SCREEN_WIDTH

The current screen width.

BRXA_ALTERNATE_SCREEN_HEIGHT

The alternate screen height.

BRXA_ALTERNATE_SCREEN_WIDTH

The alternate screen width.

BRXA_IDENTIFIER

A 48-character field provided by the bridge exit. The intended use of this field is for task-specific information to assist in on-line problem resolution. It could contain, for example, the MQ correlator for the MQ bridge, or a Web token.

BRXA_FORMATTER

An 8- byte character field to be used by the bridge exit to specify the name of a user replaceable program to be used as a formatter. If a program name is specified in this field, then the it is called for all BMS, terminal, and interval control requests. The bridge exit is only called for XM, SYNC and MSG requests.

BRXA_CALL_EXIT_FOR_SYNC
A 1-character field that tells CICS whether or not to call the bridge exit for SYNCPOINT requests.

If this field is set to ’Y’ (BRXA_YES), then the bridge exit is called; if it is set to ’N’ (BRXA_NO), then the bridge exit is not called.

BRXA_NEXTTRANID_SOURCE
A 1-character field that indicates how the next transaction was specified. This indicator can have three settings:

BRXA_IMMEDIATE
The next TRANSID value came from a RETURN IMMEDIATE command.

BRXA_STARTED
The next TRANSID value came from a START TERMID command.

BRXA_NORMAL
The next TRANSID value came from a RETURN TRANSID or SET TERMINAL/NETNAME command.

BRXA_BRDATA_PTR
A fullword (4-byte) field that contains the address of the data specified by the BRDATA parameter on the START TRANSID BREXIT command.

BRXA_BRDATA_LEN
A fullword (4-byte) field that contains the length of the data specified by the BRDATA parameter on the START TRANSID BREXIT command.

BRXA_INTERVAL
A 4-character field containing the INTERVAL value specified by the user transaction on a START for its bridge facility.

BRXA_TIME
A 4-character field containing the TIME value specified by the user transaction on a START for its bridge facility.

BRXA_HOURS
A fullword (4-byte) field containing the HOURS value specified by the user transaction on a START for its bridge facility.

BRXA_MINUTES
A fullword (4-byte) field containing the MINUTES value specified by the user transaction on a START for its bridge facility.

BRXA_SECONDS
A fullword (4-byte) field containing the SECONDS value specified by the user transaction on a START for its bridge facility.

BRXA_START_AFTER
A 1-character field containing the AFTER value specified by the user transaction on a START for its bridge facility.

BRXA_START_AT
A 1-character field containing the AT value specified by the user transaction on a START for its bridge facility.

BRXA command area

The command area contains information relating to the command that has caused the bridge exit to be called. Common fields appear in the first **common** section of the command area, and fields specific to a particular command, or group of commands, follow.

The following diagrams show the various fields in the BRXA_COMMAND_AREA, and the commands for which they are valid.

An ’I’ in the table indicates that the field is an input parameter to the exit; it is set by CICS before passing control to the exit, and any changes to the value made by the exit are ignored by CICS.

An ’O’ in the table indicates that the field is an output parameter from the exit; the exit must set this value before return (unless the default value is acceptable), because CICS uses the value in completing the command.

A ’-’ or a blank in the table indicates that the field is not applicable to that command. The values on entry to the exit are undefined, and CICS ignores any value set in the field by the exit.

Table 9. BRXA command area field usage

Field name	**SEND**	**RECEIVE**	**CONVERSE**	**ISSUE ERASEAUP**	**ISSUE DISCONNECT**	**FREE**	**SEND MAP**	**SEND CONTROL**	**SEND TEXT**	**RECEIVE MAP**	**RECV PARTN**	**RETRIEVE**	**Syncpoint**
brxa_command_common													
brxa_function_code	I	I	I	I	I	I	I	I	I	I	I	I	I
brxa_command_code	I	I	I	I	I	I	I	I	I	I	I	I	I
brxa_user_abend_code	O	O	O	O	O	O	O	O	O	O	O	O	O
brxa_from_ptr	I	-	I	-	-	-	I	-	I	-	-	-	-
brxa_from_len	I	-	I	-	-	-	I	-	I	-	-	-	-
brxa_into_ptr	-	O	O	-	-	-	-	-	-	O	O	O	-
brxa_into_len	-	O	O	-	-	-	-	-	-	O	O	O	-
brxa_resp	O	O	O	O	O	O	O	O	O	O	O	O	O
brxa_resp2	O	O	O	O	O	O	O	O	O	O	O	O	O
brxa_cposn	-	O	O	-	-	-	-	-	-	O	O	-	-
brxa_aid	-	O	O	-	-	-	-	-	-	O	O	-	-
brxa_erase_indicator	I	-	I	-	-	-	I	I	I	-	-	-	-
brxa_last_indicator	I	-	I	-	-	-	I	I	I	-	-	-	-
brxa_wait_indicator	I	-	I	I	-	-	I	I	I	-	-	-	-
brxa_tc_command													
brxa_ctlchar	I	-	I	-	-	-							
brxa_buffer_indicator	-	I	-	-	-	-							

Table 9. BRXA command area field usage (continued)

Field name	SEND	RECEIVE	CONVERSE	ISSUE ERASEAUP	ISSUE DISCONNECT	FREE	SEND MAP	SEND CONTROL	SEND TEXT	RECEIVE MAP	RECV PARTN	RETRIEVE	Syncpoint
brxa_strfield_indicator	I	-	I	-	-.	-							
brxa_defresp_indicator	I	-	I	-	-	-							
brxa_invite_indicator	I	-	-	-	-	-							
brxa_bms_command													
brxa_mapset							I	-	-	I	-		
brxa_map							I	-	-	I	-		
brxa_ads_descriptor_ptr							I	-	-	I	-		
brxa_cursor							I	I	I	-	-		
brxa_msr_data							I	I	I	-	-		
brxa_data_indicator							I	-	-	-	-		
brxa_eraseaup_indicator							I	I	-	-	-		
brxa_freekb_indicator							I	I	I	-	-		
brxa_alarm_indicator							I	I	I	-	-		
brxa_msr_indicator							I	I	I	-	-		
brxa_frset_indicator							I	I	-	-	-		
brxa_text_type							-	-	I	-	-		
brxa_ic_command													
brxa_rtermid												O	
brxa_rtransid												O	
brxa_queue												O	
brxa_sync_command													
brxa_explicit													I
brxa_rollback													I

BRXA command area - common

Offset Hex	Type	Len	Name
(0)	STRUCTURE	48	BRXA_COMMAND_COMMON
(0)	CHARACTER	8	BRXA_COMMAND_AREA_EYECATCHER
(8)	CHARACTER	2	BRXA_FUNCTION_CODE
(A)	CHARACTER	2	BRXA_COMMAND_CODE
(C)	CHARACTER	4	BRXA_USER_ABEND_CODE
(10)	ADDRESS	4	BRXA_FROM_PTR
(14)	FULLWORD	4	BRXA_FROM_LEN
(18)	ADDRESS	4	BRXA_INTO_PTR
(1C)	FULLWORD	4	BRXA_INTO_LEN
(20)	FULLWORD	2	BRXA_RESP

Offset Hex	Type	Len	Name
(22)	HALFWORD	2	BRXA_RESP2
(24)	HALFWORD	2	BRXA_CPOSN
(26)	CHARACTER	1	BRXA_AID
(27)	CHARACTER	1	BRXA_ERASE_INDICATOR
(28)	CHARACTER	1	BRXA_LAST_INDICATOR
(29)	CHARACTER	1	BRXA_WAIT_INDICATOR
(2A)	CHARACTER	1	BRXA_FMT_RESPONSE
(2B)	CHARACTER	1	BRXA_READ_NOWAIT_ISSUED
(2C)	CHARACTER	1	BRXA_REQUEST_NEXT_ISSUED
(2D)	CHARACTER	3	(reserved)

BRXA_COMMAND_AREA_EYECATCHER

An eye-catcher to identify the area as a bridge command area. This is set by CICS, before passing control to the bridge exit, to the value BRXA_COMMAND_AREA_EYE ('>BRCOMMA'), which is defined in the DFHBRACx copy book.

BRXA_FUNCTION_CODE

A 2-character code identifying the CICS function for which the bridge exit was called. For calls to initialize a transaction, terminate a transaction and abend a transaction, this is '00'. For all other requests, this is the 2-digit value in the first byte of EIBFN converted to character form. Valid EBCDIC characters are used for the function and command code to simplify testing of the values in user transaction exit programs written in all the supported languages, and to simplify passing of the codes to other systems. A constant with a meaningful name is provided for all the supported languages to simplify testing. The value is:

```
BRXA_XM
BRXA_TC
BRXA_IC
BRXA_SYNC
BRXA_BMS
BRXA_MSG
```

BRXA_COMMAND_CODE

A two-character code identifying the CICS command for which the bridge exit was called. For transaction initialization this is '02', for transaction termination this is '04' and, for transaction abend this is '06'. For all other requests, this is the value in the second byte of EIBFN converted to character form.

See the *CICS Application Programming Reference* for information about EIBFN values. Valid EBCDIC characters are used for the function and command code to simplify testing of the values in user transaction exit programs written in all the supported languages, and to simplify passing of the codes to other systems. Constants with meaningful names are provided for all the supported languages to simplify testing. The values are:

```
BRXA_INIT
BRXA_TERM
BRXA_ABEND
*   tc
BRXA_RECEIVE
BRXA_SEND
BRXA_CONVERSE
BRXA_ISSUE_DISCONNECT
BRXA_ISSUE_ERASEAUP
BRXA_FREE
*   bms
BRXA_RECEIVE_MAP
BRXA_SEND_MAP
```

```
BRXA_SEND_TEXT
BRXA_SEND_CONTROL
*   ic
BRXA_RETRIEVE
*   sync
BRXA_SYNCPOINT
*   msg
BRXA_READ_MESSAGE_NOWAIT
BRXA_READ_MESSAGE_WAIT
BRXA_WRITE_MESSAGE
```

BRXA_USER_ABEND_CODE

The abend code. CICS initializes this value to blanks. If the exit changes it to any other value, CICS generates a transaction abend with this code.

BRXA_FROM_PTR

The address of the FROM data in SEND, CONVERSE, SEND MAP, SEND TEXT, and START commands. This is zero for other commands, or if FROM is not specified on the command.

BRXA_FROM_LEN

The length of the FROM data in SEND, CONVERSE, SEND MAP, SEND TEXT, and START commands. This is zero for other commands, or if FROM is not specified on the command.

BRXA_INTO_PTR

The address of the INTO data in RECEIVE, CONVERSE, RECEIVE MAP and RETRIEVE commands. This must be set by the bridge exit, and CICS copies data from this address into the INTO area specified on the command, or copies the address into the SET parameter specified on the command.

Note: The exit must GETMAIN storage for INTO input because local storage could be reused on return from the bridge exit.

BRXA_INTO_LEN

The length of the INTO data in RECEIVE, CONVERSE, RECEIVE MAP, and RETRIEVE commands. This must be set by the user transaction exit, and CICS copies this value into the LENGTH, FLENGTH, or INTOLENGTH parameter specified on the command, and uses the value when copying data into the INTO area.

Note: CONVERSE is the only command which has both FROM and INTO, and the BRXA_FROM_PTR and BRXA_INTO_PTR (and corresponding lengths) could be replaced by a single BRXA_DATA_PTR (and BRXA_DATA_LEN). For CONVERSE, the exit replaces the FROM address and length by the INTO address and length.

BRXA_RESP

The resp code to be set (by CICS) in EIBRESP. This will be set to zero by CICS before calling the exit, and the exit must set this value if anything other than a normal response is required. CICS will generate an ABRN transaction abend if the value returned is not one that could normally be produced by CICS for this command.

If this value is zero on return, CICS may itself set the EIBRESP value and raise a condition.

BRXA_RESP2

The RESP2 code CICS returns and stores in EIBRESP2. This is set to zero by CICS before calling the exit, and the exit must change this value if anything other than a normal response is required.

CICS does not check the value specified for consistency with the command. If this value is zero on return, CICS may itself set the EIBRESP2 value and raise a condition.

BRXA_CPOSN
The cursor position to be set (by CICS) in EIBCPOSN for RECEIVE, CONVERSE, and RECEIVE MAP commands. This is set to zero by CICS before calling the exit, and the exit must change this value if the user transaction uses the value in EIBCPOSN.

BRXA_AID
The attention identifier (PF key code) to be set (by CICS) in EIBAID for RECEIVE, CONVERSE, and RECEIVE MAP commands. This is set to ENTER (X'7D') by CICS before calling the exit, and the exit must change this value if the user transaction expects another value in EIBAID. The exit can use the values defined in DFHAID copy books to set the value (these are EBCDIC values of the 3270 AID characters).

BRXA_ERASE_INDICATOR
A 1-character value which is set (by CICS) to indicate whether ERASE, ERASE ALTERNATE, or ERASE DEFAULT is specified on SEND, CONVERSE SEND MAP, SEND TEXT, or SEND CONTROL commands. Constants with meaningful names are provided for all languages to allow the bridge exit to test this value if necessary. The values are:

```
BRXA_ERASE
BRXA_ERASE_ALTERNATE
BRXA_ERASE_DEFAULT
```

BRXA_LAST_INDICATOR
A 1-character field indicating whether LAST is specified on a SEND command. Valid values are 'Y' or 'N'; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_WAIT_INDICATOR
A one-character field indicating whether WAIT is specified on a SEND command, or on a RETRIEVE command. Valid values are 'Y' or 'N'; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_FMT_RESPONSE
This field is used by the formatter to tell the CICS that the bridge exit should be called to read or write a message. Possible values are:

BRXA_FMT_NONE
No action. The formatter has processed the request.

BRXA_FMT_OUTPUT_BUFFER_FULL
There is no room to add the next vector. Call the bridge exit to write the message, clear the buffer, then call the formatter again.

BRXA_FMT_WRITE_MESSAGE
The request required data to be flushed. Call the bridge exit to write the message.

BRXA_FMT_READ_MESSAGE_NOWAIT
The formatter has processed the data in the message. Check to see if there is a new message before requesting any further input. Call the bridge exit to read a message, then call the formatter again.

BRXA_FMT_REQUEST_NEXT_MESSAGE
The formatter has processed the data in the message. Call the bridge exit to read a message, then call the formatter again.

BRXA_READ_NOWAIT_ISSUED
This field is used by the formatter to check if it has already returned a BRXA_FMT_READ_MESSAGE_NOWAIT for this command. Possible values are:

BRXA_NO
BRXA_FMT_READ_MESSAGE_NOWAIT has not been returned for this command.

BRXA_YES
BRXA_FMT_READ_MESSAGE_NOWAIT has been returned for this command.

BRXA_REQUEST_NEXT_ISSUED
This field is used by the formatter to check if it has already returned a BRXA_FMT_REQUEST_NEXT_MESSAGE for this command. Possible values are:

BRXA_NO
BRXA_FMT_REQUEST_NEXT_MESSAGE has not been returned for this command.

BRXA_YES
BRXA_FMT_REQUEST_NEXT_MESSAGE has been returned for this command.

BRXA command area - terminal control

The terminal control command area defines some terminal control specific parameters.

Commands supported are SEND, RECEIVE, and CONVERSE.

Offset Hex	**Type**	**Len**	**Name**
(0)	STRUCTURE	49	BRXA_TC_COMMAND
(0)	common	44	
(2C)	CHARACTER	1	BRXA_CTLCHAR
(2D)	CHARACTER	1	BRXA_BUFFER_INDICATOR
(2E)	CHARACTER	1	BRXA_STRFIELD_INDICATOR
(2F)	CHARACTER	1	BRXA_DEFRESP_INDICATOR
(30)	CHARACTER	1	BRXA_INVITE_INDICATOR

BRXA_CTLCHAR
The 3270 Write Control Character (WCC) passed on SEND and CONVERSE commands as CTLCHAR. If not specified on the command, the default value (X’C3’- unlock keyboard, reset MDT flags) is passed to the exit.

BRXA_BUFFER_INDICATOR
A 1-character field indicating whether BUFFER was specified on a RECEIVE command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_STRFIELD_INDICATOR

A 1-character field indicating whether STRFIELD was specified on a SEND or CONVERSE command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_DEFRESP_INDICATOR

A 1-character field indicating whether DEFRESP was specified on a SEND or CONVERSE command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_INVITE_INDICATOR

A 1-character field indicating whether INVITE was specified on a SEND command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA command area - BMS

The BMS command interface defines some BMS specific parameters.

Commands supported are SEND MAP, SEND TEXT, SEND CONTROL, and RECEIVE MAP.

Offset Hex	Type	Len	Name
(0)	STRUCTURE	77	BRXA_BMS_COMMAND
(0)	common	44	
(2C)	CHARACTER	7	BRXA_MAPSET
(33)	CHARACTER	1	(reserved)
(34)	CHARACTER	7	BRXA_MAP
(3B)	CHARACTER	1	(reserved)
(3C)	ADDRESS	4	BRXA_ADS_DESCRIPTOR_PTR
(40)	HALFWORD	2	BRXA_CURSOR
(42)	CHARACTER	4	BRXA_MSR_DATA
(46)	CHARACTER	1	BRXA_DATA_INDICATOR
(47)	CHARACTER	1	BRXA_ERASEAUP_INDICATOR
(48)	CHARACTER	1	BRXA_FREEKB_INDICATOR
(49)	CHARACTER	1	BRXA_ALARM_INDICATOR
(4A)	CHARACTER	1	BRXA_FRSET_INDICATOR
(4B)	CHARACTER	1	BRXA_MSR_INDICATOR
(4C)	CHARACTER	1	BRXA_TEXT_TYPE

BRXA_MAPSET

The (unsuffixed) mapset name specified on SEND MAP or RECEIVE MAP.

BRXA_MAP

The map name specified on SEND MAP or RECEIVE MAP.

BRXA_ADS_DESCRIPTOR_PTR

The address of the ADS descriptor for BMS SEND MAP and RECEIVE MAP commands. This is set by CICS, if the bridge exit has set the flag indicating that the descriptor should be loaded, and if the relevant mapset has been reassembled under CICS Transaction Server for OS/390 Release 3 to include the descriptor. Otherwise this pointer is set to 0.

BRXA_CURSOR

A halfword value containing the CURSOR position specified on SEND MAP, SEND TEXT, or SEND CONTROL command, which identifies where the cursor is to be positioned on the 3270 screen. A value of -1 is passed if the application specified CURSOR with no value on SEND MAP command, indicating that symbolic cursor positioning is required, that is, that the cursor is to be positioned in the first field in the application data structure that has a value of -1 in the corresponding length field. A value of -2 is passed if the application did not specify CURSOR on the SEND MAP command.

BRXA_MSR_DATA

The 4-character value specified in MSR on a SEND MAP, SEND CONTROL, or SEND TEXT command. Constants are provided in the copy book DFHMSRCA that allow the exit to test the values specified.

Note: If you assume that a BFB will always be constructed as if its TYPETERM were defined with MSRCONTROL(NO), then this parameter could be omitted, because BMS ignores the MSR field specified on the command for a 3270 terminal for which MSRCONTROL(NO) is specified.

BRXA_DATA_INDICATOR

A 1-character field indicating whether DATAONLY, MAPONLY, or neither is specified on the SEND MAP command. Valid values are ’D’ (DATAONLY), ’M’ (MAPONLY) or ’N’ (neither specified); the following constants are provided for the exit to test this field:

```
BRXA_DATAONLY
BRXA_MAPONLY
BRXA_NEITHER
```

(Note that if MAPONLY is specified, the FROM pointer and length are zero, because there is no application data structure in this case.)

BRXA_ERASEAUP_INDICATOR

A 1-character field indicating whether ERASEAUP is specified on a SEND MAP or SEND CONTROL command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_FREEKB_INDICATOR

A 1-character field indicating whether FREEKB is specified on a SEND MAP, SEND TEXT, or SEND CONTROL command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_ALARM_INDICATOR

A 1-character field indicating whether ALARM is specified on a SEND MAP, SEND TEXT, or SEND CONTROL command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_MSR_INDICATOR

A 1-character field indicating whether MSR is specified on a SEND MAP, SEND TEXT, or SEND CONTROL command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_FRSET_INDICATOR

A 1-character field indicating whether FRSET is specified on a SEND MAP or SEND CONTROL command. Valid values are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

BRXA_TEXT_TYPE

A 1-character field indicating whether NOEDIT or MAPPED is specified on a SEND TEXT command. Valid values are blank (neither NOEDIT nor MAPPED specified), ’N’ (NOEDIT specified) and ’M’ (MAPPED specified); the following constants are provided for the exit to test this field:

```
BRXA_TEXT_NORMAL
BRXA_TEXT_MAPPED
BRXA_TEXT_NOEDIT
```

BRXA command area - interval control

The interval control command area defines some interval control specific parameters.

The only command supported is RETRIEVE.

Offset Hex	Type	Len	Name
(0)	STRUCTURE	60	BRXA_IC_COMMAND
(0)	common	44	
(2C)	CHARACTER	4	BRXA_RTERMID
(30)	CHARACTER	4	BRXA_RTRANSID
(34)	CHARACTER	8	BRXA_QUEUE

BRXA_RTERMID

The value of RTERMID specified on a START command. For the RETRIEVE command, this is a field that the bridge exit can set to pass the RTERMID value back to the application issuing the RETRIEVE.

BRXA_RTRANSID

The value of RTRANSID specified on a START command. For the RETRIEVE command, this is a field that the bridge exit can set to pass the RTRANSID value back to the application issuing the RETRIEVE.

BRXA_QUEUE

The value of QUEUE specified on START command. For the RETRIEVE command this is a field in which the bridge exit can set the QUEUE value to be used by the application issuing the RETRIEVE,

BRXA command area - syncpoint

The syncpoint command area defines actions at SYNCPOINT and SYNCPOINT ROLLBACK. BRXA_EXPLICIT is used to indicate that this request originated from an explicit EXEC CICS SYNCPOINT command, or that it is an implicit syncpoint generated by CICS. It is set to ’Y’ or ’N’ before the exit is invoked; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

Valid values for BRXA_ROLLBACK are ’Y’ or ’N’; the following constants are provided for the exit to test this field:

```
BRXA_YES
BRXA_NO
```

Offset Hex	Type	Len	Name
(0)	STRUCTURE	46	BRXA_SYNC_COMMAND
(0)	common	44	
(2C)	CHARACTER	1	BRXA_EXPLICIT
(2D)	CHARACTER	1	BRXA_ROLLBACK

BRXA command area - MSG

This command area defines actions when the bridge exit is called to read or write a message. These functions are only used if the bridge exit specified a formatter on initialization.

This command area defines actions at initialization. Relevant fields are in the common part of the command area. The layout of the MSG section of the command area is :

Offset Hex	Type	Len	Name
(0)	STRUCTURE	48	BRXA_MSG_COMMAND

ADS descriptor area

The ADS descriptor allows interpretation of the BMS Application Data Structure (the symbolic map used by your application program for the data in SEND and RECEIVE MAP requests) - without requiring your program to include the relevant DSECT or copybook at compile time.

The ADS descriptor contains a header with general information about the map, and a field descriptor for every field that appears in the ADS, corresponding to every named field in the map definition macro. It can be located in the mapset from an offset field in DFHMAPDS.

The ADS descriptor is available only if the map load module has been reassembled (using CICS Transaction Server for OS/390 Release 2 or a later release) to include the descriptor, and CICS attempts to locate the descriptor only if the BRXA_LOAD_ADS_DESCRIPTOR indicator is set to BRXA_YES in the bridge exit initialization call.

| The ADS descriptor is created by the BMS macros in either a ***long*** or a ***short*** form. Long data has the same content as short data, but the fields are aligned on 4-byte boundaries to support those transport mechanisms, such as MQSeries, that require all message fields to be word aligned. The long form is supported only in the C language. The ADS descriptor is defined below in both short and long format.

ADS descriptor header

| The ADS descriptor header contains general information about the map and a pointer to the first of a variable number of chained field descriptions.

Short form:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	38	ADS_DESCRIPTOR
(0)	HALFWORD	2	ADSD_LENGTH
(2)	CHARACTER	4	ADSD_EYECATCHER
(6)	HALFWORD	2	ADSD_MAP_INDEX
(8)	HALFWORD	2	ADSD_FIELD_COUNT
(A)	HALFWORD	2	ADSD_STRUCTURE_LENGTH
(C)	HALFWORD	2	ADSD_ATTRIBUTE_NUMBER
(E)	CHARACTER	12	ADSD_ATTRIBUTE_TYPE_CODES
(1A)	CHARACTER	1	ADSD_MAP_JUSTIFY_HOR
(1B)	CHARACTER	1	ADSD_MAP_JUSTIFY_VER
(1C)	HALFWORD	2	ADSD_MAP_STARTING_LINE
(1E)	HALFWORD	2	ADSD_MAP_STARTING_COLUMN
(20)	HALFWORD	2	ADSD_MAP_LINES
(22)	HALFWORD	2	ADSD_MAP_COLUMNS
(24)	CHARACTER	1	ADSD_WRITE_CONTROL_CHARACTER
(25)	CHARACTER	1	(reserved)
(26)	STRUCTURE	*	ADSD_FIRST_FIELD

ADSD_LENGTH
The length of the ADS descriptor.

ADSD_EYECATCHER
An eye-catcher (’ADSD’) to identify this as an ADS descriptor.

ADSD_MAP_INDEX
The index number of the map within the mapset. This is needed to determine the HTML template corresponding to the map.

ADSD_FIELD_COUNT
The number of fields within the ADS; that is, the number of named fields in the map definition. A separate field is counted for each element of an array defined with the OCCURS parameter, but subfields of group fields (GRPNAME) are not counted. The field count may be zero, in which case there are no field descriptors following the header.

ADSD_STRUCTURE_LENGTH
The length of the application data structure.

ADSD_ATTRIBUTE_NUMBER
The number of extended attributes in each field of the ADS; that is, the number of attributes specified in DSATTS in the map definition.

ADSD_ATTRIBUTE_TYPE_CODES
a 1-character code for the attribute types in each field, in order, derived from DSATTS:

C = COLOR

P = PS

H = HILIGHT

V = VALIDN

O = OUTLINE

S = SOSI

T = TRANSP

ADSD_MAP_JUSTIFY_HOR
The horizontal justification for the map, either L (LEFT) or R (RIGHT) from the JUSTIFY operand on the map definition.

ADSD_MAP_JUSTIFY_VER
The vertical justification for the map, from the JUSTIFY operand on the map definition. This can have the values F (FIRST), L (LAST), B (BOTTOM), or blank (no vertical JUSTIFY operand).

ADSD_MAP_STARTING_LINE
The starting line for the map, from the LINE operand on the DFHMDI macro, (LINE = NEXT gives a value of 255; LINE = SAME gives a value of 254.)

ADSD_MAP_STARTING_COLUMN
The starting column for the map, from the COLUMN operand on the DFHMDI macro. (COLUMN = NEXT gives a value of 255; COLUMN = SAME gives a value of 254.)

ADSD_MAP_LINES
The number of lines in the map from the ’SIZE=’ operand.

ADSD_MAP_COLUMNS
The number of columns in the map from the ’SIZE=’ operand.

ADSD_WRITE_CONTROL_CHAR
The 3270 encoded WCC derived from the ’CONTROL=’ operand.

ADSD_FIRST_FIELD
The first field descriptor. The address of ADSD_FIRST_FIELD can be used as the initial value of the pointer for the field descriptor (unless ADSD_FIELD_COUNT is 0).

Long form:

Offset	Type	Len	Name
Hex			
(0)	STRUCTURE	60	ADS_LONG_DESCRIPTOR
(0)	FULLWORD	42	ADSDL_LENGTH
(4)	CHARACTER	4	ADSDL_EYECATCHER
(8)	FULLWORD	4	ADSDL_MAP_INDEX
(C)	FULLWORD	4	ADSDL_FIELD_COUNT
(10)	FULLWORD	4	ADSDL_STRUCTURE_LENGTH
(14)	FULLWORD	4	ADSDL_ATTRIBUTE_NUMBER
(18)	CHARACTER	12	ADSDL_ATTRIBUTE_TYPE_CODES
(24)	CHARACTER	1	ADSDL_MAP_JUSTIFY_HOR
(25)	CHARACTER	1	ADSD_MAP_JUSTIFY_VER
(26)	CHARACTER	2	reserved
(28)	FULLWORD	4	ADSDL_MAP_STARTING_LINE
(2C)	FULLWORD	4	ADSDL_MAP_STARTING_COLUMN
(30)	FULLWORD	4	ADSDL_MAP_LINES
(34)	FULLWORD	4	ADSDL_MAP_COLUMNS
(38)	CHARACTER	1	ADSDL_WRITE_CONTROL_CHARACTER
(39)	CHARACTER	3	(reserved)
(3C)	STRUCTURE	*	ADSDL_FIRST_FIELD

ADSDL_LENGTH
The length of the ADS descriptor.

ADSDL_EYECATCHER
An eye-catcher (’ADSL’) to identify this as an ADS descriptor.

ADSDL_MAP_INDEX
The index number of the map within the mapset. This is needed to determine the HTML template corresponding to the map.

ADSDL_FIELD_COUNT
The number of fields within the ADS; that is, the number of named fields in the map definition. A separate field is counted for each element of an array defined with the OCCURS parameter, but subfields of group fields (GRPNAME) are not counted. The field count may be zero, in which case there are no field descriptors following the header.

ADSDL_STRUCTURE_LENGTH
The length of the short application data structure.

ADSDL_ATTRIBUTE_NUMBER
The number of extended attributes in each field of the ADS; that is, the number of attributes specified in DSATTS in the map definition.

ADSDL_ATTRIBUTE_TYPE_CODES
a 1-character code for the attribute types in each field, in order, derived from DSATTS:

C = COLOR

P = PS

H = HILIGHT

V = VALIDN

O = OUTLINE

S = SOSI

T = TRANSP

ADSDL_MAP_JUSTIFY_HOR
The horizontal justification for the map, either L (LEFT) or R (RIGHT) from the JUSTIFY operand on the map definition.

ADSDL_MAP_JUSTIFY_VER
The vertical justification for the map, from the JUSTIFY operand on the map definition. This can have the values F (FIRST), L (LAST), B (BOTTOM), or blank (no vertical JUSTIFY operand).

ADSDL_MAP_STARTING_LINE
The starting line for the map, from the LINE operand on the DFHMDI macro, (LINE = NEXT gives a value of 255; LINE = SAME gives a value of 254.)

ADSDL_MAP_STARTING_COLUMN
The starting column for the map, from the COLUMN operand on the DFHMDI macro. (COLUMN = NEXT gives a value of 255; COLUMN = SAME gives a value of 254.)

ADSDL_MAP_LINES
The number of lines in the map from the ’SIZE=’ operand.

ADSDL_MAP_COLUMNS
The number of columns in the map from the ’SIZE=’ operand.

ADSDL_WRITE_CONTROL_CHAR
The 3270 encoded WCC derived from the ’CONTROL=’ operand.

ADSDL_FIRST_FIELD
The first field descriptor. The address of ADSD_FIRST_FIELD can be used as the initial value of the pointer for the field descriptor (unless ADSD_FIELD_COUNT is 0).

ADS field descriptor

After the header, the ADS descriptor contains a variable number of field descriptors.

Short form:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	42	ADS_FIELD_DESCRIPTOR
(0)	CHARACTER	32	ADSD_FIELD_NAME
(20)	HALFWORD	2	ADSD_FIELD_NAME_LEN
(22)	HALFWORD	2	ADSD_OCCURS_INDEX
(24)	HALFWORD	2	ADSD_FIELD_OFFSET
(26)	HALFWORD	2	ADSD_FIELD_DATA_LEN
(28)	CHARACTER	1	ADSD_FIELD_JUSTIFY
(29)	CHARACTER	1	ADSD_FIELD_FILL_CHAR
(2A)	CHARACTER	*	ADSD_NEXT_FIELD

ADSD_FIELD_NAME
The unsuffixed field name padded with blanks.

ADSD_FIELD_NAME_LEN
The number of characters in the field name.

ADSD_OCCURS_INDEX
When OCCURS is specified for a field definition there is a separate field descriptor for each element of the array, and ADSD_OCCURS_INDEX indicates the array index for the particular field. If OCCURS is not specified, then ADSD_OCCURS_INDEX is 0.

ADSD_FIELD_OFFSET
The offset of the field within the ADSDL. The offset is to the beginning of the (fullword) length field, and you must add 4 (for the length field) + 4 (for the 3270 attribute) + 8 fullwords for the extended attributes to obtain the offset of the data part of the field.

ADSD_FIELD_DATA_LEN
The length of the field in the ADS.

ADSD_FIELD_JUSTIFY
A 1-character field Indicating whether the data is to be justified left 'L' or right 'R' if the supplied length is less than the length in the ADS.

ADSD_FIELD_FILL_CHAR
The character (blank or '0') to be used to pad the remainder of the field in the ADS.

ADSD_NEXT_FIELD
The next field descriptor. The address of ADSD_NEXT_FIELD can be used to update a pointer for the field descriptor.

Long form:

Offset Hex	Type	Len	Name
(0)	STRUCTURE	52	ADS_LONG_FIELD_DESCRIPTOR
(0)	CHARACTER	32	ADSDL_FIELD_NAME
(20)	FULLWORD	4	ADSDL_FIELD_NAME_LEN
(24)	FULLWORD	4	ADSDL_OCCURS_INDEX
(28)	FULLWORD	4	ADSDL_FIELD_OFFSET
(2C)	FULLFWORD	4	ADSDL_FIELD_DATA_LEN

Offset Hex	Type	Len	Name
(30)	CHARACTER	1	ADSDL_FIELD_JUSTIFY
(31)	CHARACTER	1	ADSDL_FIELD_FILL_CHAR
(32)	CHARACTER	2	reserved
(34)	CHARACTER	*	ADSD_NEXT_FIELD

ADSD_LONG_FIELD_NAME
The unsuffixed field name padded with blanks.

ADSDL_FIELD_NAME_LEN
The number of characters in the field name.

ADSDL_OCCURS_INDEX
When OCCURS is specified for a field definition there is a separate field descriptor for each element of the array, and ADSD_OCCURS_INDEX indicates the array index for the particular field. If OCCURS is not specified, then ADSD_OCCURS_INDEX is 0.

ADSDL_FIELD_OFFSET
The offset of the field within the ADS. The offset is to the beginning of the (halfword) length field, and you must add 2 (for the length field) + 1 (for the 3270 attribute) + attribute_number (for the extended attributes specified in DSATTS) to obtain the offset of the data part of the field.

ADSDL_FIELD_DATA_LEN
The length of the field in the ADS.

ADSDL_FIELD_JUSTIFY
A 1-character field Indicating whether the data is to be justified left ’L’ or right ’R’ if the supplied length is less than the length in the ADS.

ADSDL_FIELD_FILL_CHAR
The character (blank or ’0’) to be used to pad the remainder of the field in the ADS.

ADSDL_NEXT_FIELD
The next field descriptor. The address of ADSD_NEXT_FIELD can be used to update a pointer for the field descriptor.

Supplied copybooks

The names of the parameters and constants, translated into appropriate forms for the different programming languages supported, are defined in files supplied as part of the 3270 bridge. The files or copybooks for the various languages are listed in the following table.

Table 10. BRXA copybooks

Language	Area definition	Area constants
Assembler	DFHBRARD	DFHBRACD
C	DFHBRARH	DFHBRACH
PL/I	DFHBRARL	DFHBRACL
COBOL	DFHBRARO	DFHBRACO

Copybook example (DFHBRACD)

The following constants are provided for use by an assembler language bridge exit to set and interpret values in the BRXA.

API function codes are character equivalent constants of the first byte of the EIBFN values documented in the *CICS Application Programming Reference* manual.

```
* function codes
BRXA_XM   EQU   C'00'
BRXA_TC   EQU   C'04'
BRXA_IC   EQU   C'10'
BRXA_SYNC EQU   C'16'
BRXA_BMS  EQU   C'18'
BRXA_MSG  EQU   C'01'
```

```
* indicator values
BRXA_YES             EQU   C'Y'
BRXA_NO              EQU   C'N'
BRXA_ERASE           EQU   C'E'
BRXA_ERASE_ALTERNATE EQU   C'A'
BRXA_ERASE_DEFAULT   EQU   C'D'
BRXA_DATAONLY        EQU   C'D'
BRXA_MAPONLY         EQU   C'M'
BRXA_NEITHER         EQU   C'N'
BRXA_TEXT_NORMAL     EQU   C' '
BRXA_TEXT_MAPPED     EQU   C'M'
BRXA_TEXT_NOEDIT     EQU   C'N'
BRXA_IMMEDIATE       EQU   C'I'
BRXA_STARTED         EQU   C'S'
BRXA_NORMAL          EQU   C'B'
* Start codes
BRXA_START           EQU   C'S '
BRXA_STARTDATA       EQU   C'SD'
BRXA_TERMINPUT       EQU   C'TD'
* formatter response values
BRXA_FMT_NONE                  EQU   C'N'
BRXA_FMT_OUTPUT_BUFFER_FULL    EQU   C'F'
BRXA_FMT_WRITE_MESSAGE         EQU   C'W'
BRXA_FMT_REQUEST_NEXT_MESSAGE  EQU   C'Q'
BRXA_FMT_READ_MESSAGE_NOWAIT   EQU   C'R'
```

API command codes are character equivalent constants of the second byte of the EIBFN values documented in the *CICS Application Programming Reference* manual.

```
*   xm
BRXA_INIT             EQU  C'02'
BRXA_TERM             EQU  C'04'
BRXA_ABEND            EQU  C'06'
*   tc
BRXA_RECEIVE          EQU  C'02'
BRXA_SEND             EQU  C'04'
BRXA_CONVERSE         EQU  C'06'
BRXA_ISSUE_DISCONNECT EQU  C'14'
BRXA_ISSUE_ERASEAUP   EQU  C'18'
BRXA_FREE             EQU  C'22'
*   bms
BRXA_RECEIVE_MAP      EQU  C'02'
BRXA_SEND_MAP         EQU  C'04'
BRXA_SEND_TEXT        EQU  C'06'
BRXA_SEND_CONTROL     EQU  C'12'
*   ic
BRXA_RETRIEVE         EQU  C'0A'
*   sync
BRXA_SYNCPOINT        EQU  C'02'
*   msg (new for CTS 1.3)
BRXA_READ_MESSAGE_NOWAIT   EQU C'02'
BRXA_READ_MESSAGE_WAIT     EQU C'04'
BRXA_WRITE_MESSAGE         EQU C'06'
```

Chapter 7. Problem determination

This chapter contains Diagnosis, Modification, or Tuning Information.

This chapter helps you debug problems in the CICS 3270 bridge exit user-replaceable program, the IBM-supplied parts of the CICS 3270 bridge, and in the system setup of the CICS 3270 bridge. If you suspect that you have a problem in another part of CICS, refer to the *CICS Problem Determination Guide*.

The 3270 user program should be tested with a real 3270 terminal before transferring to a bridge environment.

Diagnostic information is designed to provide first failure data capture, so that if an error occurs, enough information about the error is available directly without having to reproduce the error situation. The information is presented in the following forms:

Messages
The CICS 3270 bridge provides CICS messages. These messages are listed in *CICS Messages and Codes*.

Trace The CICS 3270 bridge produces system trace entries containing all the important information required for problem diagnosis.

Dump Dump formatting is provided for data areas relating to the CICS 3270 bridge.

Abend codes
Transaction abend codes are standard 4-character names. The abend codes produced by the CICS 3270 bridge are listed in *CICS Messages and Codes*.

Troubleshooting

This section provides some hints on troubleshooting. It follows the general outline:

1. Define the problem.
2. Obtain information (documentation) on the problem.

Defining the problem

When you have a problem, first try to define the circumstances that gave rise to it. If you need to report the problem to the IBM software support center, this information is useful to the support personnel.

1. What is the system configuration?
   - CICS Transaction Server release
   - Release of any other products providing transport mechanisms for the 3270 bridge, such as MQSeries.
   - LE/370 release
   - OS/390 release
2. When did the problem first occur?
3. What were you trying to accomplish at the time the problem occurred?
4. What changes were made to the system before the occurrence of the problem?
   - To the OS/390 system
   - To the bridge exit

© Copyright IBM Corp. 1994, 1998

  - To the CICS user program being accessed by the bridge
  - To the end-user program
  - To the transport mechanisms
  - To CICS Transaction Server
5. What is the problem?
  - Incorrect output
  - Hang/Wait: Use CEMT INQUIRE to display details of the transaction.
  - Loop: Use CEMT INQUIRE to display the details of the transaction.
  - Abend in the bridge exit
  - Abend in a CICS program
  - Abend in the IBM-supplied part of the CICS 3270 bridge
  - Performance problem
  - Storage violation
  - Logic Error
6. At what point in the processing did the problem occur?

Documentation about the problem

To investigate most problems, you must look at the dumps, traces, and logs provided with MVS and CICS.

- System Dump: This contains the CICS internal trace
- CICS auxiliary trace, if enabled
- TCP/IP for MVS trace, if relevant
- GTF trace, if enabled
- Console log
- CSMT log
- CICS job log

To identify which are likely to be useful for your problem, try to work out the area of the CICS 3270 bridge giving rise to the problem.

Using messages and codes

CICS 3270 bridge messages have identifiers of the form DFHBR*nnnn*, where *nnnn* are four numeric characters. These numbers indicate which component generated the message, as shown in *CICS Messages and Codes*.

When the CICS 3270 bridge issues a message as a result of an error, it also makes an exception trace entry. The CICS 3270 bridge also generates information messages, for instance during enable processing and disable processing.

CICS 3270 bridge messages are supplied in English, Kanji, and Chinese. The CICS message editing utility can be used to translate them into other languages supported by CICS.

Using Trace

The CICS 3270 bridge creates CICS system trace, which is formatted using software supplied as part of CICS.

You can request level 2 tracing using SET TRACETYPE or the CETR supplied transaction. This gives a full trace of data being transmitted between the user transaction and the end-user application.

You should request level 2 tracing for the bridge by specifying BR in the SET TRACETYPE or CETR command. CICS sets BRXA_TRACE to ’Y’ if level 2 tracing is requested, but the bridge exit should create exception trace entries even if this flag is not set.

CICS trace output is described in the *CICS Problem Determination Guide*, and details of the contents of each trace points are given in the *CICS User’s Handbook*.

Dump and trace formatting

To control dump formatting of CICS 3270 bridge data areas, you change the CICS VERBEXIT in the IPCS control statement for dump formatting as follows:

```
IPCS VERBEXIT CICS520 BR=0|1|2|3, TR=1|2
```

The parameters have these meanings:

BR=0 Suppress system dump for the 3270 bridge.

BR=1 Produce system dump summary listing for the 3270 bridge.

BR=2 Produce system dump for the 3270 bridge.

BR=3 Produce system dump summary listing and a system dump for the 3270 bridge.

TR=1 Produce an abbreviated trace.

TR=2 Produce a full trace.

Full details of these and other parameters are described in the *CICS Operations and Utilities Guide*.

CICS 3270 bridge output in the formatted dump consists of the major control blocks of the CICS 3270 bridge, with interpretation of some of the fields. The CICS 3270 bridge output can be found in the IPCS output by searching for `==BR`. It is under the heading `BRIDGE FACILITY SUMMARY`.

Debugging the bridge exit

The following aids are provided to help you resolve problems occurring in the bridge exit while bridging to a 3270 user transaction:

IDENTIFIER

The bridge exit constructs a 48-byte identifier field, containing information to aid problem determination. This can contain relevant fields taken from the START data. You can access the identifier with INQUIRE TASK, or CEMT INQUIRE TASK.

EDF

The CEDX transaction provides EDF for non-terminal tasks. This allows you to use EDF with the bridge exit (which is a non terminal task). You should issue CEDX against the bridge transaction to see the initialization call to the bridge exit, otherwise you should issue CEDX against the user transaction.

Note that bridge facilities are not EDF-able.

The supplied bridge exit program, DFH0CBRE, is defined with EDF disabled; you will need to modify this is you intend to use CEDX.

See the *CICS Supplied Transactions* for more information about the use of CEDX.

Trace

Trace records are written during execution of the bridge exit. See the*CICS User's Handbook* for a description of the trace entries.

Level 2 trace will show you the messages transmitted, so you can verify that the user transaction and end-user application are cooperating correctly.

You can also use the ENTER TRACENUM command in the bridge exit to write user records to the CICS trace.

Debugging the supplied bridge exit

The supplied bridge exit, DFH0CBRE, is very well commented. You are recommended to read the source carefully before use. The following aids are provided to help you resolve problems:

Abend codes and Trace

The supplied exit creates trace and exception trace entries during execution. It also returns some abend codes (ABXx). The bridge mechanism validates correct use of BRXA, and ABRx abends result from incorrect usage. These codes and the trace points are all fully documented in the comments within DFH0CBRE.

Note: If you change the supplied exit, you can change the ABR prefix.

You need to activate the level-2 tracing with SET TRACETYPE or CETR, specifying the BR component.

Message validation

If CICS detects an invalid character in an input message, the MQCIH-ERROROFFSET field is set to the offset of the invalid character, counted from the start of the message.

Part 3. External CICS Interface

This part of the book describes the external CICS interface.

This part contains:

© Copyright IBM Corp. 1994, 1998

Chapter 8. Introduction to the external CICS interface

This chapter gives a brief overview of the external CICS interface (EXCI), covering the following topics:

- "Overview"
- "Resource recovery" on page 117
- "EXCI" on page 12
- "Requirements for the external CICS interface" on page 121

Overview

The external CICS interface is an application programming interface that enables a non-CICS program (a client program) running in MVS to call a program (a server program) running in a CICS region and to pass and receive data by means of a communications area. The CICS application program is invoked as if linked-to by another CICS application program.

This programming interface allows a user to allocate and open sessions (or pipes[1]) to a CICS region, and to pass distributed program link (DPL) requests over them. The multiregion operation (MRO) facility of CICS interregion communication (IRC) facility supports these requests, and each pipe maps onto one MRO session, with a limit of 100 pipes per EXCI address space.

The client program and the CICS server region (the region where the server program runs or is defined) must be in the same MVS image unless:

- The CICS region is running in a sysplex that supports cross-system MRO (XCF/MRO).
- All DPL requests issued by the client program specify the SYNCONRETURN option.

Although the external CICS interface does not support the cross-memory access method, it can use the XCF access method provided by the CICS XCF/MRO facility. See the *CICS Intercommunication Guide* for information about XCF/MRO.

A client program that uses the external CICS interface can operate multiple sessions for different users (either under the same or separate TCBs) all coexisting in the same MVS address space without knowledge of, or interference from, each other.

Where a client program attaches another client program, the attached program runs under its own TCB.

The programming interfaces

The external CICS interface provides two forms of programming interface: the EXCI CALL interface and the EXEC CICS interface.

[1] pipe. A one-way communication path between a sending process and a receiving process. In an external CICS interface implementation, each pipe maps onto one MRO session, where the client program represents the sending process and the CICS server region represents the receiving process.

© Copyright IBM Corp. 1994, 1998

The EXCI CALL interface: This interface consists of six commands that allow you to:

- Allocate and open sessions to a CICS system from non-CICS programs running under MVS
- Issue DPL requests on these sessions from the non-CICS programs
- Close and deallocate the sessions on completion of the DPL requests.

The six EXCI commands are:

- Initialize-User
- Allocate_Pipe
- Open_Pipe
- DPL call
- Close_Pipe
- Deallocate_Pipe

The EXEC CICS interface: The external CICS interface provides a single, composite command—EXEC CICS LINK PROGRAM—that performs all six commands of the EXCI CALL interface in one invocation.

| This command is similar but not identical to the distributed program link command of the CICS command-level application programming interface.

API restrictions for server programs

A CICS server program invoked by an external CICS interface request is restricted to the DPL subset of the CICS application programming interface. This subset (the DPL subset) of the API commands is the same as for a CICS-to-CICS server program.

See the *CICS Application Programming Guide* for details of the DPL subset for server programs.

Choosing between the EXEC CICS and the CALL interface

As illustrated in the various language versions of the CICS-supplied sample client program (see “Sample application programs” on page 175 for details), you can use both the CALL interface (all six commands) and the EXEC CICS LINK command in the same program, to perform separate requests. As a general rule, it is unlikely that you would want to do this in a production program.

Each form of the external CICS interface has its particular benefits.

- For low-frequency or single DPL requests, you are recommended to use the EXEC CICS LINK command.

  It is easier to code, and therefore less prone to programming errors.

  Note that each invocation of an EXEC CICS LINK command causes the external CICS interface to perform all the functions of the CALL interface, which results in unnecessary overhead.
- For multiple or frequent DPL requests from the same client program, you are recommended to use the EXCI CALL interface.

  This is more efficient, because you need only perform the Initialize_User and Allocate_Pipe commands once, at or near the beginning of your program, and the Deallocate_Pipe once on completion of all DPL activity. In between these functions, you can open and close the pipe as necessary, and while the pipe is opened, you can issue as many DPL calls as you want.

Illustrations of the external CICS CALL interface

The diagrams in Figure 25 through Figure 28 on page 116 illustrate the external CICS interface using the EXCI CALL interface.

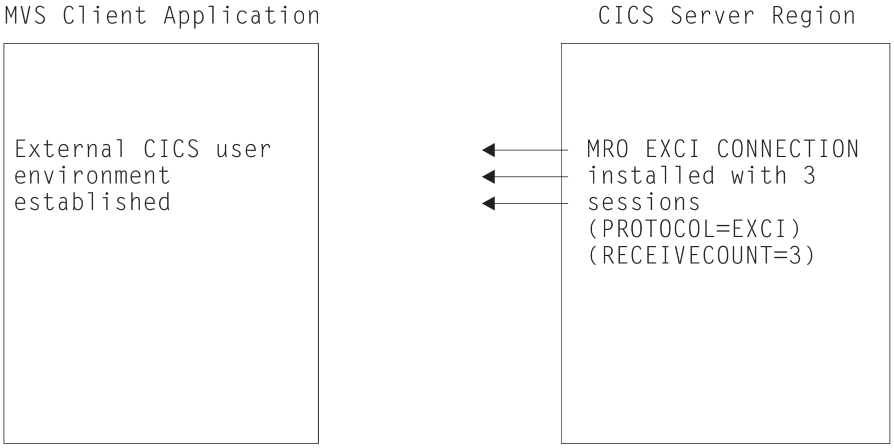

MVS Client Application
CICS Server Region
External CICS user environment established
MRO EXCI CONNECTION installed with 3 sessions (PROTOCOL=EXCI) (RECEIVECOUNT=3)

Figure 25. Stage 1: Status after an INITIALIZE_USER call

Notes:

1. In Figure 25, the target CICS region is running with IRC open, and one EXCI connection with three sessions installed, at the time the client application program issues an INITIALIZE_USER call.
2. The client application program address space is initialized with the EXCI user environment. There is no MRO activity at this stage, and no pipe exists.

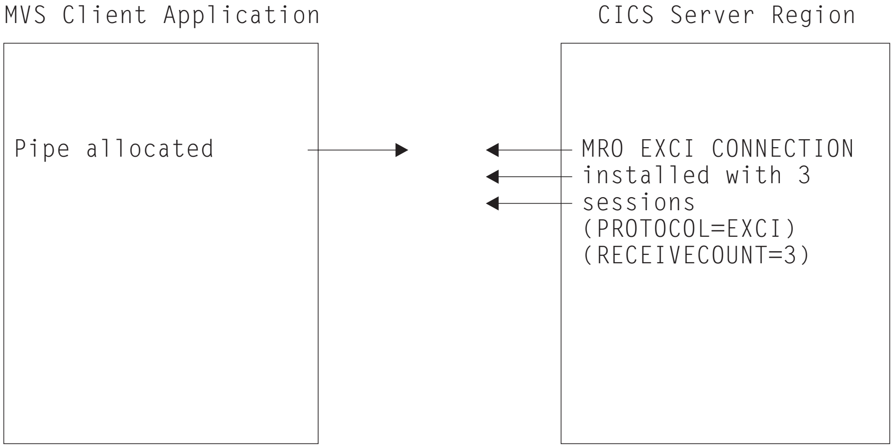

MVS Client Application
CICS Server Region
Pipe allocated
MRO EXCI CONNECTION installed with 3 sessions (PROTOCOL=EXCI) (RECEIVECOUNT=3)

Figure 26. Stage 2: Status after the first ALLOCATE_PIPE call

Note: In Figure 26, the external CICS interface logs on to MRO, identifying the target CICS server region.

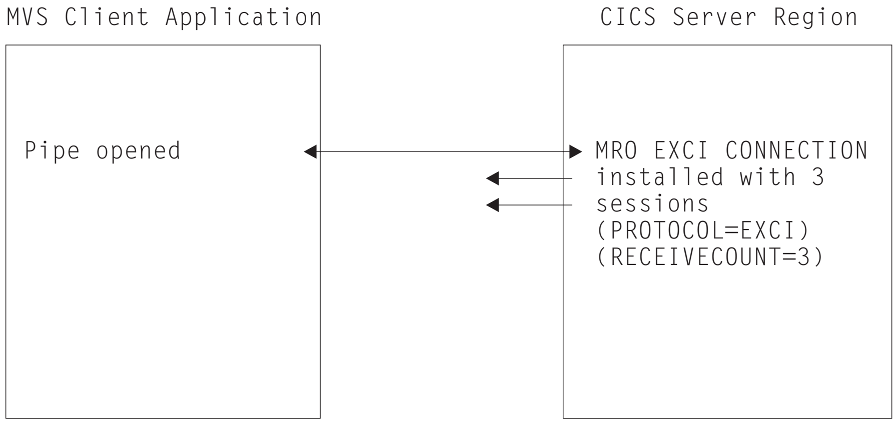


Figure 27. Stage 3: Status after the OPEN_PIPE call

Notes:

1. In Figure 27, the external CICS interface connects to the CICS server region, and the pipe is now available for use.
2. The remaining two EXCI sessions are free, and can be used by further open pipe requests from the same, or a different, client application program (provided the connection is generic).

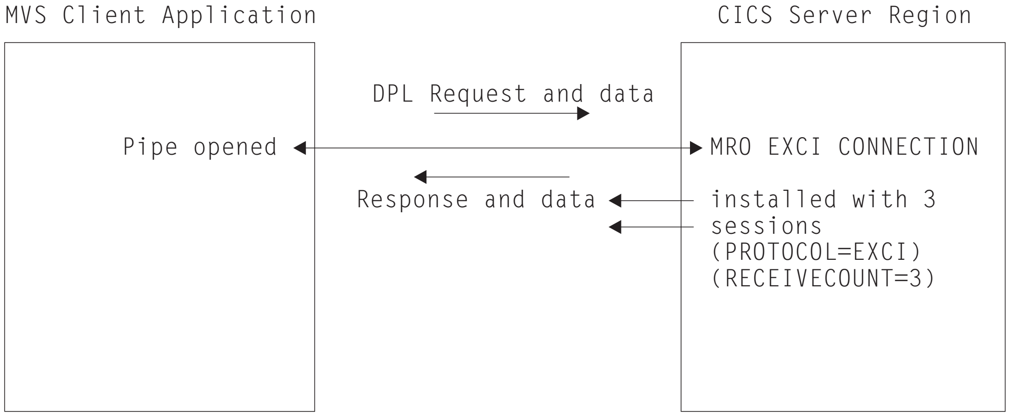


Figure 28. Stage 4: Status with one open pipe, processing a DPL call

Note: In Figure 28, the external CICS interface passes the DPL request over the open pipe, with any associated data. The CICS server region returns a response and data over the open pipe.

Closing pipes: When the client application program closes a pipe, it remains allocated ready for use by the same user, and the status is as shown in Figure 26 on page 115. At this stage, the MRO session is available for use by another open pipe request, from the same or from a different client application program (provided the connection is generic).

Deallocating pipes: When the client application program deallocates a pipe, it logs off from MRO and frees all the storage associated with the session. This leaves the status as shown in Figure 25 on page 115.

Illustration of the EXCI EXEC CICS interface

Figure 29 illustrates the EXEC CICS interface, and how it resolves to the six EXCI CALLs.

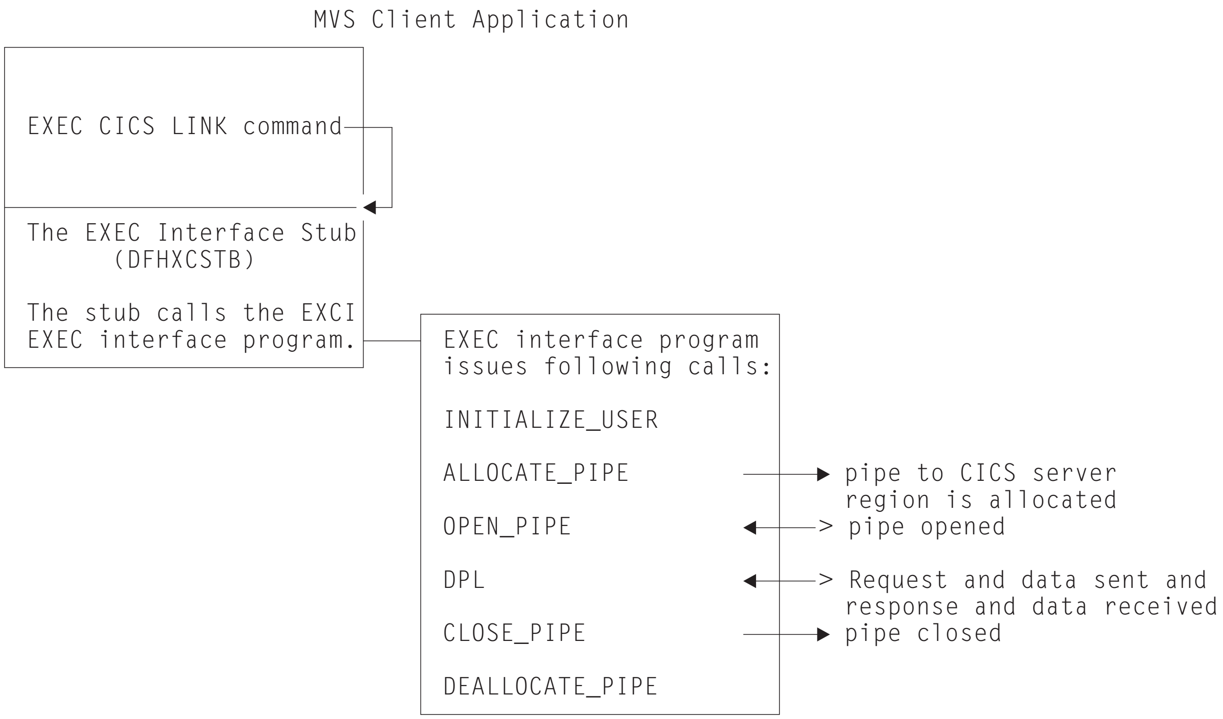


Figure 29. Illustration of the external CICS interface using the EXEC CICS command

Resource recovery

Resource recovery consists of the protocols and program interfaces that allow an application program to make consistent changes to multiple protected resources. The external CICS interface supports resource recovery.

A CICS server program that is invoked by an external CICS interface request can update recoverable resources; the changes are committed when the mirror transaction in the CICS server region takes a syncpoint. The client program can determine when syncpointing should occur. There are two options:

- Resource recovery controlled by the CICS server regions. In this case, changes to recoverable resources are committed at the completion of each DPL request, independently of the client program. Also, in addition to the syncpoint taken when the server program returns control to CICS (the SYNCONRETURN option), the server program can take explicit syncpoints during execution.
- Resource recovery controlled by the EXCI client program with the support of recoverable resource management services (RRMS). When the client program requests it, updates made by the server program in successive DPL requests are committed together.

  To support this option, CICS and the external CICS interface both make use of resource recovery services (RRS), the OS/390 syncpoint manager[2],which is an

[2] RRMS comprises three OS/390 components: registration services, context services, and resource recovery services (RRS)

MVS component of recoverable resource management services (RRMS). In the context of RRMS, CICS is a **resource manager**; the client program may issue requests to other resource managers, and have resources owned by those resource managers committed in the same **unit-of-recovery (UR).**[3]

These options are controlled as follows:

- By the DPL_opts parameter of the DPL_request.
- By the SYNCONRETURN option, either specified or omitted, on the EXEC CICS LINK PROGRAM command.

If you specify SYNCONRETURN, a syncpoint is taken on completion of each DPL request. If SYNCONRETURN is omitted, a syncpoint is taken when the client program requests it using the interfaces described in “Taking a syncpoint in the client program” on page 121.

Using RRMS with the external CICS interface

To use RRMS to coordinate DPL requests, ensure that:

- The EXCI client and the CICS region to which it sends DPL requests are running in the same MVS image (this is an RRMS restriction, and does not apply to DPL requests that specify SYNCONRETURN).
- The CICS region that receives the DPL requests is started with RRMS=YES specified as a system initialization parameter (the default is RRMS=NO).
- RRS is running in the MVS image where CICS and the client program execute. See *OS/390 MVS Programming: Resource Recovery*.

For an illustration of the concept of the external CICS interface and CICS using RRMS, see Figure 30 on page 119.

[3] A unit of recovery is analogous to a CICS unit of work

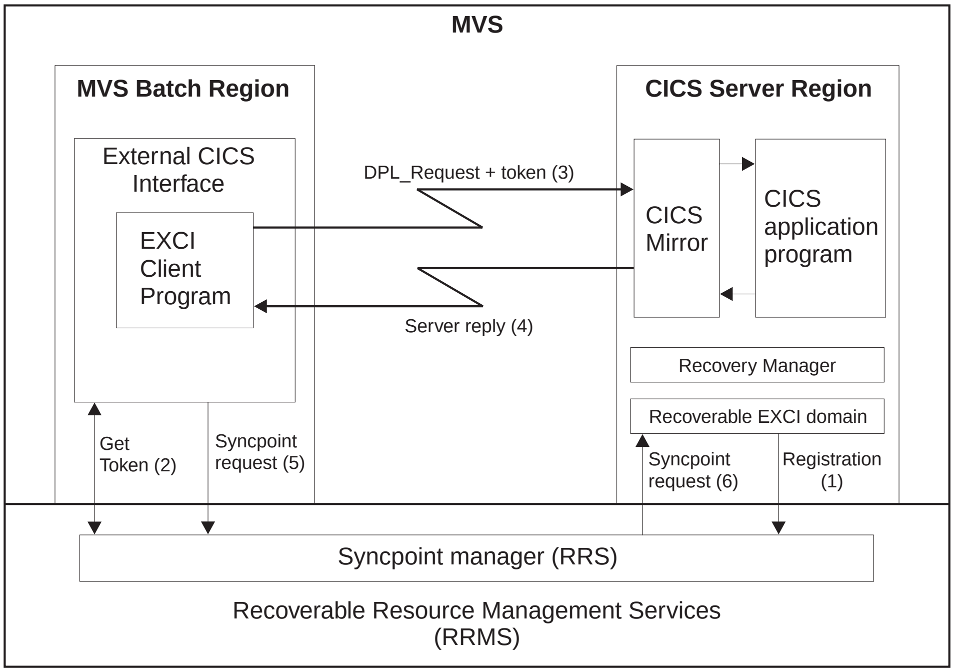


Figure 30. Conceptual view of EXCI client and CICS server region using RRMS. The main (numbered) steps within a unit-of-recovery are described in the list following the diagram

1. If the CICS system initialization parameter RRMS=YES is specified, CICS registers with RRMS as a resource manager. This registration occurs during CICS initialization.
2. When the EXCI client program issues a DPL_Request call in 2-phase commit mode (a call that omits the SYNCONRETURN option), it receives from RRMS:
   - A unit-of-recovery identifier (URID)
   - A context token
   - A pass token
3. The URID and the tokens obtained by EXCI on behalf of the client program are included on the DPL request passed to the CICS server region. If the DPL request is the first one within the UR, CICS calls RRS to express an interest in the UR, attaches a new mirror transaction, and validates the tokens. If the request is valid, the mirror program links to the specified server application program. The server program completes its work, which is all performed within the unit-of-recovery. This work can include updating recoverable resources in the local server region, or daisy-chaining to other CICS regions.
4. When the server program completes, it returns the communications area (COMMAREA) and return codes to the client program.

   Note: Steps 3 and 4 can repeated many times for the same UR.
5. When the EXCI client program is ready to commit or back out its changes, the program invokes RRS to begin the 2-phase commit protocol.
6. RRS acts as coordinator and either:
   - Asks the resource managers to prepare to commit all updates within the UR. (Note that resource managers other than the CICS server region may also have expressed an interest in the UR.) If all vote yes, RRS tells them to go ahead and commit the changes. If any vote no, all resource managers are told to perform backout.

- Tells all the resource managers that expressed an interest in the UR to perform backout of all the changes made with the UR.

The UR is now complete and CICS detaches the mirror task. If the EXCI client sends any new DPL requests after this point, EXCI starts a new unit-of-recovery and CICS attaches a new mirror transaction.

Each DPL request that specifies the SYNCONRETURN option attaches a new mirror task in the target CICS region. The first DPL request that does not specify SYNCONRETURN also attaches a new mirror task , but subsequent requests are directed to the same mirror task. When a syncpoint takes place, the mirror task ends, and the next non-SYNCONRETURN request attaches a new mirror. To see the effect of mixing DPL requests with and without the SYNCONRETURN option, see Figure 31.

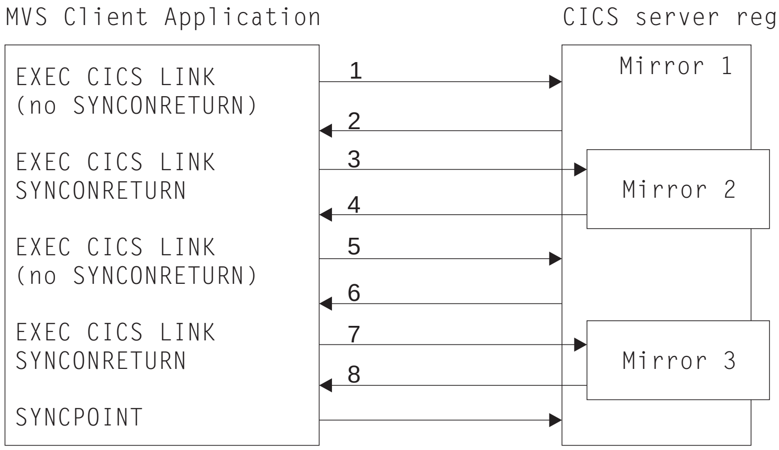


Figure 31. Effect of mixing DPL requests with and without SYNCONRETURN

1. Client issues DPL request omitting SYNCONRETURN.

   Because no mirror transaction is running, a new mirror (mirror 1) is attached.
2. The DPL request completes, and because it was issued without the SYNCONRETURN option, the mirror transaction waits for another request.
3. Client issues DPL request with the SYNCONRETURN option.

   A new mirror transaction (mirror 2) is attached.
4. On completion of the DPL request, resources updated by the mirror transaction are committed, and the mirror transaction ends.
5. Client issues another DPL request omitting SYNCONRETURN. Mirror 1 receives and executes the DPL request.
6. The DPL request completes, and once again the mirror transaction waits for another request.
7. Client issues DPL request with the SYNCONRETURN option.

   A new mirror transaction (mirror 3) is attached.
8. On completion of the DPL request, resources updated by the mirror transaction are committed, and the mirror transaction ends.
9. The client program requests a syncpoint. Resources updated by mirror 1 are committed, and the transaction ends.

Taking a syncpoint in the client program

To commit changes instigated by the client program, use one of the following MVS callable services:

Application_Commit_UR (SRRCMIT)
For a description of Application_Commit_UR, see *OS/390 MVS Programming: Callable Services for HLL*.

Commit_UR (ATRCMIT)
For a description of Commit_UR, see *OS/390 MVS Programming: Resource Recovery*.

To backout changes in the client program, use one of the following services:

Application_Backout_UR (SRRBACK)
For a description of Application_Backout_UR, see *OS/390 MVS Programming: Callable Services for HLL*.

Backout_UR (ATRBACK)
For a description of Backout_UR, see *OS/390 MVS Programming: Resource Recovery*.

If none of these interfaces is used, changes are committed or backed out explicitly when the client program either ends normally or abends. The use of implicit commit or backout is not recommended:

- The client program has no way of knowing if updates were committed or backed out; even if the program ends normally, a resource manager may choose to backout any changes.
- The run time environment for high level languages may intercept errors that would otherwise result in an operating system abend. If an error is intercepted in this way, and the client program does not take any explicit action, the program may terminate normally and updates committed. Your client program should be coded to ensure that resources are committed or backed out correctly in these situations. For example, a PL/I program might include an ON unit that issues SRRBACK when errors are encountered. A COBOL program might use the ON phrase on statements that could encounter errors to achieve the same result.

Requirements for the external CICS interface

Client programs running in an MVS address space can communicate only with CICS server regions running under Version 4 of CICS for MVS/ESA, or a later release.

Also, the client program can connect to the server CICS region only through the Version 4 of CICS for MVS/ESA (or later) level of the interregion communication program, DFHIRP.

For information about DFHIRP, and its requirement to be installed in the MVS extended link pack area (ELPA), see the *CICS Transaction Server for OS/390 Installation Guide*.

When you use RRMS to coordinate DPL requests, the server CICS region must be running CICS TS Release 3 or later.

Chapter 9. The EXCI CALL interface

The EXCI CALL interface consists of six commands that allow you to:

- Allocate and open sessions to a CICS system from non-CICS programs running under MVS
- Issue distributed program link (DPL) requests on these sessions from the non-CICS programs
- Close and deallocate sessions on completion of DPL requests.

The six EXCI commands are:

- Initialize_User
- Allocate_Pipe
- Open_Pipe
- DPL_Request
- Close_Pipe
- Deallocate_Pipe

The application program stub, DFHXCSTB: The EXCI commands invoke the external CICS interface via an application programming stub provided by CICS, called DFHXCSTB. You must include this stub when you link-edit your non-CICS program.

The CALL interface commands

In the description of each command that follows, the syntax box illustrates the assembler form of the command. The syntax shows VL,MF=(E,(1)) for each command, indicating the execute form of the CALL macro, with the parameter list storage area addressed by Register 1.

The commands are also supported by C/370, COBOL, and PL/I programming languages, using the CALL conventions appropriate for these languages.

There are examples of these CALLs, in all the supported languages, in the sample client programs provided by CICS. See “Sample application programs” on page 175 for information about these.

© Copyright IBM Corp. 1994, 1998

Initialize_User

Function

Initialize the user environment. This includes obtaining authority to use IRC facilities. The environment is created for the lifetime of the TCB, so the command needs to be issued only once per user per TCB. Further commands from this user must be issued under the same TCB.

Note: A *user* is a program that has issued an Initialize_user request (or for which an Initialize_User request has been issued), with a unique name per TCB. For example:

- A simple client program running under MVS can be a single user of the external CICS interface.
- A client program running under MVS can open several pipes and issue external CICS interface calls over them sequentially, on behalf of different vendor packages. In this case, from the viewpoint of the client program, each of the packages is a user, identified by a unique user name. Thus a single client program can operate on behalf of multiple users.
- A program running under MVS can attach several TCBs. Under each TCB, a vendor package issues external CICS interface calls on its own behalf. Each package is a client program in its own right, and runs under its own TCB. Each is also a user, with a unique user name.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─user_name),──VL,MF=(E,(1))──────────────────────────────────────────────────────►◄
```

Parameters

version_number

A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 145 for copybook details.

return_area

A 5-word output area to receive response and reason codes, and a message pointer field. For more details see “Return area for the EXCI CALL interface” on page 144.

user_token

A 1-word output area containing a 32-bit token supplied by the CICS external interface to represent the client program.

The user token corresponds to the *user-name* parameter. The client program must pass this token on all subsequent external CICS interface commands made for the user defined on the *user_name* parameter.

call_type

A 1-word input area indicating the function of the command. It must be set to 1 in the client program to indicate that this is an Initialize_User command.

The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is INIT_USER. See page 145 for copybook details.

user_name

An input area holding a name that identifies the user of the external CICS interface. Generally, this is the client program. If this user is to use a specific pipe, then the value in *user_name* must match that of the NETNAME attribute of the CONNECTION definition for the specific pipe.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the Initialize_User call:

Response OK

Command executed successfully (RC 0). Reason code:

0	Normal response

Response WARNING

The command executed successfully, but with an error (RC 4). Reason codes:

3	VERIFY_BLOCK_FM_ERROR
4	WS_FREEMAIN_ERROR

Response RETRYABLE

The command failed because of setup errors but can be reissued (RC 8). Reason code:

201	NO_CICS_IRC_STARTED

Response USER_ERROR

The command failed because of an error in either the client or the server (RC 12). Reason codes:

401	INVALID_CALL_TYPE
402	INVALID_VERSION_NUMBER
403	INVALID_USER_NAME
410	DFHMEBM_LOAD_FAILED
411	DFHMET4E_LOAD_FAILED
412	DFHXCURM_LOAD_FAILED
413	DFHXCTRA_LOAD_FAILED
419	CICS_AFCB_PRESENT
420	DFHXCOPT_LOAD_FAILED

421	RUNNING_UNDER_AN_IRB

Response SYSTEM_ERROR

The command failed (RC 16). Reason codes:

601	WS_GETMAIN_ERROR
602	XCGLOBAL_GETMAIN_ERROR
603	XCUSER_GETMAIN_ERROR
605	VERIFY_BLOCK_GM_ERROR
606	SSI_VERIFY_FAILED
607	CICS_SVC_CALL_FAILURE
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED
627	INCORRECT_SVC_LEVEL

For more information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

Allocate_Pipe

Function

Allocate a single session, or pipe, to a CICS region. This command does not connect the client program to a CICS region; this happens on the Open_Pipe command. You can allocate up to 100 pipes in an EXCI address space.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─pipe_token,──┬─CICS_applid,─┬──allocate_opts),──VL,MF=(E,(1))──────────────────►◄
               └─null_ptr,────┘
```

Parameters

version_number
: A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

  The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 145 for copybook details.

return_area
: A 5-word output area to receive response and reason codes, and a message pointer field. For more details see “Return area for the EXCI CALL interface” on page 144.

user_token
: The 1-word token returned on the Initialize_User command.

call_type
: A 1-word input area indicating the function of the command. It must be set to 2 in the client program to indicate that this is an Allocate_Pipe command.

  The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is ALLOCATE_PIPE. See page 145 for copybook details.

pipe_token
: A 1-word output area. CICS returns a 32-bit token in this area to represent the allocated session. This token must be used on any subsequent command that uses this session.

CICS_applid (or null_ptr)
: An 8-byte input area containing the generic applid of the CICS system to which the allocated session is to be connected.

  Although an applid is required to complete the Allocate_Pipe function, this parameter is optional on the Allocate_Pipe call. You can either specify the applid on this parameter to the Allocate_Pipe call, or in the user-replaceable module, DFHXCURM, using the URMAPPL parameter (DFHXCURM is always invoked during Allocate_Pipe processing). You can also use the URMAPPL parameter in DFHXCURM to override an applid specified on the Allocate_Pipe call. See “Chapter 12. The EXCI user-replaceable module” on page 165 for information about the URMAPPL parameter.

If you omit the applid from the call, you must ensure that the CALL parameter list contains a null address for *CICS_applid*. How you do this depends on the language you are using for the non-CICS client program. For an example of a call that omits an optional parameter, see “Example of EXCI CALLs with null parameters” on page 146.

allocate_opts

A 1-byte input area to represent options specified on this command. The options specify which type of session is to be used—specific or generic. X'00' represents a specific session. X'80' represents a generic session.

The equated value for these options in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) are SPECIFIC_PIPE and GENERIC_PIPE. See page 145 for copybook details.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the Allocate_Pipe call:

Response OK

Command executed successfully (RC 0). Reason code:

0	Normal response

Response USER_ERROR

The command failed because of an error in either the client or the server (RC 12). Reason codes:

401	INVALID_CALL_TYPE
402	INVALID_VERSION_NUMBER
404	INVALID_USER_TOKEN
421	RUNNING_UNDER_AN_IRB

Response SYSTEM_ERROR

The command failed (RC 16). Reason codes:

604	XCPIPE_GETMAIN_ERROR
608	IRC_LOGON_FAILURE
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED
628	IRP_LEVEL_CHECK_FAILURE

For information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

Open_Pipe

Function

Cause IRC to connect an allocated pipe to a receive session of the appropriate connection defined in the CICS region named either on the Allocate_Pipe command, or in DFHXCURM. The appropriate connection is either:

- The EXCI connection with a NETNAME value equal to the *user_name* parameter on the Initialize_User command (that is, you are using a specific connection, dedicated to this client program), or
- The EXCI connection defined as generic.

Note: This command should be used only when there is a DPL call ready to be issued to the CICS server region. When not in use, EXCI sessions should not be left open.

If you shut down CICS without the support of the CICS-supplied shutdown-assist transaction (CESD) or an equivalent, and sessions are left open, CICS may not be able to shut its IRC facility in an orderly manner. A normal shutdown of CICS without the support of the shutdown assist transaction waits if any EXCI sessions are not closed. CICS issues message DFHIR2321 indicating the following information:

- The netname of the session if it is on a specific connection
- The word GENERIC if the open sessions are on a generic connection.

If you use the CICS-supplied shutdown-assist transaction, CESD, sessions left open do not present a problem to normal shutdown, because CESD issues an immediate close of IRC.

Provided that at least one DPL_Request call has been issued on the session, message DFHIR2321 also shows the job name, step name, and procedure name of the client job that is using the session, and the MVS ID of the MVS image on which the client program is running.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─pipe_token),──VL,MF=(E,(1))───────────────────────────────────────────────────►◄
```

Parameters

version_number
A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 145 for copybook details.

return_area
A 5-word output area to receive response and reason codes, and a message pointer field. For more details, see “Return area for the EXCI CALL interface” on page 144.

user_token
The 1-word token returned on the Initialize_User command.

call_type
A 1-word input area indicating the function of the command. This must be set to 3 in the client program to indicate that this is an Open_pipe command.

The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is OPEN_PIPE. See page 145 for copybook details.

pipe_token
A 1-word output area containing the token passed by CICS on the Allocate_Pipe command. It represents the pipe being opened on this command.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the Open_Pipe call:

Response OK
Command executed successfully (RC 0). Reason code:

0 NORMAL

Response WARNING
The command executed successfully, but with an error (RC 4). Reason codes:

1 PIPE_ALREADY_OPEN

Response RETRYABLE
The command failed because of setup errors but can be reissued (RC 8). Reason codes:

202 NO_PIPE

203 NO_CICS

Response USER_ERROR
The command failed because of an error in either the client or the server (RC 12). Reason codes:

401 INVALID_CALL_TYPE

402 INVALID_VERSION_NUMBER

404 INVALID_USER_TOKEN

418 INVALID_PIPE_TOKEN

421 RUNNING_UNDER_AN_IRB

Response SYSTEM_ERROR

The command failed (RC 16). Reason codes:

609	IRC_CONNECT_FAILURE
621	PIPE_RECOVERY_FAILURE
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED

For information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

DPL_Request

Function

Issue a distributed program link request across an open pipe connected to the CICS system on which the server (or target) application program resides. The command is synchronous, and the TCB waits for a response from CICS. Once a pipe is opened, any number of DPL requests can be issued before the pipe is closed. To the server program, the link request appears just like a standard EXEC CICS LINK request from another CICS region, and it is not aware that it is sent from a non-CICS client program using EXCI.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─pipe_token,──pgmname,──┬─COMMAREA,─┬──COMMAREA_len,──data_len,─────────────────►
                         └─null_ptr,─┘

►──┬─transid,──┬───┬─uowid,───┬───┬─userid,──┬──dpl_retarea,──┬─DPL_opts─┬──)─────►
   └─null_ptr,─┘   └─null_ptr,┘   └─null_ptr,┘                └─null_ptr─┘

►─VL,MF=(E,(1))──────────────────────────────────────────────────────────────────►◄
```

Parameters

version_number
: A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

  The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 145 for copybook details.

return_area
: A 5-word output area to receive response and reason codes, and a message pointer field. For more details, see “Return area for the EXCI CALL interface” on page 144.

user_token
: A 1-word input area specifying the user token returned to the client program on the Initialize_User command.

call_type
: A 1-word input area indicating the function of the command. This must be set to 6 in the client program to indicate that the pipe is now being used for the DPL_Request call.

  The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is DPL_REQUEST. See page 145 for copybook details.

pipe_token

A 1-word input area specifying the token returned by EXCI on the Allocate_Pipe command. It represents the pipe being used for the DPL_Request call.

pgmname

The 8-character name of the CICS application program being called as the server program.

This is either the name as specified on a predefined PROGRAM resource definition installed in the CICS server region, or as it is known to a user-written autoinstall program if the program is to be autoinstalled. The program can be defined in the CICS server region as a local program, or it can be defined as remote. Programs defined as remote enable “daisy-chaining”, where EXCI-CICS DPL calls become EXCI-CICS-CICS DPL calls.

COMMAREA (or null_ptr)

A variable length input area for the communications area (COMMAREA) between the client and server programs. The length is defined by *COMMAREA_len*.

This is the storage area that contains the data to be sent to the CICS application program. This area is also used to receive the updated COMMAREA from the CICS application program (the server program).

This parameter is optional. If it is not required, you must ensure that the CALL parameter list contains a null address for this parameter. How you do this depends on the language you are using for the non-CICS client program. For an example of a call that omits an optional parameter, see “Example of EXCI CALLs with null parameters” on page 146.

COMMAREA_len

A fullword binary input area. This parameter specifies the length of the COMMAREA. It is also the length of the server program’s COMMAREA (EIBCALEN).

If you specify a COMMAREA, you must also specify this parameter to define the length.

If you don’t specify a COMMAREA, this parameter is ignored.

data_len

A fullword binary input area. This parameter specifies the length of contiguous storage, from the start of the COMMAREA, to be sent to the server program.

This parameter restricts the amount of data sent to the server program, and should be used to optimize performance if, for example, the COMMAREA is large but the amount of data being passed is small.

Note that on return from the server program, the EXCI data transformer program ensures that the COMMAREA in the non-CICS client program is the same as that of the server program. This caters for the following conditions:

- The data returned is more than the data passed in the original COMMAREA.
- The data returned is less than the data passed in the original COMMAREA.
- There is no data returned because it is unchanged.
- The server is returning null data.

The value of *data_len* must not be greater than the value of *COMMAREA_len*. A value of zero is valid and results in no data being sent to the server program.

If you don't specify a COMMAREA, this parameter is ignored.

transid (or null_ptr)

A 4-character input area containing the id of the CICS mirror transaction under which the server program is to run. This transaction must be defined to the CICS server region, and its definition *must observe the following rules*:

- It must *not* specify the server program as the initial program of the transaction
- It *must* specify the mirror program DFHMIRS, and the profile DFHCICSA.

Failure to specify DFHMIRS as the initial program means that a COMMAREA passed from the client application program is not passed to the CICS server program. Furthermore, the DPL request fails and the client application program receives a response of SYSTEM_ERROR and reason SERVER_PROTOCOL_ERROR.

The DFHCICSA profile specifies the correct value for the INBFMH parameter, which must be specified as INBFMH(ALL) for a mirror transaction.

When the CICS server region receives a DPL request, it attaches the mirror transaction and invokes DFHMIRS. The mirror program then passes control to the requested server program, passing the COMMAREA supplied by the client program. The COMMAREA passed to the server program is primed with the data only, the remainder of the COMMAREA being set to nulls.

The purpose of the *transid* parameter is to distinguish between different invocations of the server program. This enables you to run different invocations of the server program under transactions that specify different attributes. For example, you can vary the transaction priorities, or the security requirements.

A transid is optional. By default, the CICS server region uses the CICS-supplied mirror transaction, CSMI. If you don't want to specify *transid*, you must ensure that the CALL parameter list contains a null address for this parameter. How you do this depends on the language you are using for the non-CICS client program. For an example of a call that omits an optional parameter, see "Example of EXCI CALLs with null parameters" on page 146.

| If you issue multiple requests in the same MVS unit-of-recovery, the same transid must be used in all of them.

uowid (or null_ptr)

| An input area containing a unit-of-work identifier, using the APPC architected format, that is passed on the DPL_Request for correlation purposes.

| For DPL requests that are committed when the CICS program returns control to the MVS application, this parameter is optional.

| For DPL requests that are part of an RRMS unit-of-recovery, null_ptr must be specified. The unit-of-work identifier that is already associated with the RRMS unit-of-recovery is used, if there is one; if not, CICS (or another RRMS resource manager) generates a unique unit-of-work identifier and associates it with the RRMS unit-of-recovery.

If you don’t want to specify *uowid*, you must ensure that the CALL parameter list contains a null address for this parameter. How you do this depends on the language you are using for the non-CICS client program. For an example of a call that omits an optional parameter, see “Example of EXCI CALLs with null parameters” on page 146.

The *uowid* parameter is passed to the CICS server region, which uses it as the UOWID for the first unit of work executed by the CICS server program. If the server program issues intermediate syncpoints before returning to the client program, CICS uses the supplied *uowid* for the subsequent units of work, but with the two-byte sequence number incremented for each new logical unit of work. If the CICS server program updates remote resources, the client-supplied UOWID is distributed to the remote systems that own the resources.

The *uowid* parameter is supplied on the EXCI CALL interface for correlation purposes only, to allow units of work that originated from a particular client program to be identified in CICS. The *uowid* is not provided for recovery purposes between CICS and the client program.

The *uowid* can be a maximum of 27 bytes long and has the following format:

- A 1-byte length field containing the overall length of the UOWID (excluding this field).
- A 1-byte length field containing the length of the logical unit name (excluding this field).
- A logical unit name field of variable length up to a maximum of 17 bytes.

  To conform to APPC architecture rules, the LUNAME must be of the form *AAAAAAAA.BBBBBBBB*, where *AAAAAAAA* is optional and:
  - AAAAAAAA and BBBBBBBBare 18–byte names separated by a period
  - If AAAAAAAA is omitted, the period must also be omitted
  - AAAAAAAA and BBBBBBBB must be type–1134 symbol strings (that is, character strings consisting of one or more EBCDIC uppercase letters A–Z and 0–9, the first character of which must be an uppercase letter).
- The clock value—the middle 6 bytes of an 8-byte store clock (STCK) value.
- A 2-byte sequence number.

If you omit a unit-of-work identifier (by specifying a null pointer), and the DPL request is not part of an RRMS unit-of-recovery, the external CICS interface generates one for you, consisting of the following:

- A 1-byte length field set to X'1A'.
- A 1-byte LU length field set to X'11'.
- A 17-byte LU name consisting of:
  - An 8-byte eyecatcher set to ‘DFHEXCIU’.
  - A 1-byte field containing a period (.)
  - A 4-byte field containing the MVS system identifier (SYSID), in characters, under which the client is running.
  - A 4-byte field containing the address space id (ASID) in which the MVS client program is running. The field contains the four character EBCDIC representation of the two–byte hex address space id.
- The clock value—the middle 6 bytes of an 8-byte store clock (STCK) value
- A two—byte sequence number set to X'0001'.

userid (or null_ptr)

An 8-character input area containing the RACF userid for user security checking in the CICS region. The external CICS interface passes this userid to the CICS server region for user resource and command security checking in the server application program.

A userid is required only if the MRO connection specifies the ATTACHSEC(IDENTIFY) attribute. If the connection specifies ATTACHSEC(LOCAL), the CICS server region applies link security checking. See the *CICS RACF Security Guide* for information about link security on MRO connections.

See also "Chapter 15. Security" on page 187 for information about external CICS interface security considerations.

This parameter is optional. However, if you don't specify a userid, the external CICS interface passes the security userid under which the client program is running. For example, if the client program is running as an MVS batch job, the external CICS interface obtains and passes the userid specified on the USER parameter of the JOB statement.

If you specify a userid and SURROGCHK=YES is specified in the EXCI options table DFHXCOPT, the userid under which the EXCI job is running is subject to a surrogate user check. This check is performed by the external CICS interface to ensure that the client job userid is authorized to use the userid specified on the DPL call. For more information about surrogate user security checking, see "Chapter 15. Security" on page 187.

If you want to let *userid* default, you must ensure that the CALL parameter list contains a null address for this parameter. How you do this depends on the language you are using for the non-CICS client program. For an example of a call that omits an optional parameter, see "Example of EXCI CALLs with null parameters" on page 146.

| If you issue multiple requests in the same MVS unit-of-recovery, the same userid must be used in all of them. If the unit-of-recovery also includes EXEC CICS calls, you should allow the userid on all DPL_requests to default to the security userid under which the client program is running.

dpl_retarea

A 12-byte output area into which the DPL_Request processor places responses to the DPL request. Generally, these responses are from CICS, but in some cases the error detection occurs in the external CICS interface, which returns exception conditions that are the equivalent of those returned by an EXEC CICS LINK command.

This field is only meaningful in the following circumstances:

- The response field of the EXCI *return-area* has a zero value, or
- The EXCI *return-area* indicates that the server program has abended (response=USER_ERROR and reason=SERVER_ABENDED).

The 12 bytes form three fields, providing the following information:

Field 1 (*fullword value*)

This field is a fullword containing a RESP value from the DPL_Request call. See "Error codes" on page 152 for the RESP values that can be returned on a DPL_Request call.

If the DPL_Request call reaches CICS, this field contains the EIBRESP value, otherwise it contains an equivalent response set by the external CICS interface. If this field is set by the external CICS interface, RESP is further qualified by a RESP2 value in the second field.

A zero value is the normal response, which equates to EXEC_NORMAL in the return codes copybooks.

Field 2 (*fullword value*)

This field is a fullword that may contain a RESP2 value from the link request, further qualifying the RESP value in field 1.

If the DPL_Request call reaches CICS, the RESP2 field generally is null (CICS does not return RESP2 values across MRO links). However, if the RESP field indicates SYSIDERR (value 53), this field provides a reason code. See"Dpl_retarea return codes" on page 145 for more information.

If the RESP field is set by the external CICS interface, it is further qualified by a RESP2 value in this second field. For example, if the *data_len* parameter specifies a value greater than the *COMMAREA_len* parameter, the external CICS interface returns the RESP value 22 (which equates to EXEC_LENGERR in the return codes copybooks), and a RESP2 value of 13.

See the LINK conditions in the *CICS Application Programming Reference* for full details of the possible RESP and RESP2 values.

Note: The data transformer program makes special use of the RESP2 field. If any error occurs in the transformer, the error is returned in RESP2.

Field 3 (*fullword value*)

The third field, a 4-character field, contains:

- The abend code if the server program abended
- Four blanks if the server program did not abend.

If a server program abends, it is backed out to its last syncpoint which may be the start of the task, or an intermediate syncpoint. The server program can issue intermediate syncpoints because SYNCONRETURN is forced.

DPL_opts (or null_ptr)

A 1-byte input area indicating options to be used on the DPL_Request call.

If you omit this parameter, it defaults to the value X'00' (see below). If you want to omit *DPL_opts* and let it default, ensure that the CALL parameter list contains a null address for this parameter. How you do this depends on the language you are using for the non-CICS program. For an example of a call that omits an optional parameter, see "Example of EXCI CALLs with null parameters" on page 146.

Currently, the *DPL_opts* parameter applies only to resource recovery, using the following values:

X'00' Indicates that you want the client batch program to control resource recovery, using 2-phase commit protocols supported by MVS RRS. With this option, the CICS server region does not perform a syncpoint when the server program returns control to CICS. Furthermore, the CICS server application program must not take any explicit syncpoints, otherwise it is abended by CICS. For more information, see "Resource recovery" on page 117.

X'80' indicates that SYNCONRETURN is required in the CICS server region.

SYNCONRETURN specifies that the server region is to take a syncpoint on successful completion of the server program, independently of the client program. This option does *not* prevent a server program from taking its own explicit syncpoints.

The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is SYNCONRETURN. See page 145 for copybook details.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the DPL call:

Response OK
Command executed successfully (RC 0). Reason code:

0 NORMAL

Response WARNING
The command executed successfully, but with an error (RC 4). Reason codes:

6 IRP_IOAREA_FM_FAILURE

7 SERVER_TERMINATED

Response RETRYABLE
The command failed because of setup errors but can be reissued (RC 8). Reason codes:

203 NO_CICS

204 WRONG_MVS_FOR_RRMS

205 RRMS_NOT_AVAILABLE

Response USER_ERROR
The command failed because of an error in either the client or the server (RC 12). Reason codes:

401 INVALID_CALL_TYPE

402 INVALID_VERSION_NUMBER

404 INVALID_USER_TOKEN

406 PIPE_NOT_OPEN

407	INVALID_USERID
408	INVALID_UOWID
409	INVALID_TRANSID
414	IRP_ABORT_RECEIVED
415	INVALID_CONNECTION_DEFN
416	INVALID_CICS_RELEASE
417	PIPE_MUST_CLOSE
418	INVALID_PIPE_TOKEN
421	RUNNING_UNDER_AN_IRB
422	SERVER_ABENDED
423	SURROGATE_CHECK_FAILED
425	UOWID_NOT_ALLOWED

Response SYSTEM_ERROR

The command failed (RC 16). Reason codes:

612	TRANSFORM_1_ERROR
613	TRANSFORM_4_ERROR
614	IRP_NULL_DATA_RECEIVED
615	IRP_NEGATIVE_RESPONSE
616	IRP_SWITCH_PULL_FAILURE
617	IRP_IOAREA_GM_FAILURE
619	IRP_BAD_IOAREA
620	IRP_PROTOCOL_ERROR
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED
624	SERVER_TIMEDOUT
625	STIMER_SETUP_FAILURE
626	STIMER_CANCEL_FAILURE
629	SERVER_PROTOCOL_ERROR
630	RRMS_ERROR
631	RRMS_SEVERE_ERROR
632	XCGUR_GETMAIN_ERROR

For information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

Close_PIPE

Function

Disconnect an open pipe from CICS. The pipe remains in an allocated state, and its tokens remain valid for use by the same user. To reuse a closed pipe, the client program must first reissue an Open_Pipe command using the pipe token returned on the Allocate_Pipe command for the pipe. Pipes should not be left open when not in use because this prevents CICS from shutting down its IRC facility in an orderly manner. Therefore, the Close_Pipe command should be issued as soon as possible after all DPL_Request calls have completed.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─pipe_token),──VL,MF=(E,(1))─────────────────────────────────────────────────────►◄
```

Parameters

version_number

A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 142 for copybook details.

return_area

A 5-word output area to receive response and reason codes, and a message pointer field. For more details, see “Return area for the EXCI CALL interface” on page 144.

user_token

The 1-word input area specifying the token, returned to the client program by EXCI on the Initialize_User command, that represents the user of the pipe being closed.

call_type

A 1-word input area indicating the function of the command. This must be set to 4 in the client program to indicate that this is a Close_Pipe command.

The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is CLOSE_PIPE. See page 145 for copybook details.

pipe_token

A 1-word input area specifying the token, returned to the client program by EXCI on the original Allocate_Pipe command, that represents the pipe being closed.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the Close_Pipe call:

Response OK
Command executed successfully (RC 0). Reason code:

0	NORMAL

Response WARNING
The command executed successfully, but with an error (RC 4). Reason codes:

2	PIPE_ALREADY_CLOSED

Response USER_ERROR
The command failed because of an error in either the client or the server (RC 12). Reason codes:

401	INVALID_CALL_TYPE
402	INVALID_VERSION_NUMBER
404	INVALID_USER_TOKEN
418	INVALID_PIPE_TOKEN
421	RUNNING_UNDER_AN_IRB

Response SYSTEM_ERROR
The command failed (RC 16). Reason codes:

610	IRC_DISCONNECT_FAILURE
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED

For information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

Deallocate_Pipe

Function

Deallocate a pipe from CICS. On completion of this command, the pipe can no longer be used, and its associated tokens are invalid. This command should be issued for pipes that are no longer required. This command frees storage associated with the pipe.

Syntax

CALL DFHXCIS

```
►►──CALL DFHXCIS,(version_number,──return_area,──user_token,──call_type,──────────►

►─pipe_token),──VL,MF=(E,(1))────────────────────────────────────────────────────►◄
```

Parameters

version_number
A fullword binary input area indicating the version of the external CICS interface parameter list being used. It must be set to 1 in the client program.

The equated value for this parameter in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is VERSION_1. See page 145 for copybook details.

return_area
A 5-word output area to receive response and reason codes, and a message pointer field. For more details, see "Return area for the EXCI CALL interface" on page 144.

user_token
A 1-word input area containing the token returned on the Initialize_User command.

call_type
A 1-word input area indicating the function of the command. This must be set to 5 in the client program to indicate that this is a Deallocate_Pipe command.

The equated value for this call in the CICS-supplied copybook DFHXCPL*x* (where *x* indicates the language) is DEALLOCATE_PIPE. See page 145 for copybook details.

pipe_token
A 1-word input area containing the token passed back on the original Allocate_Pipe command, that represents the pipe now being deallocated.

Responses and reason codes

For all non-zero response codes, a unique reason code value identifies the reason for the response.

Note: All numeric response and reason code values are in decimal.

The following is a summary of the response and reason codes that the external CICS interface can return on the Deallocate_Pipe call:

Response OK

Command executed successfully (RC 0). Reason code:

0	NORMAL

Response WARNING

The command succeeded successfully, but with an error (RC 4). Reason codes:

5	XCPIPE_FREEMAIN_ERROR
6	IRP_IOAREA_FM_FAILURE

Response USER_ERROR

The command failed because of an error in either the client or the server (RC 12). Reason codes:

401	INVALID_CALL_TYPE
402	INVALID_VERSION_NUMBER
404	INVALID_USER_TOKEN
405	PIPE_NOT_CLOSED
418	INVALID_PIPE_TOKEN
421	RUNNING_UNDER_AN_IRB

Response SYSTEM_ERROR

The command failed (RC 16). Reason codes:

611	IRC_LOGOFF_FAILURE
622	ESTAE_SETUP_FAILURE
623	ESTAE_INVOKED

For information about response codes, see “Response code values” on page 144.

For information about the reason codes, see “Chapter 17. Response and reason codes returned on EXCI calls” on page 211.

Response code values

The values that can be returned in the response field are shown in Table 11 (all values are in decimal).

Table 11. EXCI response codes (returned in response field of return_area)

Value	Meaning	Explanation
0	OK	For all EXCI CALL commands other than the DPL_request, the call was successful. If an OK response is received for a DPL_request, you must also check *dpl_retarea* to ensure CICS did not return a condition code. If the EIBRESP field of *Dpl_retarea* is zero, the DPL call was successful.
4	WARNING	The external CICS interface detected an error, but this did not stop the CALL command completing successfully. The reason code field describes the error detected.
8	RETRYABLE	The EXCI CALL command failed. This class of failure relates to errors in the setup of the system environment, and not errors in the external CICS interface or client program. The reason code documents the specific error in the environment setup. The external CICS interface command can be reissued without changing the client program once the environment error has been corrected. The environmental errors concerned are ones that do not require an MVS re-IPL. Each reason code value for a RETRYABLE response documents whether the CALL can be reissued directly, or whether the pipe being used has to be closed and reopened first.
12	USER_ERROR	The EXCI CALL command failed. This class of error means there is an error either in the client program, or in the CICS server program, or in the CICS server region. An example of an error in the CICS server system would be a failed security check, or an abend of the CICS server program, in which case the abend code is set in the abend code field of *dpl_retarea*. Each reason code value for a response of USER_ERROR explains whether the command can be reissued directly, or whether the pipe being used has to be closed and reopened first.
16	SYSTEM_ ERROR	The EXCI CALL command failed. This class of error means that the external CICS interface has detected an error. The reason code value identifies the specific error. If the error can be corrected, then the command can be reissued. Each reason code value for a SYSTEM_ERROR response explains whether the command can be reissued directly, or whether the pipe being used has to be closed and reopened first.

Return area for the EXCI CALL interface

The format of the 5-word return area for the EXCI CALL interface is as follows:

1. 1–word response field.
2. 1–word reason field.
3. Two 1–word subreason fields—subreason field-1 and subreason field-2.
4. 1–word CICS message pointer field. This is zero if there is no message present. If a message is present, this field contains the address of the storage area containing the message, which is formatted as follows:
   - A 2-byte LL field. LL is the length of the message plus the length of the LLBB field.

- A 2-byte BB field, set to binary zero.
- A variable length field containing the text of the message.

Return area and function call EQUATE copybooks

CICS provides four language-specific copybooks that map the storage areas for the *return_area* and *dpl_retarea* parameters of the EXCI CALL commands. The copybooks also provide EQUATE statements for each type of EXCI CALL.

These copybooks, and the libraries they are supplied in for the supported languages, are shown in Table 12.

Table 12. Supplied copybooks of return areas and equated names

Copybook name	Language	Library
DFHXCPLD	Assembler	CICSTS13.CICS.SDFHMAC
DFHXCPLH	C	CICSTS13.CICS.SDFHC370
DFHXCPLO	COBOL	CICSTS13.CICS.SDFHCOB
DFHXCPLL	PL/I	CICSTS13.CICS.SDFHPL1

Return codes

All the possible return codes are contained in a CICS-supplied copybook, which you must include in the program source of your external, non-CICS program. The names of the copybooks for the supported languages, and the libraries they are supplied in, are shown in Table 13.

Table 13. Supplied copybooks of RESPONSE and REASON codes

Copybook name	Language	Library
DFHXCRCD	Assembler	CICSTS13.CICS.SDFHMAC
DFHXCRCH	C	CICSTS13.CICS.SDFHC370
DFHXCRCO	COBOL	CICSTS13.CICS.SDFHCOB
DFHXCRCL	PL/I	CICSTS13.CICS.SDFHPL1

| OS/390 provides copybooks for use with the interfaces described on “Taking a syncpoint in the client program” on page 121. These are described in *OS/390 MVS Programming: Resource Recovery* and *OS/390 MVS Programming: Callable Services for HLL*.

Dpl_retarea return codes

These are the same as for CICS-to-CICS EXEC CICS DPL commands but with the following additions for the EXCI call interface:

Table 14. Exceptional conditions. RESP and RESP2 values returned to dpl_retarea

Condition	RESP2	Meaning
LENGERR	22	COMMAREA_LEN_TOO_BIG
LENGERR	23	COMMAREA_BUT_NO_COMMAREA_LEN

| SYSIDERR also can be returned on an EXCI DPL_Request, if the DPL_Request specifies a program defined in the CICS server region as a remote program, and the link between the server and the remote CICS region is not open. In this situation, SYSIDERR is returned in the first word of the DPL_Retarea (code 53).

The reason code qualifying SYSIDERR is placed in the second word of this area (the equivalent of a RESP2 value).For SYSIDERR, the information stored in this field is derived from bytes 1 and 2 of the CICS EIBRCODE field. For example, if a remote link is not available, the EIBRCODE value stored in bytes 2 and 3 of the DPL_Retarea RESP2 field is X'0800'. For a list of the SYSIDERR reason codes that can be returned in the RESP2 field, see the SYSIDERR section of the notes on EIBRCODE in the *CICS Application Programming Reference* manual.

TERMERR also may be returned on an EXCI DPL request if the DPL request was to a program defined as remote, and an unrecoverable error occurs during conversation with the mirror on the remote CICS system. For example, suppose client program BATCH1 issues an EXCI DPL request to CICSA for program PROG1, which is defined as remote, and the request is function-shipped to CICSB where the program resides. If the session between CICSA and CICSB fails, or CICSB itself fails whilst executing the program PROG1, then TERMERR is returned to CICSA, and in turn to BATCH1.

No unique EXCI_DPL_RESP2 values are returned for TERMERR, PGMIDERR, NOTAUTH, and ROLLBACK.

Example of EXCI CALLs with null parameters

If you omit an optional parameter, such as *userid* on a DPL_Request, you must ensure that the parameter list is built with a null address for the missing parameter. The example that follows illustrates how to issue an EXCI DPL_Request with the *userid* and *uowid* parameters omitted in a COBOL program.

DPL CALL without userid and uowid (COBOL): In this example, the DPL parameters used on the call are defined in the WORKING-STORAGE SECTION, as follows:

DPL parameter	COBOL variable	Field definition
version_number	01 VERSION-1	PIC S9(8) COMP VALUE 1.
return_area	01 EXCI-RETURN-CODE.	(structure)
user_token	01 USER-TOKEN	PIC S9(8) COMP VALUE ZERO.
call_type	03 DPL-REQUEST	PIC S9(8) COMP VALUE 6.
pipe_token	01 PIPE-TOKEN	PIC S9(8) COMP VALUE ZERO.
pgmname	01 TARGET-PROGRAM	PIC X(8) VALUE ″DFHœAXCS″.
commarea	01 COMMAREA.	(structure)
commarea_len	01 COMM-LENGTH	PIC S9(8) COMP VALUE 98.
data_len	01 DATA-LENGTH	PIC S9(8) COMP VALUE 18.
transid	01 TARGET-TRANSID	PIC X(4) VALUE ″EXCI″.
dpl_retarea	01 EXCI-DPL-RETAREA.	(structure)
dpl_opts	01 SYNCONRETURN	PIC X VALUE X'80'.

The variable used for the null address is defined in the LINKAGE SECTION, as follows:

```
LINKAGE  SECTION.
  01    NULL-PTR              USAGE  IS  POINTER.
```

Using the data names specified in the WORKING-STORAGE SECTION as described above, and the NULL-PTR name as described in the LINKAGE

SECTION, the following invocation of the DPL function omits the *uowid* and the *userid* parameters, and replaces them in the parameter list with the NULL-PTR variable:

```
       DPL-SECTION.
      *
           SET ADDRESS OF NULL-PTR TO NULLS.
      *
           CALL 'DFHXCIS' USING  VERSION-1   EXCI-RETURN-CODE  USER-TOKEN
                               DPL-REQUEST    PIPE-TOKEN   TARGET-PROGRAM
                               COMMAREA       COMM-LENGTH  DATA-LENGTH
                               TARGET-TRANSID NULL-PTR     NULL-PTR
                               EXCI-DPL-RETAREA    SYNCONRETURN.
```

This example is taken from the CICS-supplied sample external CICS interface program, DFH0CXCC, which is supplied in CICSTS13.CICS.SDFHSAMP. For an example of how to omit the same parameters from the DPL call in the other supported languages, see the following sample programs:

DFH$AXCC
The assembler sample

DFH$PXCC
The PL/I sample

DFH$DXCC
The C/370™ sample.

Chapter 10. The EXEC CICS interface

This chapter describes the EXEC CICS LINK PROGRAM command for the external CICS interface. It covers the following topics:

- “EXEC CICS LINK command” on page 150
- “Retries on an EXEC CICS LINK command” on page 154
- “Translation required for EXEC CICS LINK command” on page 156

The external CICS interface provides this as a single, composite command, to invoke all the calls of the EXCI CALL interface. Each time you issue an EXEC CICS LINK PROGRAM command in a client application program, the external CICS interface invokes on your behalf each of the six EXCI calls.

© Copyright IBM Corp. 1994, 1998

EXEC CICS LINK command

Purpose

Link from an MVS client program to the specified server program in a server CICS region.

Format

LINK

```
►►──LINK──PROGRAM(name)──RETCODE(data-area)──┬──────────────┬──────────────────────────►
                                             └─SYNCONRETURN─┘

►──┬─────────────────────────────────────────┬──┬────────────────┬─────────────────────►
   └─COMMAREA(data-area)───LENGTH(data-value)─┘  └─APPLID(name)───┘

►──┬───────────────┬──┬─────────────────────────┬─────────────────────────────────────►◄
   └─TRANSID(name)─┘  └─DATALENGTH(data-value)──┘
```

Notes:

1.

Error conditions:LENGERR, LINKERR, NOTAUTH, PGMIDERR, ROLLEDBACK, SYSIDERR, TERMERR, WARNING

Comments

With the exception of the APPLID and RETCODE parameters, the external CICS interface parameters for an EXEC CICS LINK command are the same as for a CICS-CICS DPL command.

This book describes only those parameters that you can use with the external CICS interface. For programming information about the EXEC CICS LINK PROGRAM command, see the *CICS Application Programming Reference* manual.

Note that the LENGTH and DATALENGTH parameters specify halfword binary values, unlike the corresponding *COMMAREA_len* and *data_len* parameters of the EXCI CALL interface, which specify fullword values.

An external CICS interface EXEC CICS LINK command always uses a generic connection.

Parameters

The parameters that you can use on the external CICS interface form of the LINK command, are as follows:

APPLID(*name*)
Specifies the generic APPLID of the target CICS server region.

Although an applid is required for an external CICS interface command, this parameter is optional on the LINK command itself because you can also specify it in the user-replaceable module, DFHXCURM. If you omit the generic APPLID from the LINK command, you must ensure it is specified by the user-replaceable module, DFHXCURM, on the URMAPPL parameter. You can also use the URMAPPL parameter in DFHXCURM to override an applid specified on the LINK command. See “Chapter 12. The EXCI user-replaceable module” on page 165 for information about the URMAPPL parameter.

COMMAREA(*data-area*)
Specifies a communication area that is to be made available to the invoked program. In this option, a pointer to the data area is passed.

See the *CICS Application Programming Guide* for more information about passing data to CICS application programs.

DATALENGTH(*data-value*)
Specifies a halfword binary value that is the length of a contiguous area of storage, from the start of the COMMAREA, to be passed to the invoked program. If the amount of data being passed in a COMMAREA is small, but the COMMAREA itself is large so that the linked-to program can return the requested data, you should specify DATALENGTH in the interest of performance.

LENGTH(*data-value*)
Specifies a halfword binary value that is the length in bytes of the communication area.

PROGRAM(*name*)
Specifies the program name (1-8 characters) of the CICS server application program to which control is to be passed unconditionally. The specified name must either have been defined as a program to CICS, or the CICS server region must be capable of autoinstalling a definition for the named program.

Note the use of quotes:

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

PROGX is in quotes because it is the program name.

```
EXEC CICS LINK PROGRAM(DAREA)
```

DAREA is not in quotes because it is the name of a data area that contains the 8-character program name.

RETCODE(*data-area*)
Specifies a 20-byte area into which the external CICS interface places return code information. This area is formatted into five 1–word fields as follows:

RESP The primary response code indicating whether the external CICS interface LINK command caused an exception condition during its execution.

RESP2
The secondary response code that further qualifies, where necessary, some of the conditions raised in the RESP parameter.

ABCODE
Contains a valid CICS abend code if the server program abended in the server region.

MSGLEN
Indicates the length of the message (if any) issued by the CICS server region during the execution of the server program. Note that the length is the actual length of the message text only, and does not include this 1—word length field.

MSGPTR
This is the address of the message text returned by the CICS server region.

Note: MSGLEN and MSGPTR are only valid on a LINKERR condition, with the RESP2 value 414.

SYNCONRETURN
Specifies that the CICS server region, named on the APPLID parameter, is to take a syncpoint on successful completion of the server program.

TRANSID(*name*)
Specifies the name of the mirror transaction that the remote region is to attach, and under which it is to run the server program. If you omit the TRANSID option, the CICS server region attaches CSMI.

Note: The TRANSID option specified on the LINK command overrides any TRANSID option specified on the program resource definition installed in the CICS server region.

While you can specify your own name for the mirror transaction initiated by DPL requests, the transaction *must* be defined in the server region, and the transaction definition must specify the mirror program, DFHMIRS. Defining your own transaction to invoke the mirror program gives you the freedom to specify appropriate values for some other options on the transaction resource definition.

See also the important rules about specifying transid with a DPL_Request on page 134.

Error codes

Most of the exception conditions that are returned on the external CICS interface LINK command are the same as for the CICS-to-CICS distributed program link command. Those that are the same, and their corresponding numeric values are as follows:

LENGERR
22

PGMIDERR
27

SYSIDERR
53

NOTAUTH
70

TERMERR
81

ROLLEDBACK
82

These exception condition codes are returned in the RESP field.

RESP and RESP2

References to the RESP and RESP2 fields in this section are to the first two fields of the RETCODE parameter.

The exception conditions that are specific to the external CICS interface are as follows:

- The RESP2 values on the error condition LENGERR is specific to the external CICS interface.
- The exception conditions WARNING and LINKERR are specific to the external CICS interface.

The WARNING and LINKERR exceptions are a result of responses to individual EXCI calls issued by the external CICS interface in response to an EXEC CICS LINK command. These WARNING and LINKERR exceptions correspond to EXCI call responses as follows:

WARNING (RESP value 4)

This is returned when the EXCI module handling the EXEC CICS LINK request receives a USER_ERROR or SYSTEM_ERROR response to a Close_Pipe or Deallocate_Pipe request issued on behalf of an EXEC CICS LINK command. The RESP value is set to WARNING because the DPL request to CICS completed successfully, but an error occurred in subsequent processing.

The RESP2 field is set to the EXCI reason code, which gives more information about the error.

LINKERR (RESP value 88)

This is returned when the EXCI module handling the EXEC CICS LINK request receives a RETRYABLE, USER_ERROR, or SYSTEM_ERROR response to an EXCI call issued on behalf of the EXEC CICS LINK command. The DPL request has failed. The RESP2 field is set to the EXCI reason code, which gives more information about the error.

See “Chapter 17. Response and reason codes returned on EXCI calls” on page 211 for descriptions of EXCI reason codes.

Note: The external CICS interface ignores any WARNING conditions that occur in response to EXCI calls it issues on behalf of an EXEC CICS LINK command. It treats the WARNING on an EXCI call as a good response and continues normally. If no other errors occur, the EXEC CICS command completes with a zero response in the EXEC_RESP field.

Retries on an EXEC CICS LINK command

If the external CICS interface receives a RETRYABLE response on an EXCI call that it makes on behalf of an EXEC CICS LINK command, it automatically retries the EXEC CICS LINK command up to five times, providing more serious errors do not occur. If the RETRYABLE response is still received after the fifth retry, the RESP field is set to LINKERR, and the reason returned on the EXCI CALL request that causes the exception is returned in the RESP2 field.

The external CICS interface retries the EXEC CICS LINK command by first closing and deallocating the pipe, then reissuing the six EXCI CALL commands. During Allocate_Pipe processing, the EXCI CALL interface calls the user-replaceable module, DFHXCURM, to give you the opportunity to change the APPLID of the CICS system to which the request has been sent. See “Chapter 12. The EXCI user-replaceable module” on page 165 for details of DFHXCURM.

Table 15 lists all the exception conditions and RESP2 values that are specific to the EXEC CICS LINK command for the external CICS interface.

Table 15. Exceptional conditions. RESP and RESP2 values returned from the EXEC API.

Condition (RESP)	RESP2	Meaning
LENGERR (22)	22	COMMAREA length greater than 32763 bytes specified
	23	COMMAREA specified but no LENGTH parameter specified
WARNING (4)	401	Invalid *call_type* parameter value specified on Close_Pipe or Deallocate_Pipe call
	402	Invalid *version_number* parameter specified on Close_Pipe or Deallocate_Pipe call
	404	Invalid *user_token* specified on Close_Pipe or Deallocate_Pipe call
	405	A Deallocate_Pipe call has been issued against a pipe that is not yet closed
	418	An invalid pipe token has been issued on a Close_Pipe or Deallocate_Pipe call
	421	A Close_Pipe or Deallocate_Pipe command has been issued under an IRB
	610	There has been a CICS IRP logoff failure on a Deallocate_Pipe call
	611	There has been a CICS IRC disconnect failure on a Close_Pipe call
	622	There has been an MVS ESTAE setup failure on a Close_Pipe or Deallocate_Pipe call
	623	A program check on a Close_Pipe or Deallocate_Pipe call has caused the ESTAE to be invoked
LINKERR (88)	201	Command has been issued on an MVS image which has had no IRC activity since the previous IPL
	202	There are no available sessions
	203	CICS has not yet been brought up, or (2) has not yet opened IRC, or (3) no generic connection is installed, or (4) no specific connection is installed with the required netname.
	204	An EXEC CICS LINK command without the SYNCONRETURN option has been issued specifying a CICS system on a different MVS image.
	205	An EXEC CICS LINK command without the SYNCONRETURN option has been issued when RRS is not available
	401	Invalid parameter

Table 15. Exceptional conditions (continued). RESP and RESP2 values returned from the EXEC API.

Condition (RESP)	RESP2	Meaning
	402	Invalid version number
	403	User name is all blanks
	404	Invalid address in user token
	405	Command has been issued against a pipe that is not closed
	406	Command has been issued against a pipe that is not open
	407	Userid of all blanks has been passed
	408	Error in UOWID parameter
LINKERR (88)	409	Transid consisting of all blanks or zero has been passed
	410	Load of message module, DFHMEBM, failed
	411	Load of message module, DFHMET4E, failed
	412	Load of DFHXCURM failed
	413	Load of DFHXCTRA failed
	414	If run as a CICS-to-CICS linked-to program, this server program would have resulted in an error with an appropriate message sent to the terminal. Running the program as an EXCI server program returns the message addressed by the MSGPTR field of the RETCODE area.
	415	Target connection is an MRO connection, not an EXCI connection
	416	Command has been issued against a CICS region running under a release of CICS earlier than CICS for MVS/ESA 4.1
	417	Command has been issued against a pipe in the MUST CLOSE state. Further EXCI EXEC CICS LINK commands will have unpredictable results and are, therefore, not permitted
	418	Pipe_token does not address an XCPIPE control block, or there is a mismatch between user_token and pipe_token
	419	CICS runs, or did run, under the TCB that this command is attempting to use. This is not permitted and the command fails
	420	Load of DFHXCOPT failed
	421	The command has been issued under an MVS IRB, which is not permitted
	422	The server has abended
	423	Surrogate user check failed
	424	An EXEC CICS LINK command without the SYNCONRETURN option has been issued on a system that does not support RRMS
	425	A DPL request omitted the SYNCONRETURN option, but specified a value of UOWID.
	601	A GETMAIN of working storage failed. This error leads to user abend 408
	602	A GETMAIN failed. This error leads to user abend 403.
	603	A GETMAIN failed. This error leads to user abend 410
	604	A GETMAIN failed
	605	A GETMAIN for the VERIFY block failed. This error leads to user abend 409.
	606	An SSI verify request (to obtain CICS SVC instruction) failed. This error leads to user abend 405.
	607	An SVC call failed. This error leads to user abend 406.
	608	Logon to IRP failed
	609	Connect to IRP failed
	610	Disconnect from IRP failed
	611	Logoff from IRP failed
	612	Invalid data input to transformer_1
	613	Invalid data input to transformer_4

Table 15. Exceptional conditions (continued). RESP and RESP2 values returned from the EXEC API.

Condition (RESP)	RESP2	Meaning
LINKERR (88)	614	CICS has responded but has not sent any data
	615	CICS cannot satisfy the request
	616	IRP_SWITCH_PULL request (to read data sent from CICS into a larger input/output area) has failed
	617	A GETMAIN for a larger input/output area failed
	619	IRP has had a problem with the input/output area passed from the client program
	620	IRP has disconnected from EXCI
	621	A DISCONNECT command is issued in an error situation following an IRP CONNECT. The DISCONNECT has failed, indicating a serious error.
	622	XCPRH ESTAE setup command failed This error leads to user abend 402.
	623	XCPRH ESTAE invoked due to program check during the processing of this command. ESTAE attempts backout and takes a SYSMDUMP. Further requests are permitted although the pipe is now in a MUST CLOSE state.
	624	The DPL request has been passed to CICS but the time specified in DFHXCOPT has been exceeded. The request is aborted.
	625	An MVS STIMERM macro call failed
	626	An MVS STIMERM CANCEL request failed
	627	The CICS SVC is at the incorrect level. This error leads to user abend 407.
	628	DFHIRP is at the incorrect level
	630	An unexpected return code was received from RRMS when processing an EXEC CICS LINK command without the SYNCONRETURN option .
	631	An unexpected error was encountered when processing an EXEC CICS LINK command without the SYNCONRETURN option.
	632	A GETMAIN for DFHXCGUR’s working storage failed while processing an EXEC CICS LINK command without the SYNCONRETURN option .
	903	AN XCEIP ESTAE setup command failed
	904	The server program abended with the abend code in the ABCODE field of the RETCODE area
	905	An XCEIP ESTAE invoked

See “Return codes” on page 145 for details of the various copybooks that contain full details of all response and reason codes, including equated values.

Note: All numeric response and reason code values are shown in decimal.

Translation required for EXEC CICS LINK command

Application programs that use the EXEC CICS LINK form of the external CICS interface command must translate their programs before assembly or compilation. You do this using the version of the CICS translator that is appropriate for the language of your client program, specifying the translator option EXCI.

The translator option EXCI is mutually exclusive with the CICS and DLI options.

For more information about translating programs that contain EXEC CICS commands, see the *CICS Application Programming Guide*.

For information about compiling and link-editing external CICS interface client programs, see page 173.

Chapter 11. Defining connections to CICS

Connections between an EXCI client program and a CICS region require connection definitions in the CICS region. You define these using the CONNECTION and the SESSIONS resource definition facilities provided by CICS.

The following options are provided specifically for the external CICS interface:

- CONNTYPE on the CONNECTION resource definition
- EXCI on the PROTOCOL attribute of the CONNECTION and SESSIONS resource definitions.

The following topics are covered in this chapter:

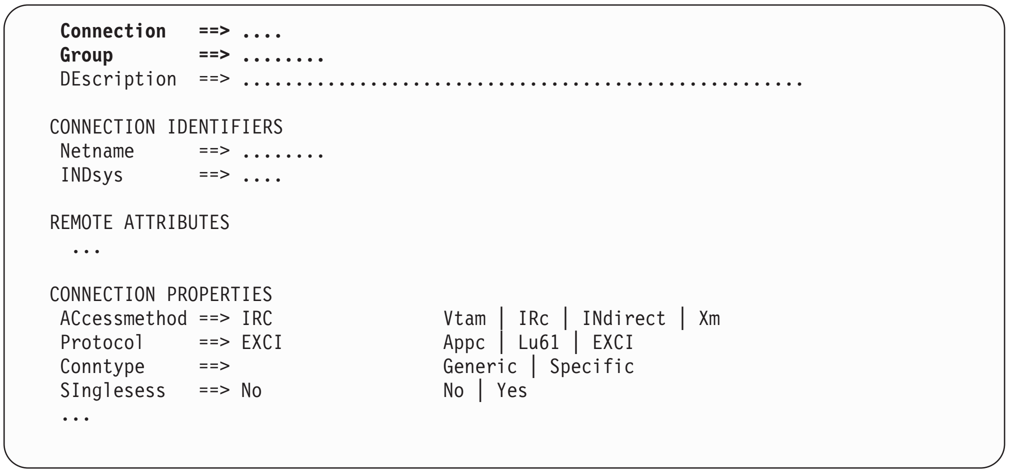
- “CONNECTION resource definition”
- “SESSIONS resource definitions for EXCI connections” on page 161
- “Inquiring on the state of EXCI connections” on page 164

CONNECTION resource definition

The EXCI option is provided on the PROTOCOL attribute of the CONNECTION resource definition to indicate that the connection is for use by an MVS program using the external CICS interface.

The CONNTYPE attribute is provided on the CONNECTION resource definition. For EXCI connections, this indicates whether the connection is generic or specific. It is not to be used for any protocol other than the external CICS interface.

The following figure illustrates the DEFINE CONNECTION panel for defining a connection using RDO. You can also use the DEFINE command with the DFHCSDUP batch utility program to define the connection. For full details of defining resources using RDO or the batch utility, see the*CICS Resource Definition Guide*.



```
 Connection   ==> ....
 Group        ==> ........
 DEscription  ==> .....................................................

CONNECTION IDENTIFIERS
 Netname      ==> ........
 INDsys       ==> ....

REMOTE ATTRIBUTES
  ...

CONNECTION PROPERTIES
 ACcessmethod ==> IRC               Vtam | IRc | INdirect | Xm
 Protocol     ==> EXCI              Appc | Lu61 | EXCI
 Conntype     ==>                   Generic | Specific
 SInglesess   ==> No                No | Yes
 ...
```

Figure 32. The DEFINE panel for CONNECTION

© Copyright IBM Corp. 1994, 1998

CONNTYPE({SPECIFIC|GENERIC})
For external CICS interface connections, indicates the nature of the connection.

SPECIFIC
The connection is for communication from a non-CICS client program to the CICS region, and is specific. A specific connection is an MRO link with one or more sessions dedicated to a single user in a client program.

Note: A *user* is a program that has issued an Initialize_User request (or for which an Initialize_User request has been issued), with a unique name per TCB. For example:

- A simple client program running under MVS can be a single user of the external CICS interface.
- A client program running under MVS can open several pipes and issue external CICS interface calls over them sequentially, on behalf of different vendor packages. In this case, from the viewpoint of the client program, each of the packages is a user, identified by a unique user name. Thus a single client program can operate on behalf of multiple users.
- A program running under MVS can attach several TCBs, under each of which a vendor package issues external CICS interface calls on its own behalf. Each package is a client program in its own right, and runs under its own TCB. Each is also a user, with a unique user name.

For a specific connection, NETNAME is mandatory.

GENERIC
The connection is for communication from a non-CICS client program to the CICS system, and is generic. A generic connection is an MRO link with a number of sessions to be shared by multiple EXCI users. For a generic connection you cannot specify the NETNAME attribute.

Note: You must install only one generic EXCI connection in a CICS region.

NETNAME
For an external CICS interface connection, NETNAME corresponds to the name of the user of a specific pipe, as specified on the *user_name* parameter of an INITIALISE_USER call.

For an external CICS interface specific pipe, you must specify a NETNAME.

For external CICS interface generic pipes, you must leave NETNAME blank.

PROTOCOL({APPC|LU61|EXCI|blank})
The type of protocol that is to be used for the link.

blank
For MRO between CICS regions. You must leave the PROTOCOL blank for MRO, and on the SESSIONS definition you must specify LU6.1 as the PROTOCOL.

APPC (LUTYPE6.2 protocol)
Advanced program-to-program communication, or APPC protocol. This is the default value for ACCESSMETHOD(VTAM). Specify this for CICS-CICS ISC.

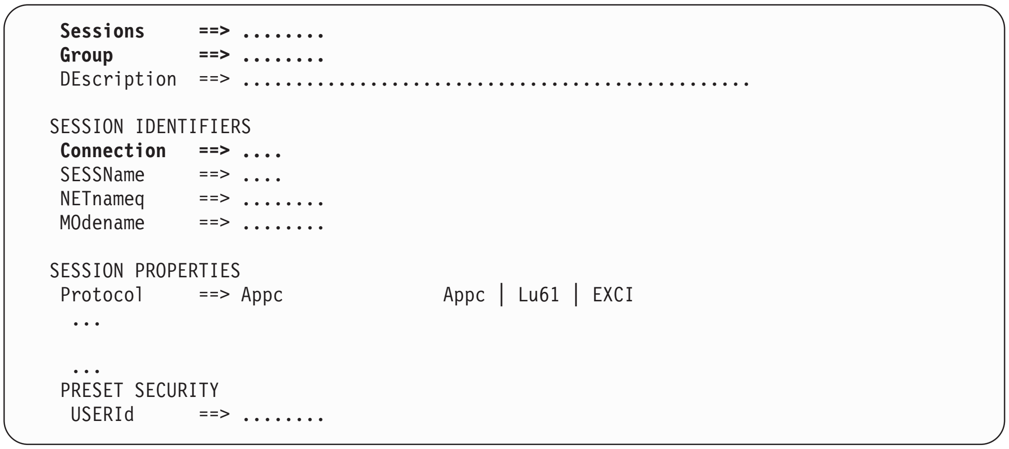
LU61
LUTYPE6.1 protocol. Specify this for CICS-CICS ISC or CICS-IMS ISC, but not for MRO.

EXCI
The external CICS interface. Specify this to indicate that this connection is for use by a non-CICS client program using the external CICS interface.

If you specify PROTOCOL(EXCI), you must also specify ACCESSMETHOD(IRC). EXCI is implemented in MRO using the CICS interregion communication program, DFHIRP, and cannot use MRO links that use MVS cross-memory services (XM). EXCI can use also XCF MRO links, which also work through DFHIRP.

SESSIONS resource definitions for EXCI connections

You indicate on the PROTOCOL attribute of the SESSIONS resource definition whether the sessions allocated on the MRO connection are for use by the external CICS interface. The following figure illustrates the DEFINE SESSIONS panel for defining sessions using RDO. You can also use the DEFINE command with the DFHCSDUP batch utility program to define the required sessions.



```
 Sessions     ==> ........
 Group        ==> ........
 DEscription  ==> ................................................

SESSION IDENTIFIERS
 Connection   ==> ....
 SESSName     ==> ....
 NETnameq     ==> ........
 MOdename     ==> ........

SESSION PROPERTIES
 Protocol     ==> Appc               Appc | Lu61 | EXCI
  ...

  ...
 PRESET SECURITY
  USERId      ==> ........
```

Figure 33. The DEFINE panel for SESSIONS

PROTOCOL({APPC|LU61|EXCI})
Indicates the type of protocol that is to be used for an intercommunication link (ISC or MRO).

APPC (LUTYPE6.2)
Advanced program-to-program communication (APPC) protocol. Specify this for CICS-CICS ISC.

LU61
LUTYPE6.1 protocol. Specify this for CICS-CICS ISC, for CICS-IMS, or for MRO.

EXCI
The external CICS interface. Specify this to indicate that the sessions are for use by a non-CICS client program using the external CICS interface. If you specify EXCI, you must leave SENDCOUNT blank.

RECEIVECOUNT({blank|*number*})

The number of MRO, LUTYPE6.1, or EXCI sessions that usually receive before sending.

For MRO, receive sessions can only receive before sending.

blank

These sessions can send only; there are no receive sessions.

number

Specifies the number of receive sessions on connections that specify blank, LU61, or EXCI on the protocol parameter of the CONNECTION definition. CICS uses the number to generate the last two or three characters of the session names (see RECEIVEPFX for details).

If you are using the default receive prefix (<), or your own 1-character prefix, specify a number in the range 1 through 999.

If you specify a 2-character prefix, the number is restricted to the range 1 through 99.

Except for external CICS interface (EXCI) connections, the RECEIVECOUNT in this system should equal SENDCOUNT in the other system.

RECEIVEPFX(<|*prefix*)

Specifies a 1- or 2-character prefix that CICS is to use as the first 1 or 2 characters of the receive session names (the names of the terminal control table terminal entries (TCTTEs) for the sessions).

Prefixes must not cause a conflict with an existing connection or terminal name.

< (MRO and EXCI sessions)

For MRO sessions, if you do not specify your own receive prefix, CICS enforces the default prefix—the less-than symbol (<), which is used in conjunction with the receive count to generate receive session names.

CICS creates the last three characters of the session names from the alphanumeric characters A through Z, and 1 through 9. These 3-character identifiers begin with the letters AAA, and continue in ascending sequence until the number of session entries reaches the limit set by the RECEIVECOUNT value. Note that receive session names are generated *after* the send sessions, and they follow in the same sequence.

For example, if the last session name generated for the send sessions is <AAJ, using the default prefix (<) CICS generates the receive session names as <AAK, <AAL, <AAM, and so on. (This method of generation of session identifiers is the same as for APPC sessions, except for the initial prefix symbol.)

Note: If you specify your own prefix, CICS generates the session names as in earlier releases, which is the same as for LUTYPE6.1 sessions.

prefix **(LUTYPE6.1 sessions)**

If the sessions are on LUTYPE6.1 ISC connections, you must specify a 1- or 2-character prefix. Do not use the default < symbol for LUTYPE6.1 sessions.

For LUTYPE6.1 sessions (and MRO if you specify your own 1- or 2-character prefix) CICS generates session names by appending a number to the prefix, either in the range 1 through 99, or 1 through 999. The number begins with 1 and is incremented by 1 until the specified RECEIVECOUNT is reached.

SENDCOUNT(blank|*number*)

The number of MRO or LUTYPE6.1 sessions that usually send before receiving.

For MRO, send sessions must send before they can receive.

blank

These sessions can receive only; there are no send sessions.

You must leave this field blank when the sessions are on an external CICS interface (EXCI) connection.

number

Specifies the number of send sessions on connections that specify blank or LU61 on the protocol parameter of the CONNECTION definition. CICS uses the number to generate the last two or three characters of the session names (see SENDPFX for details).

If you are using the default send prefix (>), or your own 1-character prefix, specify a number in the range 1 through 999.

If you specify a 2-character prefix, the number is restricted to the range 1 through 99.

Except for external CICS interface (EXCI) connections the SENDCOUNT in the sending system should equal RECEIVECOUNT in the receiving system.

SENDPFX(>|*prefix*)

Specifies a 1- or 2-character prefix that CICS is to use as the first 1 or 2 characters of the send session names (the names of the terminal control table terminal entries (TCTTEs) for the sessions).

Prefixes must not cause a conflict with an existing connection or terminal name.

> (MRO sessions)

For MRO sessions, if you do not specify your own send prefix, CICS enforces the default prefix—the greater-than symbol (>), which is used in conjunction with the send count to generate send session names.

CICS creates the last three characters of the session names from the alphanumeric characters A through Z, and 1 through 9. These 3-character identifiers begin with the letters AAA, and continue in ascending sequence until the number of session entries reaches the limit set by the SENDCOUNT value.

For example, using the default prefix (>) CICS generates session names as >AAA, >AAB, >AAC, and so on. (This method of generation of session identifiers is the same as for APPC sessions, except for the initial symbol.)

Note: If you specify your own prefix, CICS generates the session names as in earlier releases, which is the same as for LUTYPE6.1 sessions.

prefix **(for LUTYPE6.1 sessions)**

If the sessions are on LUTYPE6.1 ISC connections, you must specify a 1- or 2-character prefix. Do not use the default > symbol for LUTYPE6.1 sessions.

For LUTYPE6.1 sessions (and MRO if you specify your own 1- or 2-character prefix) CICS generates session names by appending a number to the prefix, either in the range 1 through 99, or 1 through 999. The number begins with 1 and is incremented by 1 until the specified SENDCOUNT is reached.

USERID(*userid***)**

The preset user identifier to be used for link security checking.

If you do not specify a preset userid for link security, CICS uses the userid passed from the remote user as the userid for link security, which in the case of an external CICS interface link is the client userid.

Inquiring on the state of EXCI connections

If you have access, through a CICS terminal, to the CICS server region, you can inquire about batch jobs that are running a client application program, and which are using the external CICS interface to link to a server program in CICS.

To obtain this information about batch jobs linked to CICS through MRO, you use the CEMT INQUIRE EXCI command. This command enables you to identify the names of external CICS interface batch jobs currently connected to CICS through the interregion communication (IRC) facility.

CICS returns job identifications in the form:

```
jobname.stepname.procname - mvsid
```

Either `stepname`, or `procname`, or both may not be present, indicated by the periods (.) being adjacent to one another.

The `mvsid` identifies the MVS system on which the job is running. If XCF/MRO is in use, the job can reside on a different MVS image from that on which CICS is running.

Information about jobs using the external CICS interface is available only when the job has issued at least one DPL request. A non-zero task number indicates that a DPL request is currently active. A zero task number indicates an external CICS interface session is still open (connected) for that job, although no DPL request is currently active.

See the *CICS Supplied Transactions* manual for more information about the CEMT command.

Chapter 12. The EXCI user-replaceable module

This chapter contains Product-sensitive Programming Interface information.

The external CICS interface provides a user-replaceable module, DFHXCURM. The load module is supplied in CICSTS13.CICS.SDFHEXCI, and the source in CICSTS13.CICS.SDFHSAMP.

DFHXCURM is invoked in the non-CICS region during the processing of Allocate_Pipe commands, and after the occurrence of any retryable error. The retryable responses are:

- The target CICS region is not available
- There are no pipes available on the target CICS region
- There has been no IRC activity since the MVS IPL.

As supplied, DFHXCURM is effectively a dummy program because of a branch instruction that bypasses the sample logic and returns control to the external CICS interface caller. To use the sample logic, remove the branch instruction and assemble and link-edit the module. Customizing DFHXCURM allows you to do the following:

- When invoked during Allocate_Pipe processing, you can change the specified CICS APPLID, in order to route the request to another CICS system.
- When invoked after a retryable error you can store information regarding CICS availability. You can then use this information on the next invocation of DFHXCURM for Allocate_Pipe processing, so that you can decide to which CICS system to route the request.

DFHXCURM is called using standard MVS register conventions, with register 1 containing the address of the parameter list, and register 14 the return address of the caller. The parameters addressed by register 1 are mapped in the EXCI_URM_PARMS DSECT, which is contained within the DFHXCPLD copybook. The parameters passed to DFHXCURM are as follows:

URMINV

The address of a fullword that contains the reason for the invocation of DFHXCURM, defined by the following equates:

```
URM_ALLOCATE     EQU 1  This invocation is for an Allocate_Pipe
URM_NO_CICS      EQU 2  The target CICS region is not available
URM_NO_PIPE      EQU 3  There are no pipes available
URM_NO_CICS_IRC  EQU 4  There has been no IRC activity since the MVS IPL
```

URMCICS

The address of an 8-byte area that contains the generic APPLID of the target CICS system, as specified on the *CICS_applid* parameter of the Allocate_Pipe command, or on the APPLID parameter of the EXEC CICS LINK command.

When specified by one of these commands, you can change the APPLID to that of a different target CICS region.

If the *CICS_applid* parameter is omitted from the Allocate_Pipe call, or APPLID is omitted from the EXEC CICS LINK command, the field addressed by this parameter contains 8 blanks. In this case, you must specify an APPLID in DFHXCURM before returning control to the caller.

URMAPPL

The address of an 8-byte area that contains the client program's user name as

© Copyright IBM Corp. 1994, 1998

specified on the *my_name* parameter of the Initialize_User command. Note that if DFHXCURM is invoked for an EXEC CICS LINK command, this name is always set to DFHXCEIP.

URMPROG
The address of an 8-byte area that contains the name of the target program (if available). This name is available only if DFHXCURM is invoked for an EXEC CICS LINK command. For an external CICS interface Allocate_Pipe command, the program name is not known until the DPL call is issued.

URMOPTS
The address of a 1-byte area that contains the allocate options, which can be X'00' or X'80', as specified on the *allocate_opts* parameter. This address is valid for an Allocate_Pipe request only.

URMANCH
The address of a 4-byte area that is provided for use by DFHXCURM only. A typical use for this is to store a global anchor address of an area used to save information across a number of invocations of DFHXCURM. For example, you can GETMAIN the necessary storage and save the address in the 4-byte area addressed by this parameter. The initial value of the 4-byte area is set to zero.

Chapter 13. External CICS interface options table, DFHXCOPT

The EXCI options table, generated by the DFHXCOPT macro, enables you to specify a number of parameters that are required by the external CICS interface.

CICS provides the default DFHXCOPT table in source form, which you can tailor to your own requirements. The source of the default table is supplied in CICSTS13.CICS.SDFHSAMP, and the load module is in CICSTS13.CICS.SDFHEXCI.

You assemble and link-edit the modified DFHXCOPT table into a suitable library in the STEPLIB concatenation of the job that runs the MVS client program. You can use your own version of the CICS DFHAUPLE procedure to assemble and link-edit your customized options table. The DFHAUPLE procedure is supplied in CICSTS13.CICS.SDFHINST. Unlike the tables you specify for CICS regions, the DFHXCOPT table cannot be suffixed, and the external CICS interface component loads the first table of this name that it finds in the STEPLIB concatenation.

Table 16 shows the format of the DFHXCOPT macro and its parameters.

Table 16. The DFHXCOPT macro parameters

	DFHXCO	TYPE={**CSECT**|DSECT} [,CICSSVC={**0**|*number*}] [,CONFDATA={**SHOW**|HIDETC}] [,DURETRY={**30**|*number-of-seconds*}] [,GTF={**OFF**|ON}] [,MSGCASE={**MIXED**|UPPER}] [,SURROGCHK={**YES**|NO}] [,TIMEOUT={**0**|*number*}] [,TRACE={**OFF**|1|2}] [,TRACESZE={**16**|*number-of-kilobytes*}] [,TRAP={**OFF**|ON}] You must terminate your parameters with the following END statement.
	END	DFHXCOPT

TYPE={CSECT|DSECT}
Indicates the type of table to be generated.

CSECT
A regular control section that is normally used.

DSECT
A dummy control section.

CICSSVC={0|*number*}
Specifies the CICS type 3 SVC number being used for MRO communication.

The external CICS interface must use the same SVC number that is in use by the CICS MRO regions that reside in the MVS image in which the client program is running.

If you do not specify a specific CICS SVC number, the external CICS interface determines the SVC in use for MRO by means of an MVS VERIFY command.

© Copyright IBM Corp. 1994, 1998

0 Specify zero to indicate that the external CICS interface is to obtain the CICS SVC number from MVS. This is the default.

You should only specify 0 when you are sure that at least one CICS region has logged on to DFHIRP during the life of the MVS IPL.

number
Specify the CICS SVC number, in the range 200—255, that is in use for CICS interregion communications. This must be the SVC number that is installed in the MVS image in which the client program is running (the local MVS).

If no MRO CICS regions have ever logged on to DFHIRP in the local MVS during the life of the IPL, you must specify the SVC number. If you allow this parameter to default, and the external CICS interface requests the SVC from MVS, the request will fail if no CICS region has logged on to DFHIRP.

This parameter is required in those MVS images that do not run any CICS regions, and the client program is issuing DPL requests to a server CICS region that resides in another MVS. In these circumstances, the client program logs on to the local DFHIRP using the locally defined SVC, and communicates with the remote CICS region using XCF/MRO.

Note: All CICS regions using MRO within the same MVS image must use the highest level of both DFHIRP and the CICS SVC, DFHCSVC. If your MRO CICSplex consists of CICS regions at different release levels, the DFHIRP and DFHCSVC installed in the LPA must be from highest release level of CICS within the CICSplex.

MVS client programs using the external CICS interface can communicate only with server regions running under CICS for MVS/ESA 4.1 or later.

CONFDATA={SHOW|HIDETC}
Code this parameter to indicate whether the external CICS interface is to suppress (hide) user data that might otherwise appear in EXCI trace entries output to GTF or in EXCI dumps. This option applies to the tracing of the COMMAREA flowing between the EXCI client program and the CICS server program.

SHOW
Data suppression is not in effect. User data is traced.

HIDETC
This specifies that you want EXCI to ‘hide’ user COMMAREA data from trace entries. Instead of the COMMAREA data, the trace entry contains a character string stating that the data has been suppressed.

DURETRY={30|*number-of-seconds*|0}
Specifies the total time, in seconds, that the external CICS interface is to continue trying to obtain an MVS system dump using the SDUMP macro.

DURETRY allows you to control whether, and for how long, the external CICS interface is to reissue the SDUMP if another address space in the same MVS system is already taking an SDUMP when the external CICS interface issues an SDUMP request.

In the event of an SDUMP failure, the external CICS interface reacts as follows:

- If MVS is already taking an SDUMP for another address space, and the DURETRY parameter is nonzero, the external CICS interface issues an MVS STIMERM macro to wait for five seconds, before retrying the SDUMP macro. The external CICS interface issues a message to say that it will retry the SDUMP every five seconds until the DURETRY time limit.
- If the SDUMP fails for any other reason such as:
  - There are no SYS1.DUMP data sets available, or
  - There are I/O errors preventing completion of the dump, or
  - The DURETRY limit expires while retrying SDUMP

  the external CICS interface issues a message to inform you that the SDUMP has failed, giving the reason why.

30 30 seconds allows the external CICS interface to retry up to six times (once every five seconds).

number-of-seconds
Code the total number of seconds (up to 32767 seconds) during which you want the external CICS interface to continue retrying the SDUMP macro. The external CICS interface retries the SDUMP, once every five seconds, until successful or until retries have been made over a period equal to or greater than the DURETRY value.

0 Code a zero value if you do not want CICS to retry the SDUMP.

GTF={OFF|ON}
Specifies whether all trace entries normally written to the external CICS interface internal trace table are also to be written to an MVS generalized trace facility (GTF) data set (if GTF tracing is active).

OFF
Code this if trace entries are not to be written to GTF.

ON
Code this if trace entries are to be written to GTF.

MSGCASE={MIXED|UPPER}
Specifies whether the DFHEX*xxxx*messages are to be issued in mixed or uppercase.

MIXED
Code this if messages are to be issued in mixed case.

UPPER
Code this if messages are to be issued in uppercase.

SURROGCHK={YES|NO}
Specifies whether the external CICS interface is to perform surrogate user checks against the client job user id (the user ID under which the EXCI job is running).

YES
The external CICS interface is to perform a surrogate user check to verify that the user ID under which the EXCI client job is running is authorized as a surrogate for the user ID specified on a DPL call. The check is made only when the client user ID is different from the user ID on the DPL call.

The client user ID must be authorized to the appropriate profile in the SURROGAT general resource class. You do this by giving the client user ID

READ authority to a profile named *userid*.DFHEXCI in the SURROGAT general resource class, where *userid* is the user ID specified on the DPL call.

See “Surrogate user checking” on page 189 for an example of how to authorize the batch region user ID as a surrogate user for a DPL user ID.

NO
Surrogate user checks are not to be performed.

TIMEOUT={0|*number*}
Specifies the time interval, in hundredths of a second, during which the external CICS interface waits for a DPL command to complete.

0 Specifies that you do not want any time limit applied, and that the external CICS interface is to wait indefinitely for a DPL command to complete.

number
Specifies the time interval, in hundredths of a second, that the external CICS interface is to wait for a DPL command to complete. The number represents hundredths of a second, from 1 up to a maximum of 2 147 483 647. For example:

6000	Represents a timeout value of one minute
30000	Represents a timeout value of five minutes
60000	Represents a timeout value of ten minutes.

TRACE={OFF|1|2}
Specifies whether you want external CICS interface internal tracing, and at what level.

OFF
External CICS interface internal tracing is not required. However, even with normal tracing switched off, exception trace entries are always written to the internal trace table.

1 Exception and level-1 trace entries are written to the internal trace table.

2 Exception, level-1, and level-2 trace entries are written to the internal trace table.

TRACESZE={16|*number-of-kilobytes*}
Specifies the size in kilobytes of the internal trace table for use by the external CICS interface. This table is allocated in virtual storage above the 16MB line. You should ensure that there is enough virtual storage for the trace table by specifying a large enough region size on the MVS REGION parameter.

16 16KB is the default size of the trace table, and also the minimum size.

number-of-kilobytes
The number of kilobytes of storage to be allocated for the internal trace table, in the range 16KB through 1 048 576KB. Subpool 1 is used for the trace table storage, which exists for the duration of the jobstep TCB. The table is page-aligned and occupies a whole number of pages. If the value specified is not a multiple of the page size (4KB), it is rounded up to the next multiple of 4KB.

TRAP={OFF|ON}
Specifies whether the service trap module, DFHXCTRA, is to be used. DFHXCTRA is supplied as a user-replaceable module, in which IBM service personnel can add code to trap errors.

OFF
Code this if you do not want to use DFHXCTRA.

ON
Code this if you require DFHXCTRA.

Chapter 14. Compiling and link-editing external CICS interface client programs

This chapter discusses the following topics:

- The external CICS interface stub, DFHXCSTB
- The CICS-supplied procedures for the external CICS interface
- Language considerations
- Sample application programs
- Job control language to run an EXCI client program.

The external CICS interface stub, DFHXCSTB

All programs that use the external CICS interface to pass DPL requests to a CICS server region must include the CICS-supplied program stub, DFHXCSTB.

The stub intercepts all external CICS interface commands, whether they are EXCI CALL interface commands, or EXEC CICS LINK commands, and ensures they are passed to the appropriate external CICS interface routine for processing.

DFHXCSTB is a common stub, designed for inclusion in programs written in all the supported languages. It is supplied in the CICSTS13.CICS.SDFHEXCI library.

Note: The CICSTS13.CICS.SDFHEXCI also contains entries for DFHXCIE and DFHXCIS, which are aliases for DFHXCSTB.

To help you ensure that the stub is included, CICS provides a number of procedures, one for each language, which you can use for translating, compiling, and link-editing.

The required linkage editor modes

You must specify AMODE(31) for your EXCI client program.

The CICS-supplied procedures for compiling and link-editing client programs include the following parameters on the PARM statement of the linkage editor job step:

```
LNKPARM='AMODE(31),LIST,XREF'
```

The CICS-supplied procedures for the external CICS interface

CICS provides seven procedures to enable you to translate, compile, and link-edit your client programs. Four of these are for use with specific language compilers or assembler, the other three being for use with Language Environment/370. These procedures, with the four language-specific procedures shown first, are:

DFHEXTAL
The assembler procedure for assembler versions of client programs

DFHEXTDL
The C procedure for C versions of client programs

DFHEXTPL
The PL/I procedure for PL/I versions of client programs

© Copyright IBM Corp. 1994, 1998

DFHEXTVL
The COBOL procedure for VS COBOL II versions of client programs.

DFHYXTDL
The procedure for C versions of client programs running under Language Environment/370

DFHYXTPL
The procedure for PL/I versions of client programs running under Language Environment/370

DFHYXTVL
The procedure for VS COBOL II versions of client programs running under Language Environment/370.

To ensure that the EXCI stub is included with your client program, all these procedures include a step, COPYLINK, that unloads the stub into a temporary data set defined with a block length suitable for the linkage-editor. This temporary data set is then concatenated with the temporary data set containing your object program on the SYSLIN DD statement in the LKED step.

These procedures are supplied in the CICSTS13.CICS.SDFHPROC library. You are recommended to copy these to SYS1.PROCLIB or another suitable procedure library.

Language considerations

There are some language requirements that apply to writing an MVS client program that uses the external CICS interface. These affect programs written in PL/I and C. Also, for all languages, consider how you handle return codes before terminating your MVS client program.

PL/I considerations

PL/I programs written to the external CICS interface must provide their parameters on the CALL to DFHXCIS in the form of an assembler-style parameter list.

The EXCI copybook for PL/I, DFHXCPLL, contains the necessary definition of the DFHXCIS entry point, as follows:

```
 DCL DFHXCIS  ENTRY        OPTIONS(INTER ASSEMBLER);
```

The same rule applies for the EXCI LINK command, and in this case the CICS translator ensures that the correct parameter list is built.

For an example of an EXCI client program written in PL/I, see the source of the sample program, DFH$PXCC.

C considerations

C programs written to the external CICS interface must provide their parameters on the CALL to DFHXCIS in the form of an assembler-style parameter list. You ensure this by declaring the entry point to DFHXCIS with OS LINKAGE.

The EXCI copybook for PL/I, DFHXCPLH, contains the necessary definition of the DFHXCIS entry point, as follows:

```
  #pragma linkage(dfhxcis,OS)
```

The same rule applies for the EXCI LINK command, and in this case the CICS translator ensures that the correct parameter list is built.

For an example of an EXCI client program written in C, see the source of the sample program, DFH$DXCC.

Setting the return code (R15) at termination

The external CICS interface does not clear register 15 at termination, regardless of whether your client program executes normally or not. Therefore, even if your MVS client program terminates normally after successfully using the external CICS interface, the job step could end with an undefined return code.

To ensure a meaningful return code is given at termination, set the job step return code before terminating your program. The sample client programs illustrate how you can do this, using the saved response code from last call to the external CICS interface. For example, the COBOL sample DFH0CXCC program moves SAVED-RESPONSE to special register RETURN-CODE before terminating.

Sample application programs

CICS provides a number of sample programs that are designed to help you in writing your own application programs. To help with writing programs that use the external CICS interface, CICS provides sample MVS client programs and a sample CICS server program.

The samples show you how to code client applications that use both the EXCI CALL interface and EXEC CICS LINK command.

Description of the sample applications

The sample external CICS interface programs are included on the CICS Transaction Server for OS/390 distribution tape.

Two sample MVS client programs are supplied. One is provided in assembler language, VS COBOL II, C/370, and PL/I. The other is only provided in assembler. The sample CICS server program is provided in assembler only. Assembler language programs are in source and executable form. COBOL, PL/I, and C/370 programs are provided in source form only. Each version of the client program has basically the same function, but programming methods vary somewhat according to the language used.

The sample programs, shown in Table 17 on page 176, are supplied in source form in CICSTS13.CICS.SDFHSAMP. The sample assembler server program is also supplied in executable form in CICSTS13.CICS.SDFHLOAD. The assembler client program is supplied in CICSTS13.CICS.SDFHEXCI.

Note: The assembler versions of the client program use BSAM, which requires the programs to be link-edited in RMODE(24). The assembler source code includes the required RMODE(24) statement. Normally, EXCI client programs run AMODE(31),RMODE(ANY).

Table 17. The external CICS interface sample programs

Language	Name	Type of program
Assembler	DFH$AXCC	Client program
Assembler	DFH$ATXC	Client program
Assembler	DFH$AXCS	Server program
COBOL	DFH0CXCC	Client program
PL/I	DFH$PXCC	Client program
C/370	DFH$DXCC	Client program

The sample client programs show you how to code a simple MVS client application using the EXCI CALL interface and the EXEC CICS LINK command.

Each version of the client is divided into three separate sections as follows:

1. The first section issues a single EXEC CICS LINK command to inquire on the state of the sample VSAM file, FILEA, in the target CICS system.

   If the file is in a suitable state, processing continues to sections two and three, which together provide complete examples of the use of the EXCI CALL interface.
2. The second section initiates a specific MRO connection to the target CICS system and, once the pipe is open, performs a series of calls that each retrieve a single sequential record from the sample VSAM file, until no more records are available.
3. The third section is a simple routine to close the target sample file once processing of the data is complete. It also terminates the MRO connection now that the link is no longer required.

Some of the parameters used on the EXCI CALL and EXEC CICS LINK commands in the client program need to be tailored for your own target CICS server region. Change these as required, then retranslate, compile (or assemble), and link-edit the program.

The variables and their values specified in the sample programs are given in Table 18.

Table 18. Parameters used in the sample client programs

Variable name in sample program	Default value
TARGET_FILE	FILEA
TARGET_TRANSID	EXCI
TARGET_SYSTEM	DBDCCICS (applid)
TARGET_PROGRAM	DFH$AXCS
TRAGET_USERID	Defaults to batch region’s user ID
APPLICATION	BATCHCLI

The assembler versions of the client programs are supplied pregenerated in an executable form. All versions of the program accept two run-time parameters, as follows:

1. The first (TARGET_SYSTEM) specifies the server region APPLID.

For the pregenerated assembler versions this avoids you having to reassemble the programs to specify the applid of your own CICS server region. You can also use the sample client programs with different CICS regions without needing to modify the programs each time.

2. The second specifies the user ID to be used on the call interface DPL_request.

You specify these positional parameters on the PARM statement, separated by a comma.

Using the COMMAREA in the sample programs

Data is passed between the sample client and server programs using a standard CICS communications area (COMMAREA) for passing data between programs. The definitions of this COMMAREA are identical on each side of the EXCI link to ensure that data is mapped correctly.

The sample client program DFH$AXCC minimizes data transmission by specifying a specific data length to avoid sending the whole COMMAREA over the link. The data length option specifies a binary value that is the length of a contiguous area of storage, from the start of the COMMAREA, to be passed to the server program. This is because the amount of data being passed to the server is small, but the COMMAREA itself is much larger so that the server can return the requested records from FILEA. Specifying a data length when the amount of data being passed is smaller than the COMMAREA improves performance.

| Sample program DFH$ATXC passes the full COMMAREA to the server program
| because it makes changes to the file records and the whole of the changed record
| needs to be passed to the server.

The first and third sections of the sample client programs define the COMMAREA as only 18 bytes (no data is requested from the server in these sections). For the second section, the sample client program defines the COMMAREA as 98 bytes, the extra 80 bytes being required for the sample server program to return a record from FILEA. In all sections, the data length is defined as 18 bytes. The COMMAREA structure is defined in the sample programs as follows:

Bytes	Data type	Field description
0-3	Fullword	Call type code.
4-11	Char(8)	Target file name.
12-17	Char(6)	Ridfield identifier.
18-97	Char(80)	FILEA record data area.

Note that, although the COMMAREA structure is described in both the client and server programs, the actual size of the COMMAREA made available by CICS to the server is determined by the client program. If you modify the sample programs to work with one of your own application programs, make sure you specify a COMMAREA large enough to handle the maximum amount of data the server is expected to return to the client. The server must not attempt to return data that is larger than the COMMAREA specified by the client.

For more information about using a COMMAREA for passing data between CICS programs, see the *CICS Application Programming Guide*.

Installing the EXCI sample definitions

Resource definitions that support the EXCI sample programs are included in the CICS system definition file (CSD) in groups DFH$EXCI and DFH$FILA.

Note that the sample definitions, while included in the CSD, are not included in the IBM-defined group list DFHLIST. Thus, if CICS is initialized with GRPLIST=DFHLIST, you must install the EXCI resource definition groups before using the samples. Alternatively, you can add the sample groups to your startup group list, so that they are installed automatically at system initialization.

The resource definition groups that must be installed are as follows:

DFH$EXCI
This contains definitions for the sample server transaction, server program, EXCI connections, and sessions.

Only one server program is included—in assembler language, called DFH$AXCS.

The sample application is designed to run the transaction EXCI, which is defined to invoke the DFHMIRS mirror program and references profile DFHCICSA. The required transaction definition for EXCI is included in the group.

Sample CONNECTION and SESSIONS definitions for specific and generic connections are included.

Note: Both the generic and specific connection definitions supplied in the sample group DFH$EXCI specify ATTACHSEC(IDENTIFY). This security option causes the server program DFH$EXCS to fail with an ATCY abend if you run the sample programs in an environment that does not have RACF, or an equivalent external security manager (ESM), installed and active.

If you want to run the external CICS interface sample programs without any security active, you must alter the connection resource definitions to specify ATTACHSEC(LOCAL).

DFH$FILA
This contains the definition for the supplied sample VSAM file, FILEA, which is referenced by the EXCI sample programs.

Once these are installed, you must ensure that interregion communication (IRC) is open. If IRC is not opened during CICS initialization, set it open using the CEMT SET IRC OPEN command.

Running the EXCI sample applications

If you want to use the COBOL, PL/I, or C/370 version of the EXCI client program, you must translate, compile, and link-edit the program into a suitable library.

You can use the sample JCL shown in Figure 40 on page 184 as a basis for creating your own batch job to run the client program.

If you use the pregenerated assembler version, to specify the APPLID of your target CICS server region as a parameter on the EXEC statement for the client program, as follows:

```
//*==============================================================*
//ASM       EXEC  PGM=DFH$AXCC,PARM='applid,userid'
```

Where: *applid* is the applid of the CICS server region *userid* is the userid for the DPL_request call. Note: If you omit *applid*, you must keep the comma preceding the userid.

Change PGM=DFH$AXCC to PGM=DFH$ATXC to run the other client sample program.

Results of running the EXCI sample applications

An example of the output produced by successful execution of the pregenerated assembler version of the client program, DFH$AXCC, is shown in Figure 34 on page 180.

If an error occurs while running the application, then, assuming the error is not severe, messages are written to the SYSPRINT output log displaying the reasons and/or return codes that cause processing to be aborted. Several examples of error-invoked output are shown in Figure 35, Figure 36, and Figure 37 on page 181.

```
*===================== EXCI Sample Client Program =============================*
*                                                                              *
*  EXEC Level Processor.                                                       *
*    Setting up the EXEC level call.                                           *
*    The Link Request has successfully completed.                              *
*    Server Response:                                                          *
*      The file is set to a browsable state.                                   *
*                                                                              *
*  CALL Level Processor.                                                       *
*    Initialize_User call complete.                                            *
*    Allocate_Pipe call complete.                                              *
*    Open_Pipe call complete.                                                  *
*    The connection has been successful.                                       *
*      The target file follows:                                                *
*                                                                              *
*=========================== Top of File ======================================*
000102F. ALDSON            WARWICK, ENGLAND    9835618326 11 81$1111.11Y00007300
000104S. BOWLER            LONDON,ENGLAND      1284629326 11 81$0999.99Y00007400
000106B. ADAMS             CROYDON, ENGLAND    1948567326 11 81$0087.71Y00007500
000111GENE BARLOWE         SARATOGA,CALIFORNIA 4612075301 02 74$0111.11Y00007600
000762GEORGE BURROW        SAN JOSE,CALIFORNIA 2231212101 06 74$0000.00Y00007700
000983H. L. L. CALL        WASHINGTON, DC      3451212021 04 75$9999.99Y00007800
003210B.CREPIN             NICE, FRANCE        1234567026 11 81$3349.99Y00008100
003214HUBERT C HERBERT     SUNNYVALE, CAL.     3411212000 06 73$0009.99N00008200
003890PHILIPPE SMITH, JR   NICE, FRANCE        0000000028 05 74$0009.99N00008300
004004STAN SMITH           DUBLIN, IRELAND     7111212102 11 73$1259.99N00008400
004445S. GALSON            SOUTH BEND, S.DAK.  6121212026 11 81$0009.99N00008500
004878D.C. CURRENT         SUNNYVALE, CALIF.   3221212010 06 73$5399.99N00008600
005005J. S. LAVERENCE      SAN FRANCISCO, CA.  0000000101 08 73$0009.99N00008700
005444JEAN LAWRENCE        SARATOGA, CALIF.    6771212020 10 74$0809.99N00008800
005581JOHN ALDEN III       BOSTON, MASS.       4131212011 04 74$0259.99N00008900
006016DR W. T. KAR         NEW DELHI, INDIA    7033121121 05 74$0009.88Y00009000
006670WILLIAM KAPP         NEW YORK, N.Y.      2121212031 01 75$3509.88N00009100
06968D. CONRAD             WARWICK, ENGLAND    5671382126 11 81$0009.88Y00009200
007248B. C. WILLIAMSON     REDWOOD CITY, CALF. 3331212111 10 75$0009.88N00009400
007779MRS. W. WELCH        SAN JOSE, CALIF.    4151212003 01 75$0009.88Y00009500
100000G. NEADS             TORONTO, ONTARIO    0341512126 11 81$0010.00Y00009600
111111C. MEARS             OTTAWA, ONTARIO     5121200326 11 81$0011.00Y00009700
200000A. BONFIELD          GLASCOW, SCOTLAND   6373829026 11 81$0020.00Y00009900
300000K. TRENCHARD         NEW YORK, U.S.      6473980126 11 81$0030.00Y00010000
333333D. MYRING            CARDIFF, WALES      7849302026 11 81$0033.00Y00010100
400000W. TANNER            MILAN, ITALY        2536373826 11 81$0040.00Y00010200
444444A. FISHER            CALGARY, ALBERTA    7788982026 11 81$0044.00Y00010300
500000J. DENFORD           MADRID, SPAIN       4445464026 11 81$0000.00Y00010400
555555C. JARDINE           KINGSTON, N.Y.      3994442026 11 81$0005.00Y00010500
600000F. HUGHES            DUBLIN, IRELAND     1239878026 11 81$0010.00Y00010600
666666A. BROOKMAN          LA HULPE, BRUSSELS  4298384026 11 81$0016.00Y00010700
700000A. MACALLA           DALLAS, TEXAS       5798432026 11 81$0002.00Y00010800
777777D. PRYKE             WILLIAMSBURG, VIRG. 9187613126 11 81$0027.00Y00010900
800000H. BRISTOW           WESTEND, LONDON     2423338926 11 81$0030.00Y00011000
888888B. HOWARD            NORTHAMPTON, ENG.   2369163926 11 81$0038.00Y00011100
900000D. WOODSON           TAMPA, FLA.         3566812026 11 81$0040.00Y00011200
999999R. JACKSON           RALEIGH, N.Y.       8459163926 11 81$0049.00Y00011300
 *=========================== End of File ======================================*
 *                                                                              *
 *    Closing Dpl Request has been attempted.                                   *
 *    Close_Pipe call complete.                                                 *
 *    Deallocate_Pipe call complete.                                            *
 *                                                                              *
 *=================== End of EXCI Sample Client Program ========================*
```

Figure 34. Successful execution

```
*===================== EXCI Sample Client Program =============================*
*                                                                              *
*  EXEC Level Processor.                                                       *
*    Setting up the EXEC level call.                                           *
*    The Link Request has failed.  Return codes are;                           *
*        Resp = 00000088  Resp2 = 00000203  Abend Code:                        *
*    >>>> Aborting further processing <<<<                                     *
*                                                                              *
*=================== End of EXCI Sample Client Program ========================*
```

Figure 35. No CICS return code. The target CICS region specified by the client program is not found, or IRC was not opened.

```
*===================== EXCI Sample Client Program =============================*
*                                                                              *
*  EXEC Level Processor.                                                       *
*    Setting up the EXEC level call.                                           *
*    The Link Request has successfully completed.                              *
*    Server Response:                                                          *
*      The file could not be found.                                            *
*    >>>> Aborting further processing <<<<                                     *
*                                                                              *
*=================== End of EXCI Sample Client Program ========================*
```

Figure 36. No file found. The target file name to the server program was not found on the target CICS system.

```
*===================== EXCI Sample Client Program =============================*
*                                                                              *
*  EXEC Level Processor.                                                       *
*    Setting up the EXEC level call.                                           *
*    The Link Request has failed.  Return codes are;                           *
*        Resp = 00000088  Resp2 = 00000414  Abend Code:                        *
*    A message was received from the target CICS system:                       *
*                                                                              *
 DFHAC2001 04/29/93 16:43:03 IYAHZCAZ Transaction 'BAD_' is unrecognized.  Check
 that the transaction name is correct.
*                                                                              *
*    >>>> Aborting further processing <<<<                                     *
*                                                                              *
*=================== End of EXCI Sample Client Program ========================*
```

Figure 37. Incorrect transaction identifier. The target transid passed in the external CICS interface call is not defined on the target CICS system. Note the message received from the target CICS system.

An example of the output produced by a successful execution of the pregenerated assembler version of the client program DFH$ATXC is shown in Figure 38 on page 182. If an error occurs while running the application, errors are produced as for DFH$AXCC.

After running DFH$ATXC, program DFH$AXCC should be re-run. The output should be as shown in Figure 39 on page 183. The changes up to and including records with RIDFLD values of 500000 were committed using an SRRCMIT call to tell RRS to tell CICS to perform commit processing. Later changes were backed out by issuing an SRRBACK call. This caused RRS to tell CICS to perform a ROLLBACK of these changes.

Clearly, FILEA is changed as a result of running DFH$ATXC. It should be restored to its original state by running the LOADFILE step of DFHDEFDS.

```
 *==================== TEXCI Sample Batch Client Program =======================*
 *
 *  EXEC Level Processor.
 *    Setting up the EXEC level call.
 *    The Link Request has successfully completed.
 *    Server Response:
 *      The file is set to a browsable state.
 *
 *  CALL Level Processor.
 *    Initialise_User call complete.
 *    Allocate_Pipe call complete.
 *    Open_Pipe call complete.
 *    The connection has been successful.
 *      The changed target file follows:
 *
 *=========================== Top of File =======================================*
 000100W. DAVIS            SURREY, ENGLAND     3215677826 11 81$0100.11COMMITTED
 000102F. ALDSON           WARWICK, ENGLAND    9835618326 11 81$1111.11COMMITTED
 000104S. BOWLER           LONDON,ENGLAND      1284629326 11 81$0999.99COMMITTED
 000106B. ADAMS            CROYDON, ENGLAND    1948567326 11 81$0087.71COMMITTED
 000111GENE BARLOWE        SARATOGA,CALIFORNIA 4612075301 02 74$0111.11COMMITTED
 000762GEORGE BURROW       SAN JOSE,CALIFORNIA 2231212101 06 74$0000.00COMMITTED
 000983H. L. L. CALL       WASHINGTON, DC      3451212021 04 75$9999.99COMMITTED
  003210B.CREPIN           NICE, FRANCE        1234567026 11 81$3349.99COMMITTED
 003214HUBERT C HERBERT    SUNNYVALE, CAL.     3411212000 06 73$0009.99COMMITTED
 003890PHILIPPE SMITH, JR  NICE, FRANCE        0000000028 05 74$0009.99COMMITTED
 004004STAN SMITH          DUBLIN, IRELAND     7111212102 11 73$1259.99COMMITTED
 004445S. GALSON           SOUTH BEND, S.DAK.  6121212026 11 81$0009.99COMMITTED
 004878D.C. CURRENT        SUNNYVALE, CALIF.   3221212010 06 73$5399.99COMMITTED
 005005J. S. LAVERENCE     SAN FRANCISCO, CA.  0000000101 08 73$0009.99COMMITTED
 005444JEAN LAWRENCE       SARATOGA, CALIF.    6771212020 10 74$0809.99COMMITTED
 005581JOHN ALDEN III      BOSTON, MASS.       4131212011 04 74$0259.99COMMITTED
 006016DR W. T. KAR        NEW DELHI, INDIA    7033121121 05 74$0009.88COMMITTED
 006670WILLIAM KAPP        NEW YORK, N.Y.      2121212031 01 75$3509.88COMMITTED
 006968D. CONRAD           WARWICK, ENGLAND    5671382126 11 81$0009.88COMMITTED
 007248B. C. WILLIAMSON    REDWOOD CITY, CALF. 3331212111 10 75$0009.88COMMITTED
 007779MRS. W. WELCH       SAN JOSE, CALIF.    4151212003 01 75$0009.88COMMITTED
 100000G. NEADS            TORONTO, ONTARIO    0341512126 11 81$0010.00COMMITTED
 111111C. MEARS            OTTAWA, ONTARIO     5121200326 11 81$0011.00COMMITTED
 200000A. BONFIELD         GLASGOW,  SCOTLAND  6373829026 11 81$0020.00COMMITTED
 300000K. TRENCHARD        NEW YORK, U.S.      6473980126 11 81$0030.00COMMITTED
 333333D. MYRING           CARDIFF, WALES      7849302026 11 81$0033.00COMMITTED
 400000W. TANNER           MILAN, ITALY        2536373826 11 81$0040.00COMMITTED
 444444A. FISHER           CALGARY, ALBERTA    7788982026 11 81$0044.00COMMITTED
 500000J. DENFORD          MADRID, SPAIN       4445464026 11 81$0000.00COMMITTED
 555555C. JARDINE          KINGSTON, N.Y.      3994442026 11 81$0005.00BACKEDOUT
 600000F. HUGHES           DUBLIN, IRELAND     1239878026 11 81$0010.00BACKEDOUT
 666666A. BROOKMAN         LA HULPE, BRUSSELS  4298384026 11 81$0016.00BACKEDOUT
 700000A. MACALLA          DALLAS, TEXAS       5798432026 11 81$0002.00BACKEDOUT
 777777D. PRYKE            WILLIAMSBURG, VIRG. 9187613126 11 81$0027.00BACKEDOUT
 800000H. BRISTOW          WESTEND, LONDON     2423338926 11 81$0030.00BACKEDOUT
 888888B. HOWARD           NORTHAMPTON, ENG.   2369163926 11 81$0038.00BACKEDOUT
 900000D. WOODSON          TAMPA, FLA.         3566812026 11 81$0040.00BACKEDOUT
*=========================== End of File ======================================*
*
*    Closing Dpl Request has been attempted.
*    Close_Pipe call complete.
*    Deallocate_Pipe call complete.
*
*================== End of TEXCI Sample Batch Client Program ==================*
```

Figure 38. Output from DFH$ATXC

```
*======================= EXCI Sample Batch Client Program =======================*
*
*  EXEC Level Processor.
*    Setting up the EXEC level call.
*    The Link Request has successfully completed.
*    Server Response:
*      The file is set to a browsable state.
*
*  CALL Level Processor.
*    Initialise_User call complete.
*    Allocate_Pipe call complete.
*    Open_Pipe call complete.
*    The connection has been successful.
*      The target file follows:
*
*=========================== Top of File ======================================*
000100W. DAVIS            SURREY, ENGLAND     3215677826 11 81$0100.11COMMITTED
000102F. ALDSON           WARWICK, ENGLAND    9835618326 11 81$1111.11COMMITTED
000104S. BOWLER           LONDON,ENGLAND      1284629326 11 81$0999.99COMMITTED
000106B. ADAMS            CROYDON, ENGLAND    1948567326 11 81$0087.71COMMITTED
000111GENE BARLOWE        SARATOGA,CALIFORNIA 4612075301 02 74$0111.11COMMITTED
000762GEORGE BURROW       SAN JOSE,CALIFORNIA 2231212101 06 74$0000.00COMMITTED
000983H. L. L. CALL       WASHINGTON, DC      3451212021 04 75$9999.99COMMITTED
003210B.CREPIN            NICE, FRANCE        1234567026 11 81$3349.99COMMITTED
003214HUBERT C HERBERT    SUNNYVALE, CAL.     3411212000 06 73$0009.99COMMITTED
003890PHILIPPE SMITH, JR  NICE, FRANCE        0000000028 05 74$0009.99COMMITTED
004004STAN SMITH          DUBLIN, IRELAND     7111212102 11 73$1259.99COMMITTED
004445S. GALSON           SOUTH BEND, S.DAK.  6121212026 11 81$0009.99COMMITTED
004878D.C. CURRENT        SUNNYVALE, CALIF.   3221212010 06 73$5399.99COMMITTED
005005J. S. LAVERENCE     SAN FRANCISCO, CA.  0000000101 08 73$0009.99COMMITTED
005444JEAN LAWRENCE       SARATOGA, CALIF.    6771212020 10 74$0809.99COMMITTED
05581JOHN ALDEN III       BOSTON, MASS.       4131212011 04 74$0259.99COMMITTED
06016DR W. T. KAR         NEW DELHI, INDIA    7033121121 05 74$0009.88COMMITTED
006670WILLIAM KAPP        NEW YORK, N.Y.      2121212031 01 75$3509.88COMMITTED
006968D. CONRAD           WARWICK, ENGLAND    5671382126 11 81$0009.88COMMITTED
007248B. C. WILLIAMSON    REDWOOD CITY, CALF. 3331212111 10 75$0009.88COMMITTED
007779MRS. W. WELCH       SAN JOSE, CALIF.    4151212003 01 75$0009.88COMMITTED
100000G. NEADS            TORONTO, ONTARIO    0341512126 11 81$0010.00COMMITTED
111111C. MEARS            OTTAWA, ONTARIO     5121200326 11 81$0011.00COMMITTED
200000A. BONFIELD         GLASGOW,  SCOTLAND  6373829026 11 81$0020.00COMMITTED
300000K. TRENCHARD        NEW YORK, U.S.      6473980126 11 81$0030.00COMMITTED
333333D. MYRING           CARDIFF, WALES      7849302026 11 81$0033.00COMMITTED
400000W. TANNER           MILAN, ITALY        2536373826 11 81$0040.00COMMITTED
444444A. FISHER           CALGARY, ALBERTA    7788982026 11 81$0044.00COMMITTED
500000J. DENFORD          MADRID, SPAIN       4445464026 11 81$0000.00COMMITTED
555555C. JARDINE          KINGSTON, N.Y.      3994442026 11 81$0005.00Y00010500
600000F. HUGHES           DUBLIN, IRELAND     1239878026 11 81$0010.00Y00010600
666666A. BROOKMAN         LA HULPE, BRUSSELS  4298384026 11 81$0016.00Y00010700
700000A. MACALLA          DALLAS, TEXAS       5798432026 11 81$0002.00Y00010800
777777D. PRYKE            WILLIAMSBURG, VIRG. 9187613126 11 81$0027.00Y00010900
800000H. BRISTOW          WESTEND, LONDON     2423338926 11 81$0030.00Y00011000
888888B. HOWARD           NORTHAMPTON, ENG.   2369163926 11 81$0038.00Y00011100
900000D. WOODSON          TAMPA, FLA.         3566812026 11 81$0040.00Y00011200
*=========================== End of File ======================================*
*
*    Closing Dpl Request has been attempted.
*    Close_Pipe call complete.
*    Deallocate_Pipe call complete.
*
*=================== End of EXCI Sample Batch Client Program ==================*
```

Figure 39. Successful execution of DFH$AXCC after DFH$ATXC has been successfully executed.

Job control language to run an EXCI client program

An EXCI client program runs in an MVS address space, for example, as a batch job. Note the following requirements when writing the JCL for your client program:

- Include in the STEPLIB concatenation those libraries that contain the CICS-supplied external CICS interface modules and also the client program. The external CICS interface modules are supplied in CICSTS13.CICS.SDFHEXCI, which contains the following:

  DFH$AXCC
  DFHMEBM
  DFHMET4E
  DFHXCEIX
  DFHXCIE (alias of DFHXCSTB)
  DFHXCIS (alias of DFHXCTSB)
  DFHXCOPT
  DFHXCPRX
  DFHXCSTB
  DFHXCTRA
  DFHXCURM
- You are recommended to include a DD statement for SYSMDUMP. The external CICS interface uses SYSMDUMP for some error conditions.
- The REGION parameter must specify a large enough region size to allow for the size of the internal trace table specified by the TRACESZE parameter in the DFHXCOPT options table.
- Include a SYSPRINT or equivalent DD statement for any output from the client program.

Figure 40 shows a sample job that you can use or modify to start a client program.

```
//EXCI    JOB (accounting_information),CLASS=A,TIME=1440,
//        USER=userid,PASSWORD=pswd,REGION=100M
//*===============================================================*
//*    JCL to execute an external CICS interface client program   *
//*===============================================================*
//          EXEC  PGM=pgmname
//STEPLIB   DD    DSN=CICSTS13.CICS.EXCI.LOADLIB,DISP=SHR
//          DD    DSN=CICSTS13.CICS.SDFHEXCI,DSIP=SHR
//SYSPRINT  DD    SYSOUT=A
//SYSMDUMP  DD    DSN=SYS1.SYSMDP00,VOL=SER=volid,SPACE=(CYL,(1,1)),
                  DISP=OLD,UNIT=3390
```

Figure 40. Sample job for starting an EXCI client program

Notes:

1. The job user ID, specified on the USER parameter, must be defined to RACF, or an equivalent external security manager (ESM).
2. In addition to being used for job step initiation security, the job user ID is also used for MRO logon and bind-time security checking.

   See “Chapter 15. Security” on page 187 for information about security when using the external CICS interface.
3. See “Installing the EXCI sample definitions” on page 178 for information about modifying the sample connection definitions before you run the sample

application programs in an environment that does not have RACF, or an equivalent external security manager (ESM), installed and active.

Chapter 15. Security

CICS applies security checks in a number of ways against requests received from an MVS client program. These are described in the following topics:

- MRO logon and bind-time security
- Link security
- User security
- Surrogate user checking.

MRO logon and bind-time security

DFHIRP, the CICS interregion communication program, performs two security checks against users that want to:

1. Log on to IRP (**specific connections only**)
2. Connect to a CICS region (also referred to as bind-time security).

Generic EXCI connections

The discussion about logon security checking in this chapter applies only to EXCI connections that are defined as SPECIFIC. The MRO logon security check is not performed for generic connections.

The MVS client program is treated just the same as another CICS region as far as MRO logon and connect (bind-time) security checking is concerned. This means that when the client program logs on to the interregion communication program, IRP performs logon and bind-time security checks against the user ID under which the client program is running. In the remainder of this chapter, we refer to this as the batch region's user ID.

To enable your client program to log on successfully to IRP, and to connect to the target server region, first ensure that you define the batch region's user ID in a user profile to RACF. When you have defined the batch region's user ID to RACF, you can then give the batch region the appropriate logon and bind-time authorizations.

1. Logon authorization

Authorize the batch region's user ID to the DFHAPPL.*user_name* RACF FACILITY class profile, with UPDATE authority. The *user_name* part of the profile name is the user name defined on the INITIALIZE_USER command.

Failure to authorize the batch region's user ID to the DFHAPPL profile of the specific user ID logging on to IRP causes Allocate_Pipe processing to fail with RESPONSE(SYSTEM_ERROR) REASON(IRC_LOGON_FAILURE). The subreason field-1 for a logon security check failure returns decimal 204.

See "Defining DFHAPPL FACILITY class profiles for an EXCI region" on page 188 for information about FACILITY class profiles for an EXCI client program.

2. Bind-time authorization

Authorize the batch region's user ID to the DFHAPPL.*applid* RACF FACILITY class profile of the target CICS server region, with READ authority.

© Copyright IBM Corp. 1994, 1998

Failure to authorize the batch region's user ID to the CICS server region's DFHAPPL.*applid* profile causes Open_Pipe processing to fail with RESPONSE(SYSTEM_ERROR) REASON(IRC_CONNECT_FAILURE). The subreason field-1 for a bind-time security check failure returns decimal 176.

See the *CICS RACF Security Guide* for information about the MRO logon and bind-time security checks, and for examples of how to define the RACF DFHAPPL profiles.

Defining DFHAPPL FACILITY class profiles for an EXCI region

Define the *user_name* part of the DFHAPPL profile name as follows:

- For the EXCI CALL interface, the *user_name* must be the name you specify on the *user_name* parameter of the INITIALIZE_USER command.

  Define FACILITY class profiles, with appropriate authorizations, for each user name specified in a client program if the program has INITIALIZE_USER commands for more than one user name.

  For example, if the *user_name* defined on an INITIALIZE_USER command is DCEUSER1, define the DFHAPPL profile in the FACILITY class as follows:

  ```
  RDEFINE FACILITY (DFHAPPL.DCEUSER1) UACC(NONE)
  ```

  If the batch region's user ID is CLIENTA, authorize the batch region to log on to IRP as follows:

  ```
  PERMIT DFHAPPL.DCEUSER1  CLASS(FACILITY) ID(CLIENTA)
         ACCESS(UPDATE)
  ```
- For the EXEC CICS LINK command, the *user_name* is preset by the external CICS interface as DFHXCEIP. This does not require authorization for IRP logon, because the EXEC CICS LINK interface uses a generic connection to which the logon security check does not apply.

Link security

The target CICS server region performs link security checking against requests from the client program. These security checks cover transaction attach security (when attaching the mirror transaction), and resource and command security checking within the server application program. The link user ID that CICS uses for these security checks is the batch region's user ID.

To ensure these link security checks do not cause security failures, you must ensure that the link user ID is authorized to the following resource profiles, as appropriate:

- The profile for the mirror transaction, either CSMI for the default, or the mirror transaction specified on the *transid* parameter. This is required for transaction attach security checking.
- The profiles for all the resources accessed by the CICS server application program—files, queues (transient data and temporary storage), programs, and so on. This is required for resource security checking.
- The CICS command profiles for the SPI commands issued by the CICS server application program—INQUIRE, SET, DISCARD and so on. This is required for command security checking.

See the *CICS RACF Security Guide* for information about MRO link security checking.

User security

The target CICS server region performs user security checking against the user ID passed on a DPL_ Request call. User security checking is performed only when connections specify ATTACHCSEC(IDENTIFY).

User security is performed in addition to any link security.

For user security, in addition to any authorizations you make for link security, you must also authorize the user ID specified on the DPL_Request call.

Note that there is no provision for specifying a user ID on the EXEC CICS LINK command. In this case, the external CICS interface passes the batch region's user ID. User security checking is therefore performed against the batch region's user ID if the connection definition specifies ATTACHSEC(IDENTIFY).

Note: If your connection resource definitions for the external CICS interface specify ATTACHSEC(IDENTIFY), your server programs will fail with an ATCY abend if you run them in an environment that does not have RACF, or an equivalent external security manager (ESM), installed and active.

If you want to run external CICS interface server programs without any security active, you must specify ATTACHSEC(LOCAL).

Surrogate user checking

A surrogate user check is performed to verify that the batch region's user ID is authorized to issue DPL calls for another user (that is, is authorized as a surrogate of the user ID specified on the DPL_Request call).

EXCI client jobs are subject to surrogate user checking if SURROGCHK=YES (the default) is specified in the EXCI options table, DFHXCOPT. If you specify SURROGCHK=YES (or allow it to default) authorize the batch region's user ID as a surrogate of the user ID specified on all DPL_Request calls. This means the batch region's user ID must have READ access to a profile named *userid*.DFHEXCI in the SURROGAT general resource class (where*userid* is the user ID specified on the DPL call). For example, the following commands define a surrogate profile for a DPL userid, and grant READ access to the EXCI batch region:

```
RDEFINE  SURROGAT dpl_userid.DFHEXCI UACC(NONE) OWNER(DPL_userid)
PERMIT   userid.DFHEXCI CLASS(SURROGAT) ID(batch_region_userid)
                   ACCESS(READ)
```

If surrogate user checking is enabled (SURROGCHK=YES), but no user ID is specified on the DPL_Request call, no surrogate user check is performed, because the user ID on the DPL_Request call defaults to the batch region's user ID. For this bypass of surrogate user checking to be successful, ensure that you have correctly omitted the user ID on the DPL_Request call. See "Example of EXCI CALLs with null parameters" on page 146 for information about the correct way to specify a null pointer when omitting an EXCI call parameter.

If you don't want surrogate user security checking, specify SURROGCHK=NO in the DFHXCOPT options table (note that SURROGCHK=YES is the default).

Surrogate user checking is useful when the batch region's user ID is the same as the CICS server region user ID, in which case the link security check (see "Link security” on page 188) is bypassed. In this case, a surrogate user check is recommended, because the user ID specified on the DPL_Request call is not an authenticated user ID (no password is passed).

If the batch region’s user ID and the CICS region user ID are different, link security checking is enforced. With link security, a non-authenticated user ID passed on a DPL_Request call cannot acquire more authority than that allowed by the link security check. It can acquire only the same, or less, authority than that allowed by the link security check.

Further information

For more information about CICS security, see the *CICS RACF Security Guide*.

Chapter 16. Problem determination

This chapter contains Diagnosis, Modification or Tuning information.

This chapter describes some of the aids to problem determination provided by the external CICS interface. It covers:

- Trace
- System dumps
- MVS 04xx abends for the external CICS interface
- The EXCI service trap, DFHXCTRA
- EXCI trace entry points

Details of the external CICS interface messages and abend codes are given in "Chapter 18. Messages and Codes" on page 223.

Trace

The external CICS interface writes trace data to two destinations: an internal trace table and an external MVS GTF data set. The internal trace table resides in the non-CICS MVS address space. Trace data is formatted and included in any dumps produced by the external CICS interface.

Trace entries are issued by the external CICS interface destined for the internal trace table, an MVS GTF data set, or both. They are listed in "EXCI trace entry points" on page 197.

To use GTF for external CICS interface tracing, GTF user tracing must be active, GTF must be started in the MVS image, and you must specify GTF=ON in the DFHXCOPT options table.

If you use GTF trace for both the CICS server region and the external CICS interface region, the trace entries are interleaved, which can help you with problem determination in the CICS–EXCI environment.

Note: The external CICS interface maintains a separate trace table for each user TCB in an external CICS interface application program.

The external CICS interface does not support any form of auxiliary trace.

Formatting GTF trace

To format external CICS interface trace entries written to GTF, you can use the standard CICS DFHTR530 trace formatting routine.

To format external CICS interface trace entries you use the same FID and ID as for CICS (that is, FID=X'EF', and ID=X'F6C').

© Copyright IBM Corp. 1994, 1998

System dumps

The external CICS interface produces MVS SYSMDUMPs for some error conditions and MVS SDUMPs for other, more serious conditions. These dumps contain all the external CICS interface control blocks, as well as trace entries.

Formatting system dumps

You can use the CICS IPCS verb exit, DFHPD530, to format the system dumps. The following keywords are available for use when formatting an external CICS interface dump using DFHPD530:

KE
Formats PSW and registers, and all external CICS interface control blocks.

LD
Formats a load map of where the external CICS interface modules are loaded in the address space, and gives their PTF level.

MRO
Formats the MRO control blocks for the external CICS interface address space, including common control blocks that reside in the MVS common service area (CSA). This option also formats some MRO blocks that reside in the CICS address space for pipes connected to CICS.

TR
Formats the external CICS interface trace table. You can format the trace table in abbreviated and full forms (TR=1 gives you the abbreviated trace).

SU
Produces a dump summary.

Multiple TCBs

If the external CICS interface takes a system dump when there is more than one TCB in use, it dumps only the control blocks and trace table for the TCB that requested the dump.

If you take a dump of the external CICS address space using a console command, the CICS verb exit routine, DFHPD530 formats the control blocks and trace tables for every TCB it finds in the dump.

Capturing SYSMDUMPs

To capture SYSMDUMPs produced by the external CICS interface, ensure you always include a DD statement for the SYSMDUMP data set in the client application program’s JCL.

Using the MVS DUMP command at the console for dumps

In addition to the dumps taken automatically by the external CICS interface, you can also force a dump of an address space running a client application program by means of an MVS DUMP command entered at the console. You can use the CICS IPCS verb exit routine DFHPD530 to format dumps taken in this way.

Note: You can also issue the DUMP command from TSO, SDSF, or NetView.

You can use the console to issue the DUMP command also to dump the CICS server address space as well as the client address space, and use the CICS IPCS verb exit routine DFHPD530 to format both address space dumps together.

MVS 04xx abends for the external CICS interface

The following MVS 04xx abends can occur when you are running an external CICS interface job:

0401

Explanation: An external CICS interface (EXCI) request was issued using the CALL API or the EXEC API, and the EXCI stub DFHXCSTB link-edited with the application detected that it was running in AMODE 24. The external CICS interface only supports calls made in AMODE 31.

System Action: The application terminates abnormally.

User Response: Change the application so that EXCI calls are made in AMODE 31, or relink-edit the application AMODE 31.

Module: DFHXCSTB

0402

Explanation: The external CICS interface module DFHXCPRH issued an MVS ESTAE macro to establish a recovery environment, but a nonzero return code was returned from MVS.

System Action: The application terminates abnormally with a dump.

User Response: Examine the dump and any associated MVS messages produced to determine why the MVS ESTAE request failed.

If the error occurred while processing an INITIALIZE_USER request on behalf of the application, an attempt to format the dump using the CICS IPCS dump formatter does not produce any formatted output. This is because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0403

Explanation: The external CICS interface module DFHXCPRH issued an MVS GETMAIN request to obtain storage for its XCGLOBAL block, but a nonzero return code was returned from MVS.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0403 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(XCGLOBAL_GETMAIN_ERROR) in its return area. The subreason1 field of the return area contains the R15 return code from MVS indicating why the GETMAIN failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(602).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the MVS GETMAIN request failed. If the reason is insufficient storage, increase the region size of the batch application.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0404

Explanation: The external CICS interface module DFHXCPRH needed to take an MVS SDUMP for an earlier reported problem. However the error has occurred too early in EXCI initialization for EXCI dump services to be available.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0404 which invokes its ESTAE routine from which a SYSMDUMP is taken instead of an SDUMP to capture the earlier reported problem.

User Response: Examine the SYSMDUMP to determine the cause of the earlier reported problem.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0405

Explanation: The external CICS interface module DFHXCPRH issued an IEFSSREQ SSI verify request to MVS to determine the number of the CICS SVC type 3 SVC to use. The SSI VERIFY request failed.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0405 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(SSI_VERIFY_FAILED) in its return area. The subreason1 field of the return area contains the R15 return code from MVS indicating why the SSI verify failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(606).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the SSI VERIFY request failed.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0406

Explanation: The external CICS interface module DFHXCPRH called the CICS SVC to initialize the EXCI environment. The CICS SVC call failed.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0406 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(CICS_SVC_CALL_FAILURE) in its return area. The subreason1 field of the return area contains the R15 return code from the CICS SVC indicating why it failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(607).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the CICS SVC call failed.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0407

Explanation: The external CICS interface module DFHXCPRH issued a call to the CICS SVC to check whether the SVC in use is at the correct level to be used with the external CICS interface. The check failed indicating that the CICS SVC is not at the correct level.

System Action: Message DFHEX0100 is output, and module DFHXCPRH issues an MVS abend with abend code 0407 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(INCORRECT_SVC_LEVEL) in its return area. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(627).

User Response: See the explanation of message DFHEX0100 for guidance.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0408

Explanation: The external CICS interface module DFHXCPRH issued an MVS GETMAIN request for its working storage but a nonzero return code was returned from MVS.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0408 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(WS_GETMAIN_ERROR) in its return area. The subreason1 field of the return area contains the R15 return code from MVS indicating why the GETMAIN failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(601).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the MVS GETMAIN request failed. If the reason is insufficient storage, increase the region size of the batch application.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0409

Explanation: The external CICS interface module DFHXCPRH issued an MVS GETMAIN request for storage required for its SSI VERIFY request, but a nonzero return code was returned from MVS.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0409 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(VERIFY_BLOCK_GM_ERROR) in its return area. The subreason1 field of the return area contains the R15 return code from MVS indicating why the

GETMAIN failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(605).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the MVS GETMAIN request failed. If the reason is insufficient storage, increase the region size of the batch application.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCPRH

0410

Explanation: The external CICS interface module DFHXCPRH issued an MVS GETMAIN request for an XCUSER block but a nonzero return code was returned from MVS.

System Action: Module DFHXCPRH issues an MVS abend with abend code 0410 which invokes its ESTAE routine to clear up its environment. A SYSMDUMP is taken before returning control to the application. An application using the EXCI CALL API receives RESPONSE(SYSTEM_ERROR) REASON(XCUSER_GETMAIN_ERROR) in its return area. The subreason1 field of the return area contains the R15 return code from MVS indicating why the GETMAIN failed. An application using the EXCI EXEC API receives RESP(LINKERR) RESP2(603).

User Response: Use the MVS R15 return code obtained from the application or from the dump to determine why the MVS GETMAIN request failed. If the reason is insufficient storage, increase the region size of the batch application.

Module: DFHXCPRH

0411

Explanation: The external CICS interface dump module DFHXCDMP was attempting to call the CICS SVC in order for an MVS SDUMP to be taken to capture an earlier problem. DFHXCDMP was unable to call the SVC as no SVC number was available. DFHXCDMP issued an 0411 abend in order that the callers ESTAE routine is invoked which takes a SYSMDUMP instead.

System Action: A SYSMDUMP is taken instead of an SDUMP for an earlier reported problem.

User Response: Use the SYSMDUMP produced to diagnose the earlier reported problem.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCDMP

0412

Explanation: The external CICS interface dump module DFHXCEIP was processing an EXCI EXEC API request and detected that the EXEC parameter list passed to it contained a function that is not supported by the external CICS interface.

System Action: The application is abnormally terminated with a dump.

User Response: This error indicates that the parameter list being passed to the EXCI has not been generated by the CICS translator. The translator should always be used. Correct the application to specify the correct EXCI EXEC API command.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter may not produce any formatted output for the job if this was the first EXCI request for this TCB.

Module: DFHXCEIP

0413

Explanation: The external CICS interface dump module DFHXCEIP was processing an EXCI EXEC API request and detected that the EXEC parameter list passed to it did not require the mandatory RETCODE parameter in which return codes are returned to the application.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter may not produce any formatted output for the job if this was the first EXCI request for this TCB.

System Action: The application is abnormally terminated with a dump.

User Response: This error indicates that the parameter list being passed to the EXCI has not been generated by the CICS translator. The translator should always be used. Correct the application to specify RETCODE.

Module: DFHXCEIP

0414

Explanation: The external CICS interface module DFHXCEIP issued an MVS ESTAE macro to establish a recovery environment but a nonzero return code was returned from MVS.

System Action: The application terminates abnormally with a dump.

User Response: Examine the dump and any associated MVS messages to determine why the MVS ESTAE request failed.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter may not produce any formatted output for the job if this was the first EXCI request for this TCB.

Module: DFHXCEIP

0415

Explanation: The external CICS interface module DFHXCEIP detected an error early in EXCI initialization before EXCI dump services were available. DFHXCEIP issues abend 0415 so that its ESTAE routine is invoked from where an SYSMDUMP is taken instead to capture the error.

System Action: The application terminates abnormally with a dump.

User Response: Examine the SYSMDUMP to determine the cause of the earlier reported error.

An attempt to format the SYSMDUMP produced with the CICS IPCS dump formatter does not produce any formatted output for the job because the error occurred too early in EXCI initialization for there to be any control blocks.

Module: DFHXCEIP

The EXCI service trap, DFHXCTRA

A user-replaceable program, DFHXCTRA, is available for use under the guidance of IBM service personnel. It is the equivalent of DFHTRAP used in CICS. It is invoked every time the external CICS interface writes a trace entry.

DFHXCTRA can perform one or all of the following actions:

1. Request the external CICS interface to write a trace entry on its behalf
2. Instruct the external CICS interface to take an SDUMP
3. Instruct the external CICS interface to skip writing the current trace entry to GTF
4. Instruct the external CICS interface to disable DFHXCTRA

The CICS-supplied sample version of DFHXCTRA performs all four of the above functions if it detects a trace entry that indicates that a FREEMAIN error occurred while trying to free an EXCI pipe control block.

The source for DFHXCTRA is supplied in CICSTS13.CICS.SDFHMAC. The parameter list passed to DFHXCTRA is defined in the copybook DFHXCTRD, which is supplied in CICSTS13.CICS.SDFHMAC. DFHXCTRD also defines all the external CICS interface trace points for use by DFHXCTRA.

Problem determination with RRMS

When Recoverable Resource Management Services (RRMS) is used to coordinate DPL requests, you can obtain additional problem determination information from RRMS. To do this, you use ISPF dialogs provided by Resource Recovery Services (RRS). The dialogs enable you to:

- Browse the RRS log streams
- Display information about RRS resource managers
- Display information about RRS Units of Recovery

Please refer to *OS/390 MVS Programming: Resource Recovery* for information about how to install and use the dialogs.

EXCI trace entry points

Reviewer

Additional trace points.

surrogate user checking - EX2009 and EX200A

Table 19. External CICS interface trace entries

Point ID	Module	Lvl	Type	Data
EX 0001	DFHXCPRH	Exc	PIPE_ALREADY_OPEN	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token
EX 0002	DFHXCPRH	Exc	PIPE_ALREADY_CLOSED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token
EX 0003	DFHXCPRH	Exc	VERIFY_BLOCK_FM_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0005	DFHXCPRH	Exc	XCPIP_ FM_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token
EX 0006	DFHXCPRH	Exc	IRP_IOAREA_FM_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0008	DFHXCPRH	Exc	XFRASTG1_FM_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0201	DFHXCPRH	Exc	NO_CICS_IRC_STARTED	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer
EX 0202	DFHXCPRH	Exc	NO_PIPE	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid
EX 0203	DFHXCPRH	Exc	NO_CICS_ON_OPEN	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid
EX 0204	DFHXCPRH	Exc	NO_CICS_ON_DPL_1	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid
EX 0205	DFHXCPRH	Exc	NO_CICS_ON_DPL_2	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid
EX 0206	DFHXCPRH	Exc	NO_CICS_ON_DPL_3	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0403	DFHXCPRH	Exc	INVALID_APPL_NAME	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0405	DFHXCPRH	Exc	PIPE_NOT_CLOSED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token
EX 0406	DFHXCPRH	Exc	PIPE_NOT_OPEN	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token
EX 0407	DFHXCPRH	Exc	INVALID_USERID	1. Caller's parameter list 2. Call type 3. Caller's user name
EX 0408	DFHXCPRH	Exc	INVALID_UOWID	1. Caller's parameter list 2. Call type 3. Caller's user name 4. UOWID
EX 0409	DFHXCPRH	Exc	INVALID_TRANSID	1. Caller's parameter list 2. Call type 3. Caller's user name
EX 0414	DFHXCPRH	Exc	ABORT_RECEIVED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Target CICS applid 5. Message to be returned
EX 0415	DFHXCPRH	Exc	INVALID_CONNECTION	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Connection name 5. Target CICS applid
EX 0416	DFHXCPRH	Exc	INVALID_CICS_RELEASE	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Target CICS applid

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0417	DFHXCPRH	Exc	PIPE_MUST_CLOSE	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Pipe token
EX 0418	DFHXCPRH	Exc	INVALID_PIPE_TOKEN	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Pipe token
EX 0422	DFHXCPRH	Exc	SERVER_ABENDED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. DPL return area
EX 0423	DFHXCPRH	Exc	SURROGATE_CHECK_FAILED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Job user ID 5. Surrogate resource name 6. ESM return code and reason code
EX 0603	DFHXCPRH	Exc	XCUSER_GM_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0604	DFHXCPRH	Exc	XCPIPE_GM_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0605	DFHXCPRH	Exc	VERIFY_BLOCK_GM_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0606	DFHXCPRH	Exc	SSI_VERIFY_FAILED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0607	DFHXCPRH	Exc	SVC_CALL_FAILED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0608	DFHXCPRH	Exc	IRP_LOGON_FAILURE	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid 6. Logon name
EX 0609	DFHXCPRH	Exc	IRP_CONNECT_FAIL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token 6. Target CICS applid
EX 0610	DFHXCPRH	Exc	IRP_DISC_FAIL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid 6. Pipe token
EX 0611	DFHXCPRH	Exc	IRP_LOGOFF_FAILED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid 6. Pipe token
EX 0612	DFHXCPRH	Exc	TRANSFORM_1_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0613	DFHXCPRH	Exc	TRANSFORM_4_ERROR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0614	DFHXCPRH	Exc	IRP_NULL_DATA	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid
EX 0615	DFHXCPRH	Exc	IRP_NEG_RESPONSE	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid
EX 0616	DFHXCPRH	Exc	IRP_SWITCH_PULL_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Target CICS applid 6. Pipe token
EX 0617	DFHXCPRH	Exc	IRP_IOAREA_GM_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0619	DFHXCPRH	Exc	IRP_BAD_IOAREA	1. Caller's parameter list 2. Call type 3. Caller's user name 4. IOAREA address
EX 0620	DFHXCPRH	Exc	IRP_PROTOCOL_ERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Target CICS applid 5. Pipe token
EX 0621	DFHXCPRH	Exc	PIPE_RECOVERY_FAILURE	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Target CICS applid 5. Pipe token
EX 0622	DFHXCPRH	Exc	ESTAE_SETUP_FAIL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0623	DFHXCPRH	Exc	ESTAE_INVOKED	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. MVS abend code
EX 0624	DFHXCPRH	Exc	TIMEDOUT	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Server program name 5. Target CICS applid
EX 0625	DFHXCPRH	Exc	STIMER_SETUP_FAIL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0626	DFHXCPRH	Exc	STIMER_CANCEL_FAIL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer
EX 0627	DFHXCPRH	Exc	INCORRECT_SVC_LEVEL	1. Caller's parameter list 2. Call type 3. Caller's user name 4. SVC instruction
EX 0800	DFHXCPRH	Exc	RESP shows LENGERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. COMMAREA length 6. Data length
EX 0801	DFHXCPRH	Exc	RESP shows INVREQ	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. DPL options specified
EX 0802	DFHXCPRH	Exc	RESP shows PGMIDERR	1. Caller's parameter list 2. Call type 3. Caller's user name 4. Program name 5. Target CICS applid

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 0803	DFHXCPRH	Exc	RESP shows ROLLEDBACK	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Program name 5. Target CICS applid
EX 0804	DFHXCPRH	Exc	RESP shows NOTAUTH	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Program name 5. Target CICS applid
EX 0805	DFHXCPRH	Exc	RESP shows SYSIDERR	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Program name 5. Target CICS applid 6. DPL_Retarea
EX 0806	DFHXCPRH	Exc	RESP shows TERMERR	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Program name 5. Target CICS applid
EX 0904	DFHXCTRP	Exc	Overlength trace data field	1. XCTRP parameter list
EX 0905	DFHXCTRA	Exc	DFHXCTRA trace entry	1. User specified data

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 1000	DFHXCPRH	EX 1	Entry	For INIT_USER commands: 1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Caller’s register 14 For Allocate_Pipe requests: 1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. CICS name 5. Allocate options 6. Caller’s register 14 For Open, Close, and Deallocate requests: 1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. CICS name 5. Pipe token 6. Caller’s register 14 For DPL requests: 1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. CICS name 5. Pipe token 6. Program name 7. Caller’s register 14

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 1001	DFHXCPRH	EX 1	Exit	For INIT_USER, OPEN, CLOSE, and DEALLOCATE requests: 1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Caller's register14 For Allocate requests: 1. Caller's parameter list 2. Call type 3. Caller's user name 4. Return codes and message pointer 5. Pipe token 6. Caller's register 14 For DPL requests: 1. Caller's parameter list 2. Call type 3. Caller's user name 4. Target CICS system 5. Pipe token
EX 1010	DFHXCEIP	EX 1	Entry	1. Program name 2. Target CICS applid 3. Transaction ID 4. Caller's register 14 5. Up to the first 100 bytes of COMMAREA (if passed) 6. COMMAREA length, if COMMAREA passed 7. Data length, if COMMAREA passed
EX 1011	DFHXCEIP	EX 1	Exit	1. EXEC retarea 2. Program name 3. Target CICS applid 4. Transaction ID 5. Caller's register 14 6. Up to the first 100 bytes of COMMAREA (if passed) 7. COMMAREA length, if COMMAREA passed

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 2000	DFHXCPRH	EX 2	IRP_LOGON	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Target CICS applid 5. IRP user ID 6. SLCB address 7. Connection name
EX 2001	DFHXCPRH	EX 2	IRP_CONN	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Target CICS applid 5. IRP user ID 6. IRP thread ID 7. SCCB address
EX 2002	DFHXCPRH	EX 2	IRP_DISC	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Target CICS applid 5. Pipe token
EX 2003	DFHXCPRH	EX 2	IRP_LOGOFF	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Pipe token 5. IRP user ID
EX 2004	DFHXCPRH	EX 2	IRP_SWITCH	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Target CICS applid 5. IRP user ID 6. IRP user thread
EX 2005	DFHXCPRH	EX 2	IRP_SWITCH_DATA	1. User’s appl name 2. Pipe token 3. Request header 4. Bind data 5. UOWID/USERID FMH 6. Transformed DPL request to CICS (up to 1000 bytes) 7. Final 1000 bytes of transformed DPL request

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 2006	DFHXCPRH	EX 2	IRP_DATA	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Target CICS applid 5. Length of data returned 6. Data (first 1000 bytes) 7. Data (final 1000 bytes)
EX 2007	DFHXCPRH	EX 2	PRE_URM	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Parameters passed to DFHXCURM 5. URMINV, reason for calling URM 6. URMCICS, target CICS applid 7. URMANCH, URM anchor point address
EX 2008	DFHXCPRH	EX 2	POST_URM	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Parameters passed to DFHXCURM 5. URMINV, reason for calling URM 6. URMCICS, target CICS applid 7. URMANCH, URM anchor point address
EX 2009	DFHXCPRH	EX 2	PRE-RACROUTE	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Userid 5. Surrogate resource name 6. RACROUTE parameter list
EX 200A	DFHXCPRH	EX 2	POST-RACROUTE	1. Caller’s parameter list 2. Call type 3. Caller’s user name 4. Userid 5. Surrogate resource name 6. RACROUTE parameter list
EX 3000	DFHXCEIP	Exc	ESTAE_SETUP_ERROR	1. Return area (20 bytes) 2. MVS return code
EX 3001	DFHXCEIP	Exc	ESTAE_INVOKED	1. Return area (20 bytes)
EX 3002	DFHXCEIP	Exc	INV_CTYPE_ON_INIT	1. Return area (20 bytes) 2. Call type

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 3003	DFHXCEIP	Exc	INV_VNUM_ON_INIT	1. Return area (20 bytes) 2. Version number
EX 3004	DFHXCEIP	Exc	INV_APPL_NAME_ON_INIT	1. Return area (20 bytes) 2. User name
EX 3005	DFHXCEIP	Exc	INV_CTYPE_ON_ALLOC	1. Return area (20 bytes) 2. Call type
EX 3006	DFHXCEIP	Exc	INV_VNUM_ON_ALLOC	1. Return area (20 bytes) 2. Version number
EX 3007	DFHXCEIP	Exc	INV_UTOKEN_ON_ALLOC	1. Return area (20 bytes) 2. User token
EX 3008	DFHXCEIP	Exc	INV_CTYPE_ON_OPEN	1. Return area (20 bytes) 2. Call type
EX 3009	DFHXCEIP	Exc	INV_VNUM_ON_OPEN	1. Return area (20 bytes) 2. Version number
EX 3010	DFHXCEIP	Exc	INV_UTOKEN_ON_OPEN	1. Return area (20 bytes) 2. User token
EX 3011	DFHXCEIP	Exc	INV_PTOKEN_ON_OPEN	1. Return area (20 bytes) 2. Pipe token
EX 3012	DFHXCEIP	Exc	INV_CTYPE_ON_DPL	1. Return area (20 bytes) 2. Call type
EX 3013	DFHXCEIP	Exc	INV_VNUM_ON_DPL	1. Return area (20 bytes) 2. Version number
EX 3014	DFHXCEIP	Exc	INV_UTOKEN_ON_DPL	1. Return area (20 bytes) 2. User token
EX 3015	DFHXCEIP	Exc	INV_PTOKEN_ON_DPL	1. Return area (20 bytes) 2. Pipe token
EX 3017	DFHXCEIP	Exc	INV_USERID_ON_DPL	1. Return area (20 bytes)
EX 3018	DFHXCEIP	Exc	PIPE_NOT_OPEN_ON_DPL	1. Return area (20 bytes) 2. Pipe token
EX 3019	DFHXCEIP	Exc	PIPE_MUST_CLOSE_ON_DPL	1. Return area (20 bytes) 2. Pipe token
EX 3020	DFHXCEIP	Exc	INV_CTYPE_ON_CLOSE	1. Return area (20 bytes) 2. Call type
EX 3021	DFHXCEIP	Exc	INV_VNUM_ON_CLOSE	1. Return area (20 bytes) 2. Version number
EX 3022	DFHXCEIP	Exc	INV_UTOKEN_ON_CLOSE	1. Return area (20 bytes) 2. User token

Table 19. External CICS interface trace entries (continued)

Point ID	Module	Lvl	Type	Data
EX 3023	DFHXCEIP	Exc	INV_PTOKEN_ON_CLOSE	1. Return area (20 bytes) 2. Pipe token
EX 3024	DFHXCEIP	Exc	INV_CTYPE_ON_DEALL	1. Return area (20 bytes) 2. Call type
EX 3025	DFHXCEIP	Exc	INV_VNUM_ON_DEALL	1. Return area (20 bytes) 2. Version number
EX 3026	DFHXCEIP	Exc	INV_UTOKEN_ON_DEALL	1. Return area (20 bytes) 2. User token
EX 3027	DFHXCEIP	Exc	INV_PTOKEN_ON_DEALL	1. Return area (20 bytes) 2. Pipe token
EX 3028	DFHXCEIP	Exc	PIPE_NOT_CLOSED_ON_DEALL	1. Return area (20 bytes) 2. Pipe token
EX 3029	DFHXCEIP	Exc	XCEIP_RETRYING	1. Return area (20 bytes)
EX 3030	DFHXCEIP	Exc	SURROGATE_CHECK_FAILED	1. Return area (20 bytes)
EX 4000	DFHXCGUR	EX 1	Entry	1. DFHXCGUR parameter list
EX 4001	DFHXCGUR	EX 2	Exit	1. DFHXCGUR parameter list
EX 4002	DFHXCGUR	EX 1	PRE_SVC1	1. SVC parameter list
EX 4003	DFHXCGUR	EX 1	POST_SVC	1. SVC parameter list
EX 4004	DFHXCGUR	Exc	RRMS_NOT_SUPPORTED	1. None
EX 4005	DFHXCGUR	Exc	RRMS_ERROR	1. None
EX 4006	DFHXCGUR	Exc	SVC_EXCEPTION	1. SVC return code

Chapter 17. Response and reason codes returned on EXCI calls

This chapter gives details of the reason codes for the responses returned on the EXCI call interface.

Note: All numeric response and reason code values shown are in decimal.

Reason code for response: OK

0 NORMAL

Explanation: Call completed normally.

Reason codes for response: WARNING

1 PIPE_ALREADY_OPEN

Explanation: An Open_Pipe request has been issued for a pipe that is already open.

System Action: None. The pipe remains open.

User Response: If this response is unexpected, investigate whether an incorrect pipe token has been used on the Open_Pipe call.

2 PIPE_ALREADY_CLOSED

Explanation: A Close_Pipe request has been issued for a pipe that is already closed.

System Action: The external CICS interface ignores the request and the pipe remains closed.

User Response: If the response is unexpected, check that the Close_Pipe call is specifying the correct pipe token.

3 VERIFY_BLOCK_FM_ERROR

Explanation: Initialize_User processing requires storage below 16MB to build the parameter list for the SSI Verify call, and an error has occurred during the FREEMAIN for this area.

System Action: The return code from the FREEMAIN is returned in the EXCI subreason field-1. The Initialize_User request continues unaffected.

User Response: If the problem persists, take a dump of the batch region and use the dump, together with the return code from the MVS FREEMAIN, to determine why the FREEMAIN is failing.

4 WS_FREEMAIN_ERROR

Explanation: An attempt to FREEMAIN working storage has resulted in an MVS FREEMAIN error.

System Action: The return code from the FREEMAIN is returned in the EXCI subreason field-1. The Initialize_User request continues unaffected.

User Response: If the problem persists, take a dump of the batch region and use the dump, together with the return code from the MVS FREEMAIN to determine why the FREEMAIN is failing.

5 XCPIPE_FREEMAIN_ERROR

Explanation: An attempt to FREEMAIN pipe storage has resulted in an MVS FREEMAIN error.

System Action: The return code from the FREEMAIN is returned in the EXCI subreason field-1. However, the external CICS interface continues processing the Deallocate_Pipe request. If the request fails with another error, this reason code is overwritten.

User Response: If the problem persists, take a dump of the client application program address space, and use the dump, with the return code from the MVS FREEMAIN to determine why the FREEMAIN is failing.

6 IRP_IOAREA_FM_FAILURE

Explanation: An attempt to FREEMAIN an MRO I/O area has resulted in an MVS FREEMAIN error.

System Action: The return code from the FREEMAIN is returned in the EXCI subreason field-1, but the DPL request continued to completion. Reason IRP_IOAREA_FM_FAILURE is returned to your application only if the DPL request completes, otherwise it is overwritten by subsequent response and reason codes.

User Response: If the problem persists, take a dump of the batch region and use it with the return code from the MVS FREEMAIN to determine why the FREEMAIN is failing.

© Copyright IBM Corp. 1994, 1998

7 SERVER_TERMINATED

Explanation: The CICS session, on which the server program has been executing, has been freed by CICS.

System Action: The CICS application server program has been detached at some point in its processing, and control is returned to the external CICS interface, which writes a trace entry for this error.

User Response: The most likely reason for this error is that the server program has caused CICS to terminate, perhaps by an EXEC CICS PERFORM SHUTDOWN command. During shutdown, CICS frees EXCI sessions so that shutdown can complete.

8 XFRASTG1_FM_FAILURE

Explanation: An attempt to FREEMAIN the transmission area has resulted in an MVS FREEMAIN error.

System Action: The return code from the FREEMAIN is returned in the EXCI subreason field-1 but the DPL request continued to completion. Reason XFRASTG1_FM_FAILURE is returned to your application only if the DPL request completes, otherwise it is overwritten by subsequent response and reason codes.

User Response: If the problem persists, take a dump of the batch region and use it with the return code from the MVS FREEMAIN to determine why the FREEMAIN is failing.

Reason codes for response: RETRYABLE

201 NO_CICS_IRC_STARTED

Explanation: An Initialize_User command has been issued on an MVS image that has had no IRC activity since the previous IPL, and the external CICS interface cannot determine the CICS SVC number.

System Action: The Initialize_User call fails, and the external CICS interface invokes the user-replaceable module, DFHXCURM.

User Response: Ensure that a CICS region in the MVS image has logged on to IRC (that is, has started up with the system initialization parameter IRCSTRT=YES or has started IRC dynamically with an OPEN IRC command). Alternatively, if there is no local CICS region in the MVS image, you must specify the SVC parameter that the external CICS interface is to use, by coding a CICSSVC parameter in the DFHXCOPT table. This situation can occur if you are using XCF to communicate to a CICS region in another MVS image. Once the problem has been resolved, re-issue the Initialize_User request.

202 NO_PIPE

Explanation: An attempt has been made to open a pipe, but the target CICS system associated with the pipe has no free receive sessions.

System Action: The Open_pipe call fails, and the external CICS interface invokes the user-replaceable module, DFHXCURM.

User Response: This situation can occur even if the client application program has allocated (using Allocate_Pipe calls) no more pipes than the number of receive sessions defined on the target connection. This is because CICS can be in the process of cleaning up a pipe from a Close_Pipe request. For this reason, you are recommended to specify a larger RECEIVECOUNT value than is theoretically necessary when defining the SESSIONS resource definition to CICS. The application program can reissue the Open_Pipe request.

203 (on Open_Pipe call)
NO_CICS

Explanation: An attempt has been made to open a pipe but the target CICS system is not available, or hasn't yet opened IRC, or the target connection is out of service, or the relevant EXCI connection definition is not installed in the target CICS.

System Action: The open pipe request fails, and the external CICS interface invokes the user-replaceable module, DFHXCURM.

User Response: If subreason field-1 is non-zero (the IRP response code (R15)), subreason field-2 contains the IRP reason code. For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPScopybook, listed under the heading IRC.

When you have corrected the problem, your client application program can reissue the Open_Pipe call.

204 WRONG_MVS_FOR_RRMS

Explanation: A DPL request omitting the SYNCONRETURN option has been made specifying a CICS region that is on a different MVS system from the batch program. Because the Recoverable Resource Management Services (RRMS) context is not recognised in the target system, the request is rejected.

System Action: The DPL request fails, and the external CICS interface invokes the user-replacable module, DFHXCURM.

User Response: Ensure that the batch program that issued the DPL request and the CICS region it was sent to are on the same MVS system.

205 RRMS_NOT_AVAILABLE

Explanation: A DPL request omitting the SYNCONRETURN option has been made when the Resource Recovery Services (RRS) is not available. There are two cases:

- When Resource Recovery Services (RRS) is not available.
- When Resource Recovery Services has restarted since the last DPL request omitting the SYNCONRETURN option, and there has been no intervening syncpoint.

- -

Note: RRS is a part of Recoverable Resource Management Services (RRMS).

System Action: The DPL request fails, and the external CICS interface invokes the user-replacable module, DFHXCURM.

User Response: Retry the DPL request when Resource Recovery Services has restarted since the last DPL request omitting the SYNCONRETURN option, and there has been no intervening syncpoint.

Reason codes for response: USER_ERROR

401 INVALID_CALL_TYPE

Explanation: An invalid *call-type* parameter value is specified on this EXCI request.

System Action: The request is rejected.

User Response: Check your EXCI client program and ensure the *call_type* parameter specifies the appropriate value for the EXCI call, as follows.

1	Initialize_User
2	Allocate_Pipe
3	Open_Pipe
4	Close_Pipe
5	Deallocate_Pipe
6	DPL

402 INVALID_VERSION_NUMBER

Explanation: The *version_number* parameter does not specify a value of 1.

System Action: The request is rejected.

User Response: Check the client application program and ensure that all EXCI calls specify the value of 1 for the version number.

403 INVALID_APPL_NAME

Explanation: The *user_name* parameter consists of all blank characters (X'40').

System Action: The call is rejected.

User Response: Change the application program to specify a valid, non-blank user name.

404 INVALID_USER_TOKEN

Explanation: The client application program has issued an EXCI request using a user token that is unknown to the external CICS interface.

System Action: The request is rejected.

User Response: The Initialize_User call returns a 4-byte token that must be used on *all* further requests for the that user. Check the client application program and correct the error to ensure that the correct token is passed.

405 PIPE_NOT_CLOSED

Explanation: A Deallocate_Pipe request has been issued against a pipe that has not yet been closed.

System Action: The external CICS interface ignores the request and the pipe remains open.

User Response: Check the client application program, and ensure that the Deallocate_Pipe request is intended. If so, issue a Close_Pipe request for the pipe before issuing the Deallocate_Pipe request.

406 PIPE_NOT_OPEN

Explanation: A DPL call has been issued on a pipe that is not open.

System Action: The external CICS interface rejects the DPL request.

User Response: Check the client application program, and ensure that an Open_Pipe request is issued before using the pipe on a DPL request. If an Open_Pipe has been issued by the application program, check that it has not been closed inadvertently before all the DPL requests have been made.

407 INVALID_USERID

Explanation: A DPL request has been issued with a USERID parameter that consists of all blanks.

System Action: The DPL request is rejected.

User Response: Check the EXCI client program and ensure that the DPL request passes a valid USERID parameter. If you don't want to specify a userid, code the call parameter list with a null address for *userid*. If you pass a null address, the external CICS interface passes the userid under which the client application program is running (the batch region's userid).

408 INVALID_UOWID

Explanation: A DPL request has been issued with a *uowid* parameter that has invalid length fields.

System Action: The DPL request is rejected.

User Response: Check the client application program and ensure that the DPL request passes a valid *uowid* parameter. If you don't want to specify a unit of work id, code the call parameter list with a null address for *uowid*, in which case the external CICS interface generates a unit of work id for you.

409 INVALID_TRANSID

Explanation: A DPL request has been issued with a *transid* parameter that consists of all blanks.

System Action: The DPL request is rejected.

User Response: Check the client application program and ensure that the *transid* parameter is specified correctly or has not been overwritten in some way. If you don't want to specify your own transid, code the call parameter list with a null address for *transid*, in which case the external CICS interface uses the default CICS mirror transaction, CSMI.

410 DFHMEBM_LOAD_FAILED

Explanation: During Initialize_User processing, the external CICS interface attempted to load the main message module in preparation for issuing external CICS interface messages, and the load of this module failed.

System Action: The Initialize_User call is rejected. The return code from the MVS load macro (R15) is returned in the subreason field-1. The external CICS interface handles the error, and returns the abend (R0) that would have occurred in the subreason field-2.

User Response: Using the MVS return code, determine why the load failed. The most likely reason is that the message module, DFHMEBM, is not in any library included in the STEPLIB of the batch job. Ensure the CICSTS13.CICS.SDFHEXCI library is included in the STEPLIB concatenation, and restart the client application program.

411 DFHMET4E_LOAD_FAILED

Explanation: The load of message module, DFHMET4E, has failed. During Initialize_User processing, the external CICS interface attempted to load its message table in preparation for issuing messages. The load of this module failed.

System Action: The Initialize_User call is rejected. The return code from the MVS load macro (R15) is returned in the subreason field-1. The external CICS interface handles the error, and returns the abend (R0) that would have occurred in the subreason field-2.

User Response: Using the MVS reason code, determine why the load failed. The most likely reason is that the message table, DFHMET4E, is not in any library included in the STEPLIB of the batch job. Ensure the CICSTS13.CICS.SDFHEXCI library is included in the STEPLIB concatenation, and restart the client application program.

412 DFHXCURM_LOAD_FAILED

Explanation: During Initialize_User processing, the external CICS interface attempted to load the user-replaceable module, DFHXCURM. The load of this module failed.

System Action: The Initialize_User call is rejected. The return code from the MVS load macro (R15) is returned in the subreason field-1. The external CICS interface handles the error, and returns the abend (R0) that would have occurred in the subreason field-2.

User Response: Using the MVS reason code, determine why the load failed. The most likely reason is that module DFHXCURM is not in any library included in the STEPLIB of the batch job. Ensure the library containing the module is included in the STEPLIB concatenation, and restart the client application program.

413 DFHXCTRA_LOAD_FAILED

Explanation: During Initialize_User processing, the external CICS interface attempted to load the trap module (DFHXCTRA). The load of this module has failed.

System Action: The Initialize_User call is rejected. The return code from the MVS load macro (R15) is returned in the subreason field-1. The external CICS interface handles the error, and returns the abend (R0) that would have occurred in the subreason field-2.

User Response: Using the MVS reason code, determine why the load failed. The most likely reason is that DFHXCTRA is not in any library included in the STEPLIB of the batch job. Ensure the library containing the module is included in the STEPLIB concatenation, and restart the client application program.

414 IRP_ABORT_RECEIVED

Explanation: Whilst processing a DPL request, an error occurred in the CICS server region, resulting in an abort FMH7 flow being returned to the external CICS interface.

System Action: A message is returned to the client application program. This is the message that would have been issued to the terminal if the server program had been initiated from a terminal. A pointer to the message is returned to the client application program in the message pointer field of the EXCI return area. See the description of the EXCI return areas for the exact definition of the message format. The pipe is put into a “must close” state.

User Response: Use the message to determine the cause of the error. A typical example is where the server transaction cannot be attached, either because is disabled, or it has not been defined, or because of a security failure. Correct the problem, close and reopen the pipe, and reissue the DPL request.

415 INVALID_CONNECTION_DEFN

Explanation: A DPL request has been rejected by CICS because the target connection is not defined for use by an external CICS client application program.

System Action: The DPL request is rejected and the pipe is put into a “must close” state.

User Response: The most likely reason for this is that the connection definition in the CICS server region has been defined incorrectly as a CICS-to-CICS MRO connection, instead of an EXCI connection. Ensure that PROTOCOL(EXCI) is specified on the appropriate CONNECTION and SESSIONS resource definitions. You must close and reopen the pipe before reissuing the DPL request.

416 INVALID_CICS_RELEASE

Explanation: A DPL request has been rejected by the target CICS server region because it doesn’t recognize the request.

System Action: The DPL call is rejected and the pipe is put into a “must close” state.

User Response: The most likely reason for this is that the client application program has specified a target CICS server region that does not support the external CICS interface. CICS regions earlier than CICS for MVS/ESA 4.1 do not recognize EXCI call requests. Correct the problem, close and reopen the pipe and then reissue the DPL request.

417 PIPE_MUST_CLOSE

Explanation: A DPL request has been issued on a pipe that is in a “must close” state.

System Action: The DPL request is rejected.

User Response: Some EXCI errors are serious enough to require that the pipe be closed and reopened in order to restore the pipe to a point where it can be used for further DPL requests. Others, more minor errors, allow further calls without closing and reopening the pipe. A previous error on this pipe has been of the more serious variety and the pipe is now in a “must close” state. Close and reopen the pipe and reissue the DPL request.

418 INVALID_PIPE_TOKEN

Explanation: An Open_Pipe, Close_Pipe, Deallocate_Pipe, or DPL request has been issued, but the pipe token passed on the call is either not a valid pipe, or is not a valid pipe allocated for this user (that is, there is mismatch between the user token and the pipe token).

System Action: The call is rejected.

User Response: Ensure that the pipe token has not been overwritten and is being passed correctly on the call. Also ensure there is no mismatch between the user token and the pipe token.

419 CICS_AFCB_PRESENT

Explanation: An Initialize_User request has been issued on a TCB that has already been used by CICS or by CICS batch shared database. The external CICS interface cannot share a TCB with CICS, ensuring that a CICS application program cannot issue EXCI requests.

System Action: The Initialize_User request is rejected.

User Response: To use the external CICS interface, you must create a new TCB (or daughter TCB), and issue the EXCI calls under that unique TCB.

420 DFHXCOPT_LOAD_FAILED

Explanation: During Initialize_User processing, the external CICS interface attempted to load its options module, DFHXCOPT. The load of this module failed.

System Action: The Initialize_User call is rejected. The return code from the MVS load macro (R15) is returned in the subreason field-1. The external CICS interface handles the error, and returns the abend (R0) that would have occurred in the subreason field-2.

User Response: Using the MVS reason code, determine why the load failed. The most likely reason is that DFHXCOPT is not in any library included in the

STEPLIB of the batch job. Correct the problem and restart the client application program.

421 RUNNING_UNDER_AN_IRB

Explanation: The EXCI call is issued under an MVS IRB, which is not permitted.

System Action: The call is rejected.

User Response: Determine why the call was issued under an IRB and change the client application program.

422 SERVER_ABENDED

Explanation: Whilst processing a DPL request, the CICS server application program abended without handling the error.

System Action: The server application program is abended and backout out. The abend code is returned in the abend code field of the EXCI return area.

User Response: Determine why the server program abended and fix the problem.

423 SURROGATE_CHECK_FAILED

Explanation: A DPL request has been issued specifying a USERID parameter. The userid specified was subject to a surrogate user security because SURROGCHK=YES is specified in the EXCI options table, DFHXCOPT. The surrogate user check failed. The surrogate security check verifies whether the EXCI batch region's userid is authorized as a surrogate of the userid specified on the DPL call.

System Action: The DPL call is rejected. The MVS external security manager's return code and reason code are returned in subreason field-1 and field-2. For RACF, these documented in the *External Security Interface (RACROUTE) Macro Reference for MVS*, GC23-3733.

User Response: If you want surrogate user checking, ensure that the EXCI batch region's userid has READ access to the profile *userid*.DFHEXCI in the SURROGAT general resource class, where *userid* is the userid specified on the DPL call.

If you don't want surrogate user security checking, specify SURROGCHK=NO in the DFHXCOPT options table.

See "Surrogate user checking" on page 189 for more information.

424 RRMS_NOT_SUPPORTED

Explanation: A DPL request omitting the SYNCONRETURN option has been made on a system that is not running OS/390 Release 5 (or a later, upward-compatible, release).

Reviewer:

Add a reference here to whichever book specifies the pre-req software. Is this the CICS Program Directory?

System Action: The call is rejected.

User Response: Ensure that the batch program is run on a system that is running the correct level of OS/390.

425 UOWID_NOT_ALLOWED

Explanation: A DPL request omitted the SYNCONRETURN option, but specified a value of UOWID. This combination of parameters is not permitted on a DPL request.

System Action: The DPL_Request is rejected.

User Response: Check the client application program and ensure that the correct combination of parameters is used on the DPL call.

Reason codes for response: SYSTEM_ERROR

601 WS_GETMAIN_ERROR

Explanation: During Initialize_User processing, a GETMAIN for working storage failed.

System Action: Processing cannot continue without working storage, so the request is terminated. At this point the external CICS interface trace and dump services are not available to provide diagnostic information, therefore EXCI issues an MVS abend (U0408) to force a SYSMDUMP. The return code from the MVS GETMAIN request is returned in the return area.

User Response: Locate the GETMAIN return code in the dump, and use this and the rest of the dump to determine why the GETMAIN failed. A possible reason for this is that the region size specified for the job is too small. If this is the case, increase the region size and restart the client application program.

602 XCGLOBAL_GETMAIN_ERROR

Explanation: During Initialize_User processing, a GETMAIN failed for a critical control block (XCGLOBAL).

System Action: Processing cannot continue without this control block, and the request is terminated. At this point the external CICS interface trace and dump services are not available to provide diagnostic information, therefore EXCI issues an MVS abend (U0403) to force a SYSMDUMP. The return code from the MVS GETMAIN request is returned in the return area.

User Response: Locate the GETMAIN return code in the dump, and use this and the rest of the dump to determine why the GETMAIN failed. A possible reason for this is that the region size specified for the job is too small. If this is the case, increase the region size and restart the client application program.

603 XCUSER_GETMAIN_ERROR

Explanation: During Initialize_User processing, a GETMAIN request failed for the user control block (XCUSER).

System Action: Initialize_User processing is terminated. The return code from the GETMAIN is returned in subreason field-1 of the return area. The external CICS interface issues message DFHEX0003 and issues an MVS user abend (0410) to force a SYSMDUMP.

User Response: Use the return code from the GETMAIN, with the dump, to determine why the GETMAIN failed. A possible reason for this is that the region size of the job is too small. If this is the case, increase the region size and restart the client application program.

604 XCPIPE_GETMAIN_ERROR

Explanation: During Allocate_Pipe processing, a GETMAIN request for the pipe control block (XCPIPE) failed.

System Action: Allocate_Pipe processing is terminated. The return code from the GETMAIN is returned in subreason field-1 of the EXCI return area. The external CICS interface issues message DFHEX0003, and takes a system dump.

User Response: Use the return code from the GETMAIN, and the dump, to determine why the GETMAIN failed. A possible reason for this is that the region size for the job is too small. If this is the case, increase the region size and restart the client application program.

605 VERIFY_BLOCK_GM_ERROR

Explanation: During Initialize_User processing, a GETMAIN failed for an EXCI internal control block.

System Action: Initialize_User processing is terminated. The return code from the GETMAIN is returned in the subreason field-1 of the EXCI return area. This error occurs before EXCI dumping services are initialized, Therefore EXCI issues an MVS abend (U0409) to force a SYSMDUMP The return code from the MVS GETMAIN request is returned in the return area.

User Response: Locate the GETMAIN return code in the dump, and use this and the rest of the dump to determine why the GETMAIN failed. A possible reason for this is that the region size specified for the job is too small. If this is the case, increase the region size and restart the client application program.

606 SSI_VERIFY_FAILED

Explanation: A VERIFY call to the MVS subsystem interface (SSI) to obtain the current CICS SVC number failed.

System Action: The Initialize_User request is terminated. The return code from the SSI call is returned in subreason field-1 of the return area. This error occurs before the external CICS interface dump services are initialized, therefore EXCI issues an MVS user abend (0405) to force a SYSMDUMP.

User Response: Locate the return code in the dump, and use this with the rest of the dump and SSI documentation to determine why the VERIFY request failed. When the problem is resolved, restart the client application program.

607 CICS_SVC_CALL_FAILURE

Explanation: During Initialize_User processing, a call to the currently installed CICS SVC failed.

System Action: The return code from the CICS SVC is returned in the subreason field-1 of the EXCI return area. This error occurs before the external CICS interface dump services are initialized, therefore EXCI issues an MVS user abend (0406) to force a SYSMDUMP.

User Response: Contact your IBM support center for assistance, with the return code and the dump available.

608 IRC_LOGON_FAILURE

Explanation: During Allocate_Pipe processing, an attempt by the external CICS interface to LOGON to DFHIRP failed.

System Action: The two return codes returned from DFHIRP are returned in the subreason fields. The IRP response code (R15) is in subreason field-1, and the IRP reason code, if any, is given in subreason field-2. The Allocate_Pipe request fails.

User Response: For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPS copybook, listed under the heading IRC. Use the return codes to determine why the logon failed, or contact your IBM support personal with details of the failure.

609 IRC_CONNECT_FAILURE

Explanation: During Open_Pipe processing, an attempt to connect to the target CICS system failed.

System Action: The Open_Pipe request fails. The DFHIRP return code is returned in the EXCI subreason field-1.

User Response: For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPS copybook, listed under the heading IRC.

Use the return code to determine why the logon failed, and reissue the open pipe request.

Note: This error is not caused by the target CICS being unavailable, which is returned as a RETRYABLE condition (NO_CICS).

610 IRC_DISCONNECT_FAILURE

Explanation: During Close_Pipe processing, CICS issued a DFHIRP disconnect call to terminate the connection to CICS. This request has failed.

System Action: The call fails and the pipe is left open. The IRP return code (R15) and any IRP reason code (R0) are returned in the EXCI subreason field-1 and field-2 respectively. The external CICS interface takes a system dump.

Although the disconnect failed, it is possible that the pipe is still connected to CICS. However, all connections are automatically disconnected at the end of the batch program.

User Response: For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPS copybook, listed under the heading IRC. Use the return code and the dump to determine the cause of the error.

611 IRC_LOGOFF_FAILURE

Explanation: During Deallocate_Pipe processing, CICS issued a DFHIRP logoff call. This request failed.

System Action: The Deallocate_Pipe call fails and the pipe remains allocated. The IRP return code (R15) and any IRP reason code (R0) are returned in the EXCI subreason field-1 and field-2 respectively. The external CICS interface takes a system dump.

Note: Because it remains allocated, the pipe is available for further calls. Any storage associated with the pipe is not freed. However, this storage is freed at the end of the client application program.

User Response: For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPS copybook, listed under the heading IRC. Use the return code and the dump to determine the cause of the error.

612 TRANSFORM_1_ERROR

Explanation: During DPL processing, whilst processing the data in preparation for sending to CICS, an internal call to program DFHXFQ resulted in an error.

System Action: The DPL request is terminated.

User Response: The return code from the call is returned in the EXCI subreason field-1, and the external CICS interface takes a system dump.

This is an external CICS interface error. Contact your IBM support center with details of the return code and the dump.

613 TRANSFORM_4_ERROR

Explanation: During DPL processing, whilst processing the data returned by the CICS server region, an internal call to module DFHXFQ resulted in an error.

System Action: The DPL request is terminated. Note that the server application program has executed. The return code from the call to DFHXFQ is returned in the EXCI subreason field-1. This return code corresponds to any EIBRCODE information that was available. The external CICS interface takes a system dump.

User Response: This is an external CICS interface error. Contact your IBM support center with details of the return code and the dump.

614 IRP_NULL_DATA_RECEIVED

Explanation: During DPL processing, a request has been sent to the target CICS and this target CICS has replied without returning any data.

System Action: The DPL processing is terminated and the external CICS interface takes a system dump.

User Response: This is an internal protocol error. Contact your IBM support center with details of the dump.

615 IRP_NEGATIVE_RESPONSE

Explanation: An internal protocol error has occurred whilst trying to communicate with the target CICS region.

System Action: The DPL request fails, the pipe is put into a “must close” state, and the external CICS interface takes a system dump.

User Response: This is an external CICS interface error. Keep the dump and contact your IBM support center.

Note: The pipe is in a “must close” state. Before attempting further calls, the pipe must first be closed and reopened.

616 IRP_SWITCH_PULL_FAILURE

Explanation: An internal protocol error has occurred whilst trying to communicate with the target CICS region.

System Action: The DPL request fails, the pipe is put into a “must close” state, and the external CICS interface takes a system dump. The IRP return code (R15) and reason code if any (R0) are returned in the EXCI subreason field-1 and subreason field-2.

User Response: This is an external CICS interface error. Keep the dump and contact your IBM support center.

Note: The pipe is in a “must close” state, and before attempting further DPL calls, the pipe must first be closed and reopened.

617 IRP_IOAREA_GM_FAILURE

Explanation: During DPL processing, an MVS GETMAIN request for an internal control block failed.

System Action: The DPL request is terminated. The return code from the GETMAIN is returned in the EXCI subreason field-1.

Note: This error occurs whilst processing the data returned by CICS, after the server application program has completed execution. This error results in the pipe being put into a “must close” state.

User Response: Use the return code to determine why the GETMAIN failed. A possible reason for this is that the region size of the job is too small. If this is the case, increase the region size and restart the batch job.

619 IRP_BAD_IOAREA

Explanation: During a DPL request, an I/O area has been supplied to DFHIRP that could not be used.

System Action: The DPL request is terminated, the pipe is forced into a “must close” state, and the external CICS interface takes a system dump.

User Response: This is an external CICS interface error. Contact the IBM support center with details of the return code and the dump.

Note: The pipe is in a “must close” state after this error, and before attempting further calls must first be closed and reopened.

620 IRP_PROTOCOL_ERROR

Explanation: An internal protocol error has occurred whilst trying to communicate with the target CICS system.

System Action: The DPL request is terminated, the pipe is forced into a “must close” state, and the external CICS interface takes a system dump.

User Response: This is an external CICS interface error. Keep the dump and contact your IBM support center.

Note: The pipe is in a “must close” state after this error, and before attempting further calls must first be closed and reopened.

621 PIPE_RECOVERY_FAILURE

Explanation: An error has occurred during an open pipe request. The external CICS interface attempts to recover by disconnecting the pipe again. During this disconnection, further errors have occurred.

System Action: The Open_Pipe call is terminated and the pipe is placed in a “must close” state. The return code from DFHIRP is returned in the EXCI subreason field-1, and a system dump is taken.

User Response: For an explanation of the IRP return codes, see the interregion control blocks in the *CICS Data Areas* manual. The IRP return codes are in the DFHIRSPS copybook, listed under the heading IRC. Use the dump and IRP return codes to determine why the disconnect failed. You may also want to use the EXCI trace to determine the earlier error that caused the open pipe recovery routine to be invoked.

Note: The pipe is now in a “must close” state and if further calls are to be issued, the pipe must be closed and reopened again first.

622 ESTAE_SETUP_FAILURE

Explanation: In order to protect itself from possible program checks the external CICS interface establishes an MVS ESTAE. In this case, the MVS ESTAE macro has failed.

System Action: The call terminated, and the return code from the MVS ESTAE command is returned in the EXCI subreason field-1. This error may occur before EXCI dump services are initialized, therefore an EXCI issues an MVS abend (U0402) to force a SYSMDUMP.

User Response: Use the return code and the dump to determine why the ESTAE command failed. This may be an internal EXCI error and if the problem persists, contact your IBM support center.

623 ESTAE_INVOKED

Explanation: A program check is encountered during call processing, and the ESTAE is invoked.

System Action: The program check is handled by the EXCI ESTAE and an attempt is made to recover to a state that can support further EXCI calls. The MVS abend code is returned in the EXCI subreason field-1 of the return area. To aid further diagnosis, a SYSMDUMP is taken.

User Response: Use the return code and the dump to determine why a program check occurred in the external CICS interface. The most likely reason for this is that the EXCI code abended whilst trying to access the client program's parameters. Use the EXCI trace to determine if any of the parameters might have caused this error. If this is not the case, this may be an error in the external CICS interface. Keep the dump and contact your IBM support center.

624 SERVER_TIMEDOUT

Explanation: A DPL request has been issued and the target server program has executed in the CICS server region. However, the server program has been executing for longer than the time-out value specified in the DFHXCOPT table.

System Action: The external CICS interface stops waiting for the server program to complete. Because the server program might complete some time after the time-out, and try to respond to the DPL call, the pipe is forced into a "must close" state.

User Response: Determine why the server application program timed out. Either there is a problem with the server program itself (for example, it might be in a loop), or the timeout value is too low.

625 STIMER_SETUP_FAILURE

Explanation: In order to provide a TIMEOUT mechanism, the external CICS interface issues an MVS STIMERM macro call. This call has failed.

System Action: The return code from the call is returned in the subreason field-1 of the EXCI return area. The DPL request is terminated and the external CICS interface takes a system dump. The pipe is placed in a "must close" state.

User Response: Use the MVS return code and the dump to determine why the call failed. This could be an external CICS interface error. Contact your IBM support center with details of the dump.

Note: The pipe is in a "must close" state after this error, and before attempting further calls must first be closed and reopened.

626 STIMER_CANCEL_FAILURE

Explanation: On successful completion of a DPL request, the cancel of an STIMERM request issued to check the TIMEOUT value has failed with an error.

System Action: The return code from the STIMERM CANCEL is returned in the subreason field-1 of the EXCI return area. The pipe is placed in a "must close" state, and the external CICS interface takes a system dump.

User Response: Use the return code and the dump to determine why the MVS STIMERM CANCEL command failed. This could be an external CICS interface error. Contact your IBM support center with details of the dump.

Note: The pipe is in a "must close" state after this error, and before attempting further calls must first be closed and reopened.

627 INCORRECT_SVC_LEVEL

Explanation: The release level of the CICS SVC (DFHCSVC) is not the same (or higher) than the release level of the external CICS interface.

System Action: The Initialize_User request is terminated. This error occurs before the external CICS interface SDUMP facilities are initialized, therefore EXCI issues an MVS abend (U0407) to force a SYSMDUMP.

User Response: Determine the level of the CICS SVC being used and ensure it is the same release level as the external CICS interface, or higher. If the SVC number is allowed to default (CICSSVC=0 in DFHXCOPT), the SVC number being used is the SVC first used by a CICS region on the MVS image. That is, the SVC used by the first CICS region to open the CICS interregion communications (IRC). If the SVC number is specified on CICSSVC in DFHXCOPT, the SVC number specified is at an incorrect level. For more information, see the description of the CICSSVC parameter in "Chapter 13. External CICS interface options table, DFHXCOPT" on page 167.

628 IRP_LEVEL_CHECK_FAILURE

Explanation: The release level of the module DFHIRP is not at the same, or higher, level than the release level of the external CICS interface.

System Action: The Allocate_pipe request is terminated. The IRP return code (R15) is returned in the EXCI subreason field-1, and the function level of DFHIRP being used is returned in the EXCI subreason field-2. Subreason field-2 is only meaningful if subreason field-1 is zero. The external CICS interface takes a system dump.

User Response: Check the level of the DFHIRP module installed in the LPA. Ensure that it is at least the same as the external CICS interface. The installed level of DFHIRP must be the highest level of CICS or external CICS interface in use in the MVS image. For more details about installing DFHIRP, see the *CICS Transaction Server for OS/390 Installation Guide*.

629 SERVER_PROTOCOL_ERROR

Explanation: A response to a DPL request has been returned by CICS but the external CICS interface does not understand the response.

System Action: The DPL request is terminated and the external CICS interface takes a system dump.

User Response: Use the dump to determine why the response was in error. The most likely reason for this is that the CICS application server program was not running under the control of a CICS mirror task. This can happen if the transaction definition named by the transid parameter on the DPL call does not specify DFHMIRS as the program name. This would cause unidentified responses being sent from the CICS server region.

630 RRMS_ERROR

Explanation: An unexpected return code was received from Recoverable Resource Management Services (RRMS) while processing a DPL_Request.

System Action: DPL_Request processing is terminated.

The value in subreason field-1 of the return area indicates which RRMS interface returned the unexpected return code:

1	CTXRCC
2	ATRRURD
3	CTXSDTA

The return code from the RRMS request is returned in subreason field-2.

The external CICS interface issues message DFHEX0002, and takes a system dump.

User Response: Use the return code from the RRMS request and the dump, to determine why the request failed. This may be an internal EXCI error or a problem with RRMS and you may need the assistance of your IBM support center.

631 RRMS_SEVERE_ERROR

Explanation: During the processing of a DPL_Request, the EXCI code encountered an unexpected error while using its interface with Recoverable Resource Management Services (RRMS).

System Action: DPL_Request processing is terminated.

The external CICS interface issues message DFHEX0002, and takes a system dump.

User Response: Use the dump, to determine why the request failed. This may be an internal EXCI error and you may need the assistance of your IBM support center.

632 XCGUR_GETMAIN_ERROR

Explanation: During DPL_Request processing, a GETMAIN request for working storage for module DFHXCGUR failed.

System Action: DPL_Request processing is terminated.

The return code from the GETMAIN is returned in subreason field-1 of the return area. The external CICS interface issues message DFHEX0003, and takes a system dump.

User Response: Use the return code from the GETMAIN, and the dump, to determine why the GETMAIN failed. A possible reason is that the region size of the job is too small. If this is the case, increase the region size and restart the client application program.

Chapter 18. Messages and Codes

This chapter lists messages and abend codes for the external CICS interface.

DFHEX0001 An abend (code *aaa/bbbb*) has occurred in module *modname*.

Explanation: An unexpected program check or abend *aaa/bbbb* has occurred in module *modname*. This implies that there may be an error in external CICS interface code.

Alternatively, unexpected data has been passed on an external CICS interface call or storage has been overwritten.

The code *aaa/bbbb* is, if applicable, a 3-digit hexadecimal MVS system completion code *aaa* (for example, 0C1 or D37). If an MVS code is not applicable, this field is filled with three hyphens. The 4-digit code *bbbb*, which follows *aaa* is, if applicable, a user abend code produced by the external CICS interface. If the user abend code is not applicable, this field is filled with four hyphens.

System Action: An exception entry is made in the external CICS interface internal trace table, and to the GTF trace dataset (if GTF is active), and a SYSMDUMP is taken.

The external CICS interface terminates the current request, and attempts to recover to a consistent state so that further EXCI requests can be serviced. For an application using the EXCI CALL API, a response of EXCI_SYSTEM_ERROR with a REASON of ESTAE_INVOKED is returned to the application. For an application using the EXCI EXEC API, an EXEC_RESP of LINKERR is returned to the application, together with an EXEC_RESP2 of ESTAE_INVOKED or EXEC_ESTAE_INVOKED, depending on whether the call level ESTAE routine, or the EXEC level ESTAE routine was invoked.

User Response: Look up the MVS code *aaa*, if there is one, in the relevant MVS codes manual which is detailed in the book list in the front of this manual.

If applicable, see the description of abend code *bbbb* for further guidance.

You may need further assistance from IBM to resolve this problem. See Part 4 of the *CICS Problem Determination Guide* for guidance on how to proceed.

Destination: Console

Module: DFHXCPRH, DFHXCEIP

DFHEX0002 A severe error (code *X'code'*) has occurred in module *modname*.

Explanation: An error has been detected in module *modname*. The code *code* is the exception trace point ID which uniquely identifies what the error is and where the error was detected.

System Action: An exception entry is made in the EXCI internal trace table and to GTF if it is active, (*X'code'* in the message). A system dump is taken.

This is a critical error and the EXCI request is terminated. The external CICS interface attempts to recover to a consistent state so that further EXCI requests can be issued. For applications using the EXCI CALL API, the EXCI_REASON returned to the application indicates the reason for the error. For applications using the EXCI EXEC API, the reason is returned in the EXEC_RESP2 field of the RETCODE area.

User Response: This failure indicates a serious error in the external CICS interface code. For further information about the EXCI exception trace entries, refer to the *CICS Problem Determination Guide*.

You need further assistance from IBM to resolve this problem. See *CICS Problem Determination Guide* for guidance on how to proceed.

Destination: Console

Module: DFHXCPRH, DFHXCEIP

DFHEX0003 A GETMAIN request in module *modname* (code *X'code'*) has failed. Reason *X'rc'*.

Explanation: An MVS GETMAIN was issued by module *modname*, but it failed with return code *rc*.

The code *code* is the exception trace point ID which uniquely identifies the place where the MVS GETMAIN was issued.

System Action: An exception entry is made in the EXCI internal trace table (code *code* in the message). This is a critical error and the EXCI request is terminated. The external CICS interface attempts to recover to a consistent state so that further EXCI requests can be issued.

For applications using the EXCI CALL API, the EXCI_REASON returned to the application indicates the point of failure.

For applications using the EXCI EXEC API, the point of failure is returned in the EXEC_RESP2 field of the RETCODE area.

For EXCI_REASON and EXCI_RESP of 603, the EXCI module DFHXCPRH also issues abend 0410 which drives the ESTAE exit. Message DFHEX0001 is issued and a SYSMDUMP is taken

© Copyright IBM Corp. 1994, 1998

User Response: Look up the MVS GETMAIN return code *rc* in the relevant MVS codes manual.

If the reason is insufficient storage, try increasing the size of the region for the batch EXCI job.

You may need further assistance from IBM to resolve this problem. See the *CICS Problem Determination Guide* for guidance on how to proceed.

Destination: Console

Module: DFHXCPRH, DFHXCTRI

DFHEX0100 The installed level of CICS SVC does not support the EXCI call.

Explanation: The external CICS interface module DFHXCPRH detected that the level of CICS SVC (DFHCSVC) in use does not support the external CICS interface.

System Action: The EXCI request is terminated. An exception trace is made in the EXCI internal trace table, and if GTF is active, in the GTF trace data set. The external CICS interface module DFHXCPRH issues abend 0407 which drives the ESTAE exit. Message DFHEX0001 is issued, and a SYSMDUMP is taken.

User Response: Check the level of DFHCSVC installed in the LPA, which Generally, the latest level of DFHCSVC must be used when running CICS and the external CICS interface. For more information about installing DFHCSVC see the *CICS Transaction Server for OS/390 Installation Guide*.

Destination: Console

Module: DFHXCPRH

DFHEX0101 Unable to start interregion communication because DFHIRP services are down level.

Explanation: The version of DFHIRP being used is at a lower level than that of the External CICS Interface (EXCI) module DFHXCPRH.

System Action: The EXCI allocate pipe request is rejected, and a return code passed back to the batch application.

User Response: Update the level of the DFHIRP module in the LPA such that it matches the level of the latest CICS version in use.

Destination: Console

Module: DFHXCPRH

DFHEX0110 EXCI SDUMP has been taken. Dumpcode: *dumpcode***, Dumpid:** *dumpid***.**

Explanation: This message is issued on successful completion of a MVS SDUMP issued by external CICS interface module DFHXCDMP. An error, signalled by a previous message, caused a call to be made to DFHXCDMP to take a system dump.

The dump code *dumpcode* is an 8-character system dump code identifying the external CICS interface problem. A system dump code is the EXCI message number with the DFH prefix removed.

dumpid is the unique 9-character string identifying this dump.

System Action: The EXCI request is terminated.

User Response: See the EXCI message indicated by *dumpcode* for further guidance.

You may need further assistance from IBM to resolve this problem. See the *CICS Problem Determination Guide* for guidance on how to proceed.

Destination: Console

Module: DFHXCDMP.

DFHEX0111 EXCI SDUMP attempted but SDUMP is busy - will retry every five seconds for *nnnn* **seconds.**

Explanation: At the time of the MVS SDUMP request issued by DFHXCDMP another address space in the same MVS system was in the process of taking an SDUMP. This causes MVS to reject the new request. A nonzero value for the dump retry parameter in the DFHXCOPT table means that the external CICS interface waits five seconds before retrying the SDUMP request. If necessary, the external CICS interface retries every five seconds for the total time specified on the retry parameter.

System Action: The external CICS interface issues an MVS STIMERM macro which causes it to wait for five seconds. The request is reissued when the delay interval has expired.

User Response: None.

Destination: Console

Module: DFHXCDMP.

DFHEX0112 SDUMP request failed - *reason X'nn'***.**

Explanation: An MVS SDUMP request issued from the external CICS interface has failed to complete successfully. The possible reasons, (*reason*) for the failure are as follows:

ONLY PARTIAL DUMP
The SYS1.DUMP data set to which the dump is written is not large enough to contain all of the dumped storage.

SDUMP BUSY
At the time of the MVS SDUMP request issued by the EXCI, another address space in the same MVS system was in the process of

taking an SDUMP. This causes MVS to reject the new request. If a nonzero value is specified for the dump retry parameter in DFHXOPTS table, the EXCI has retried the SDUMP request every five seconds for the specified period. This message is only issued if SDUMP is still busy after the final retry.

STIMERM FAILED
In order to delay for five seconds before retrying SDUMP after an SDUMP BUSY condition, the EXCI issues an MVS STIMERM macro request. MVS has indicated that the STIMERM request has failed.

NO DATA SET AVAILABLE
No SYS1.DUMP data sets were available at the time the SDUMP request was issued.

REJECTED BY MVS, REASON = X'nn'
MVS has rejected the SDUMP request because of user action (for example, specifying DUMP=NO in the MVS IPL) or because of an I/O error or terminating error in the SDUMP routine. X'nn' is the SDUMP reason code.

NOT AUTHORIZED FOR EXCI
SDUMP is not authorized for the external CICS interface.

INSUFFICIENT STORAGE
The EXCI issued an MVS GETMAIN for subpool 253 storage during the processing of the SDUMP request. The GETMAIN has been rejected by MVS.

System Action: The EXCI proceeds as if the dump had been successful.

User Response: The user response depends on the reasons, (*reason*), for the failure.

ONLY PARTIAL DUMP
Increase the size of the SYS1.DUMP data sets and cause the SDUMP request to be reissued.

SDUMP BUSY
Cause the SDUMP to be reissued after, if appropriate, increasing the dump retry time in DFHXCOPT.

STIMERM FAILED
Use MVS problem determination methods to fix the STIMERM failure and then cause the SDUMP request to be reissued.

NO DATA SET AVAILABLE
Clear a SYS1.DUMP data set and then cause the SDUMP request to be reissued.

REJECTED BY MVS, REASON = X'nn'
No action is required if the dump is suppressed deliberately. If the dump has failed because of an error in the MVS SDUMP routine, use MVS problem determination methods to fix the error and then cause the SDUMP request to be reissued. See the *OS/390 MVS Programming: Assembler Services Reference*, GC28-1910, for an explanation of the SDUMP reason code X'nn'.

NOT AUTHORIZED FOR EXCI
This reason is unlikely because SDUMP is unconditionally authorized during EXCI initialization, and should be authorized throughout the EXCI run. If you do get this reason, the EXCI AFCB (authorized function control block) has probably been accidentally overwritten.

INSUFFICIENT STORAGE
Ensure sufficient storage is available to MVS for subpool 253 requests.

Destination: Console

Module: DFHXCDMP

DFHEX0113 EXCI trace Initialization has failed.

Explanation: An attempt to initialize external CICS interface (EXCI) trace facilities during EXCI initialization has failed.

System Action: The EXCI request continues without trace facilities. An earlier message identifies the cause of the failure.

User Response: Refer to the earlier message to determine the cause of the failure.

Destination: Console

Module: DFHXCTRI

DFHEX0114 Incorrect data has been passed for EXCI tracing causing a program check in DFHXCTRP.

Explanation: Some data passed to the external CICS interface (EXCI) trace module DFHXCTRP for addition to the EXCI internal trace table, or GTF trace, caused a program check to occur when an attempt was made to access it.

The most likely cause of this error is incorrect data passed on an EXCI CALL API request that the trace program DFHXCTRP is attempting to access.

System Action: The EXCI request is terminated and a SYSMDUMP is taken.

User Response: Examine the dump to determine the source of the incorrect data.

You may need further assistance from IBM to resolve this problem. See the *CICS Problem Determination Guide* for guidance on how to proceed.

Destination: Console

Module: DFHXCTRI

DFHEX0115 EXCI trace services have been disabled due to a previous error.

Explanation: An error occurred in the external CICS interface (EXCI) trace module DFHXCTRP indicated by message DFHEX0001. In trying to recover from the error, module DFHXCTRI determined that the error was not caused by accessing incorrect data passed to DFHXCTRP, but was due to a program check in DFHXCTRP.

System Action: The EXCI trace facilities are disabled to prevent further errors. A SYSMDUMP is taken.

User Response: See the DFHEX0001 message and the SYSMDUMP to determine the cause of the error.

You may need further assistance from IBM to resolve this problem. See the *CICS Problem Determination Guide*A for guidance on how to proceed.

Destination: Console

Module: DFHXCTRI

DFHEX0116 Program check occurred within global trap exit - DFHXCTRA now marked unusable.

Explanation: After making a trace entry, the external CICS interface (EXCI) trace program DFHXCTRP called the EXCI field engineering global trap program DFHXCTRA. A program check occurred during execution of DFHXCTRA.

System Action: The EXCI marks the currently active version of DFHXCTRA as unusable and ignores it on subsequent calls to DFHXCTRP for all subsequent calls made under this TCB. The EXCI request is terminated, and a SYSMDUMP is taken.

User Response: Use the dump to find the cause of the program check.

You may need further assistance from IBM to resolve this problem. See the *CICS Problem Determination Guide* for guidance on how to proceed.

You should use the global trap exit only in consultation with an IBM support representative.

Destination: Console

Module: DFHXCTRI

Part 4. CICS ONC RPC support

This part describes the support in CICS for Open Network Computing Remote Procedure Call (ONC RPC) clients.

This part contains:

© Copyright IBM Corp. 1994, 1998

Chapter 19. Introduction to ONC RPC

This chapter gives a summary of CICS ONC RPC, ONC RPC, and TCP/IP. It contains the following sections:

CICS ONC RPC—a brief summary

CICS ONC RPC allows client applications to access CICS programs by calling them as remote procedures using the ONC RPC format.

CICS ONC RPC can be used:

- To allow clients to use existing CICS programs and the transaction processing services they provide
- To allow clients to use newly created CICS programs

TCP/IP for MVS is a prerequisite for CICS ONC RPC; it provides the library code for Sun Microsystems’ ONC RPC Version 3.9. Hence, CICS ONC RPC servers work with any remote client compatible with ONC RPC Version 3.9, regardless of operating system or machine type. See the *TCP/IP for MVS: Programmer’s Reference* for information about the function of ONC RPC Version 3.9 supported by TCP/IP for MVS.

Figure 41 shows how CICS ONC RPC allows a variety of client applications to communicate with CICS programs using ONC RPC.

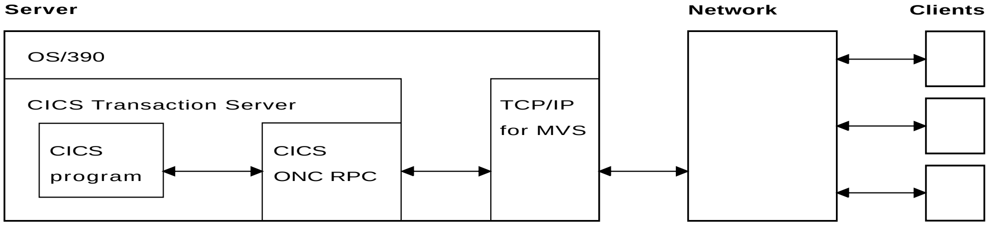


Figure 41. How CICS ONC RPC might be used

The CICS program called to service a client request is executed by a transaction that has no principal facility. It is therefore not allowed to use some commands of the CICS application programming interface:

- Terminal control commands that reference the principal facility
- Options of EXEC CICS ASSIGN that return terminal attributes
- BMS commands
- Sign-on and sign-off commands.

© Copyright IBM Corp. 1994, 1998

Introduction to ONC RPC

This section introduces the basics of ONC RPC operation, its place in TCP/IP networks, and how its main facilities work. It does not cover all aspects of ONC RPC or TCP/IP, only those that relate to CICS ONC RPC.

CICS ONC RPC

In the rest of this chapter, boxes like this point out how CICS ONC RPC implements the area of ONC RPC being described in the text.

RPC

When a process invokes or calls a process on a remote system, that call is a *remote procedure call (RPC)*. The calling process is a *client* (that is, a process requesting a service); the remote process is a *server* (a process offering a service). As shown in Figure 42, the client sends a request for a procedure to be run, and supplies parameters for that particular run. Once the server has run the procedure, it returns the reply.

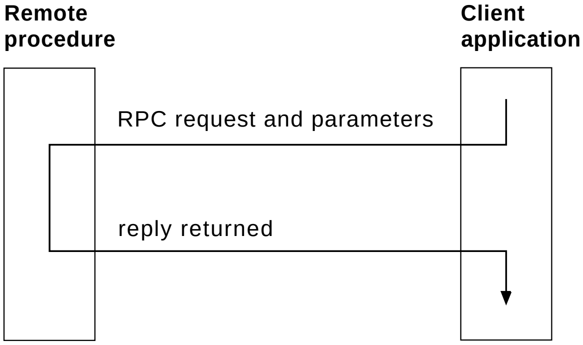


Figure 42. Basic RPC operation

In the RPC model, there is no provision for coordinating changes to recoverable resources in different servers, nor for coordinating changes to recoverable resources in successive calls to the same server. Committing changes to recoverable resources is under the control of the remote procedure, not the client application.

Several RPC implementations have been developed and are now available on a variety of systems. RPC allows a programmer to network an application by distributing the procedures that make up the application across different processors. This is done without the programmer becoming involved with the details of the communication interface required to transmit the parameters to and from the remote procedures.

ONC

ONC is Open Network Computing, a range of software developed by Sun Microsystems. As well as the ONC RPC routines, Sun provides XDR (eXternal Data Representation) routines, which are used for data conversion. The ONC RPC and XDR protocols and formats are supported on many different platforms.

CICS ONC RPC

CICS ONC RPC allows users to run only ONC RPC servers under CICS hosts. It does not support client applications running under CICS.

TCP/IP

ONC RPC applications use the TCP/IP family of protocols. See “TCP/IP protocols” on page 8 for more information about TCP/IP.

ONC RPC facilities

The ONC RPC implementation consists of:

- **XDR routines**. XDR library functions are supplied to enable conversion to and from the standard data format for transmission.
- **RPCGEN compiler**. This takes a user-written definition of a remote procedure interface and generates the required parts of the application that deal with the RPC interface.
- **ONC RPC API library**.

XDR routines

Data exchanged between systems engaged in ONC RPC must always flow in a standard format specified by XDR, because different machine architectures have different representations of the same information.

Both client and server use *XDR routines* to convert the input and output parameters between XDR format and the local data format. You either write these yourself, or specify an XDR library function, as described below. In Figure 43, **inproc** and **outproc** are the XDR routines.

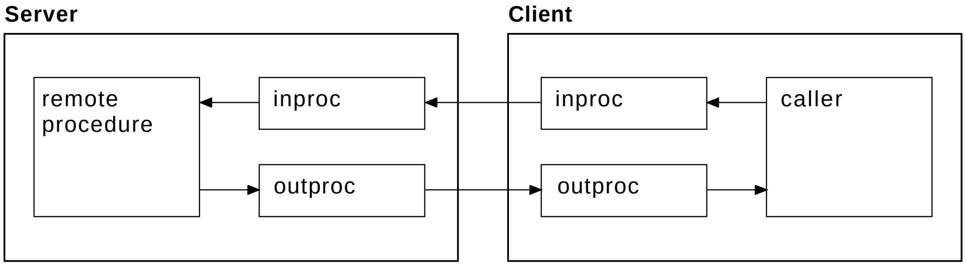


Figure 43. XDR routines used in a remote procedure call

Notice that in Figure 43, the same XDR routine, **inproc**, is used to encode and decode the data as it flows from client to server, and similarly for **outproc** as it flows back to the client. The source for **inproc** is the same in the client and server, but XDR library functions in the routines are compiled to encode or decode as appropriate. Such routines are termed *bidirectional*, and they help to ensure that the encoding and decoding is done symmetrically in the two routines.

Using XDR library functions

XDR library functions are a set of C functions supplied with ONC RPC, which application programmers can use when writing XDR routines. They can be used as follows, depending on the complexity of the structure pointed to by the call argument and reply parameters.

For parameters that are simple single-field C data types
Use an XDR library function for **inproc** and **outproc**.

For parameters that are C data type vectors, arrays, strings, and so on
Use an XDR library function for **inproc** and **outproc**.

For more complex structures
Write an XDR routine, using XDR library functions as required. Alternatively, use the RPCGEN compiler, described in “RPCGEN compiler”, to create an XDR routine from an XDR data description.

CICS ONC RPC

CICS ONC RPC supports the use of the XDR library functions that support data conversion.

RPCGEN compiler

To use RPCGEN, you write a program definition in RPCL, a language similar to a subset of C, designed for the definition of ONC RPC distributed programs. The definition defines the data to be transferred and procedures to be used for both client and server. The client application source program is written as though the remote procedure call were a call to a local program. The code to send the call and get the reply are part of the client stub, which is generated by RPCGEN. Similarly the code the server needs to accept the call and send back the reply are part of the server stub, which is also generated by RPCGEN. Figure 44 on page 233 illustrates the role of RPCGEN in application development.

CICS ONC RPC

RPCGEN may only be used for:

- Generating pairs of XDR routines, as described in the previous section
- Generating a client stub to be linked with the application for the client system
- Generating header files

CICS ONC RPC does not use the server stub generated by RPCGEN.

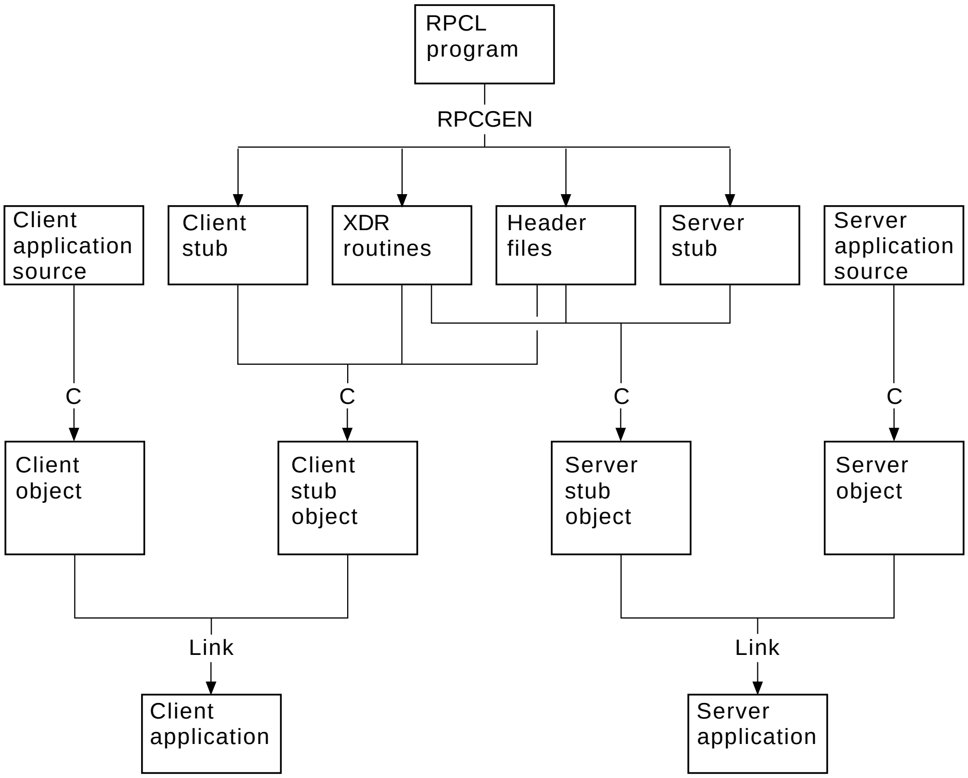


Figure 44. Using the RPCGEN compiler

ONC RPC API library

The ONC RPC API library contains two types of call: high level and low level.

The high-level ONC RPC API can be used only with UDP. It enables users to make remote procedure calls very simply and with a minimum of library calls, but at a cost of some restriction in available function. The main function of the API is provided by three calls:

registerrpc
Used in the server to register a procedure to be called as a remote procedure by clients.

svc_run
Used in the server to see if a request has arrived from a client.

callrpc
Used in the client to make a remote procedure call.

The low-level ONC RPC API contains many more calls, which give more control and flexibility. For example:

- Low-level calls give the user the choice of transport below ONC RPC, including TCP or UDP.
- With low-level calls, user-written network registration services other than the Portmapper (the Portmapper is described below) can be used.
- Low-level calls allow the variation of ONC RPC time-outs and retry values.

- Low-level calls allow standard ONC RPC authorization to be applied. Only UNIX authorization is available in ONC RPC Version 3.9.

CICS ONC RPC

CICS ONC RPC provides all the server function. You don’t specify any server RPC calls.

The client can make its request with the high-level call **callrpc**, or can use low-level calls. CICS ONC RPC is implemented using low-level ONC RPC calls. The implementation allows concurrent dispatching of individual procedures and allows TCP to be supported as well as UDP.

ONC RPC naming and routing

Remote procedures in ONC RPC are identified by the 3-tuple: program number, version number, and procedure number.

It is usual to package several related procedures together into a single program. When changes are made to the procedures, a new version of the program is created, but the new version usually contains the same procedure numbers as the previous version.

Procedure zero

Users define procedure numbers for each program, conventionally starting at 1 and proceeding in sequence. Procedure 0 is usually defined as a procedure with no parameters and no processing that returns an empty reply. This is useful for clients, who can call procedure 0 to see if a particular service exists and to test performance on a null call.

Registration and the Portmapper

Servers on a host need to let clients know their logical addresses and which services they offer. In ONC RPC, servers generally do this by *registering* with a utility service called the *Portmapper*. This maintains a list of mappings from program/version numbers (also qualified by protocol used) to TCP/IP port numbers on a host.

The Portmapper itself can always be located by clients because it is always on well-known port 111 on a given host. If using low-level calls, the client first asks the Portmapper for the port number for the particular remote procedure, and then calls that port directly. The high-level call, **callrpc**, performs the same function transparently to the user.

CICS ONC RPC

Registration is done by CICS ONC RPC automatically, or under operator control.

Routing

Before calling a procedure, a client asks the Portmapper at the host for the port number of the program and version that the client wishes to call. (The protocol is determined when the connection between TCP/IP systems is set up.) In the remote procedure call, the client supplies only the IP address, port number, and procedure number. Figure 45 shows how the IP address, port number, and procedure number identify the server procedure.

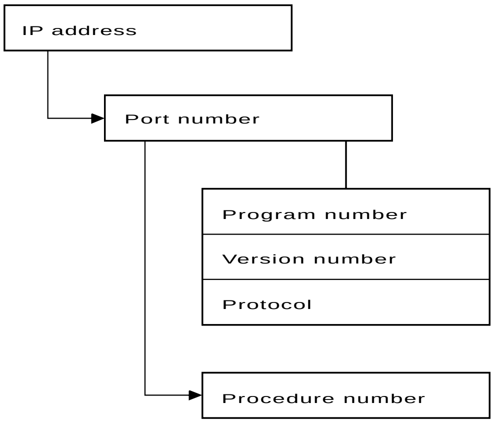


Figure 45. TCP/IP and RPC routing

Types of remote procedure call

Synchronous
This is the normal method of operation. The client makes a call and does not continue until the server returns the reply.

Nonblocking
The client makes a call and continues with its own processing. The server does not reply.

Batching
This is a facility for sending several client nonblocking calls in one batch.

Broadcast RPC
RPC clients have a broadcast facility, that is, they can send messages to many servers and then receive all the consequent replies.

Callback RPC
The client makes a nonblocking client/server call, and the server signals completion by calling a procedure associated with the client.

CICS ONC RPC

CICS ONC RPC cannot support callback RPC, because callback requires that both ends contain both client and server procedures.

Chapter 20. Overview of CICS ONC RPC

This chapter describes the CICS ONC RPC operating environment, and how you can control and customize it. It contains the following sections:

- “ONC RPC remote procedures and CICS programs”
- “CICS ONC RPC transactions” on page 238
- “User-replaceable programs” on page 239
- “Control flow in a client request” on page 241
- “Data flow in a client request” on page 242

ONC RPC remote procedures and CICS programs

This section describes how CICS programs are identified by ONC RPC.

Identifying the CICS program

In CICS ONC RPC, the CICS programs are identified by a 4-tuple.

- Program number—same as the ONC RPC program number
- Version number—same as the ONC RPC version number
- Procedure number—same as the ONC RPC procedure number
- Protocol—determined by the protocol used to communicate between the client system and TCP/IP for MVS.

When a client request arrives, the CICS program chosen to service it is the one associated with the 4-tuple just described. Figure 46 shows a state of CICS ONC RPC in which five 4-tuples are associated with three CICS programs.

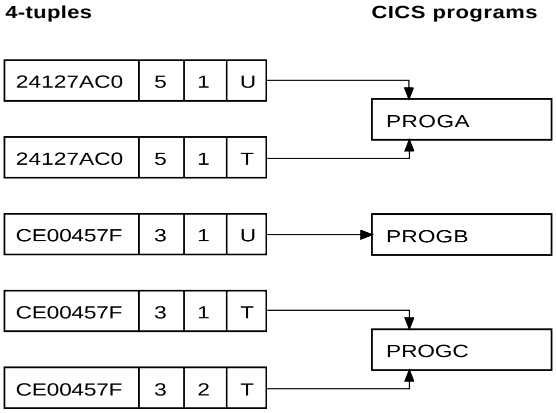


Figure 46. Remote procedures and CICS programs

The program numbers are given in hexadecimal. The protocols are U for UDP and T for TCP.

- If a client request arrives for program 24127AC0, version 5, procedure 1, the CICS program PROGA is used to service it whether the protocol is TCP or UDP.

© Copyright IBM Corp. 1994, 1998

- If a request arrives for program CE00457F, version 3, procedure 1, and the protocol is UDP, the CICS program PROGB is used to service it. But if the same request arrives and the protocol is TCP, PROGC is used to service it.

  It is, however, usual to use the same program, version, and procedure irrespective of the protocol used to transmit the request.
- The CICS program PROGC is also used for procedure 2 of the same program and version if the protocol is TCP.

How you set up and control the relationship between 4-tuples and CICS programs is described in “Chapter 22. Configuring CICS ONC RPC using the connection manager” on page 253.

Where the CICS program might be

The CICS program might be in one of three places:

- In the same CICS region as CICS ONC RPC
- In a different CICS region on the same host
- On a different host that supports CICS and inbound DPL

The CICS programs can reside on any CICS system accessible by means of DPL from the CICS region running CICS ONC RPC. DPL operation is described in the *CICS Intercommunication Guide*.

CICS ONC RPC transactions

Three CICS transactions are supplied with CICS ONC RPC:

- Connection manager
- Server controller
- Alias

Connection manager (CRPC)

The connection manager is a transaction that allows you to enable and disable CICS ONC RPC, and configure and inquire on it. You run the connection manager transaction as required, and several instances of it can be active at the same time. The connection manager is described in “Chapter 22. Configuring CICS ONC RPC using the connection manager” on page 253.

Server controller (CRPM)

The server controller monitors the TCP/IP for MVS interface for client requests, and starts instances of the alias transaction, using EXEC CICS START, to service them. The server controller is a transaction of long duration. It is started by the connection manager when CICS ONC RPC is enabled, and stopped when CICS ONC RPC is disabled. Only one instance of the server controller can be active in a CICS system.

Alias (CRPA)

CICS ONC RPC supplies one alias *program*. Multiple instances of the alias *transaction* can be run in parallel, each in response to a client request.

An alias is started by the server controller for each client request that arrives to be processed, as shown in Figure 47. This allows CICS ONC RPC to process many client requests concurrently.

The alias program uses EXEC CICS LINK to transfer control to the CICS program.

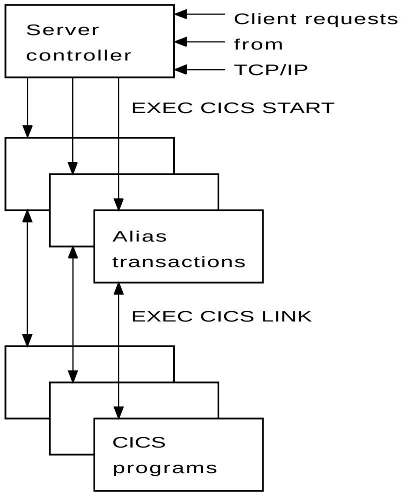


Figure 47. The server controller and alias transactions

User-replaceable programs

Servicing a client request involves not only a CICS program, but a converter program and XDR routines. For compatibility with earlier releases of CICS you can use a resource checker program to validate incoming client requests, or you can use CICS security facilities.

XDR routines

XDR (eXternal Data Representation) is described in “XDR routines” on page 231.

You need to provide one or two XDR routines for each 4-tuple. You always need an inbound XDR routine, and unless the client call is nonblocking, you need an outbound XDR routine as well.

The XDR routines for each 4-tuple are specified by using the connection manager.

Resource checker module

CICS ONC RPC provides an interface to a resource checker (which you write). The module can be used to validate incoming client requests. It is described in “Writing the resource checker” on page 307.

Converters

You can also supply a converter for each program-version-procedure-protocol 4-tuple. Each converter can contain up to three functions.

- **Getlengths function**. The **Getlengths** function might be called by the connection manager when a 4-tuple is registered. **Getlengths** can supply the following information:
  - – The length of the input and output data for the CICS program
  - – Whether the output data overlays the input data in the communication area

  Because its processing is done before any client requests are received, It is appropriate to use **Getlengths** to provide the values of data lengths that do not vary from call to call. Refer to “Lengths of the CICS program input and output data” on page 283 for a fuller description of when **Getlengths** should be used for this purpose.
- **Decode function**. The **Decode** function is called by the server controller on receipt of a client request. **Decode** can do the following:
  - – Supply the length of the input and output data for the CICS program. If the parameter lengths vary from call to call, **Decode** should return them for the current call.
  - – Reconstruct the data from the client as a communication area for the CICS program. “Data flow in a client request” on page 242 illustrates the kinds of data that **Decode** might have to handle. The incoming data might include pointers, and **Decode** must gather up the incoming data into the communication area.
  - – Convert data structures from C format to the format appropriate to the programming language in which the CICS program is written.
  - – Process data from the client that is not intended for the CICS program. For example the data from the client might include the name of the CICS program to be called, and **Decode** can feed this information back to the server controller.
- **Encode function**. The **Encode** function is called by the alias when the CICS program ends. **Encode** can do the following:
  - – Reconstruct the data from the communication area into the form expected by the client. “Data flow in a client request” on page 242 illustrates the kinds of data that **Encode** might have to handle. The client might expect to receive data accessed by pointers, and **Encode** must build this structure from the data in the communication area.
  - – Convert data structures from the format appropriate to the programming language in which the CICS program is written into C format.

Not all 4-tuples need a converter with all three functions. You use the connection manager to specify the converter and the use of **Getlengths**, **Decode**, and **Encode** for each 4-tuple.

The way that particular language data structures are stored is documented in the appropriate language manuals, and a correspondence between C data types and those in other languages is given in the *IBM SAA AD/Cycle Language Environment/370 Programming Guide*.

For detailed instructions on the writing of converters, refer to “Step 4—Write the converter” on page 283.

Control flow in a client request

Figure 48 shows the components involved in processing a typical client ONC RPC request.

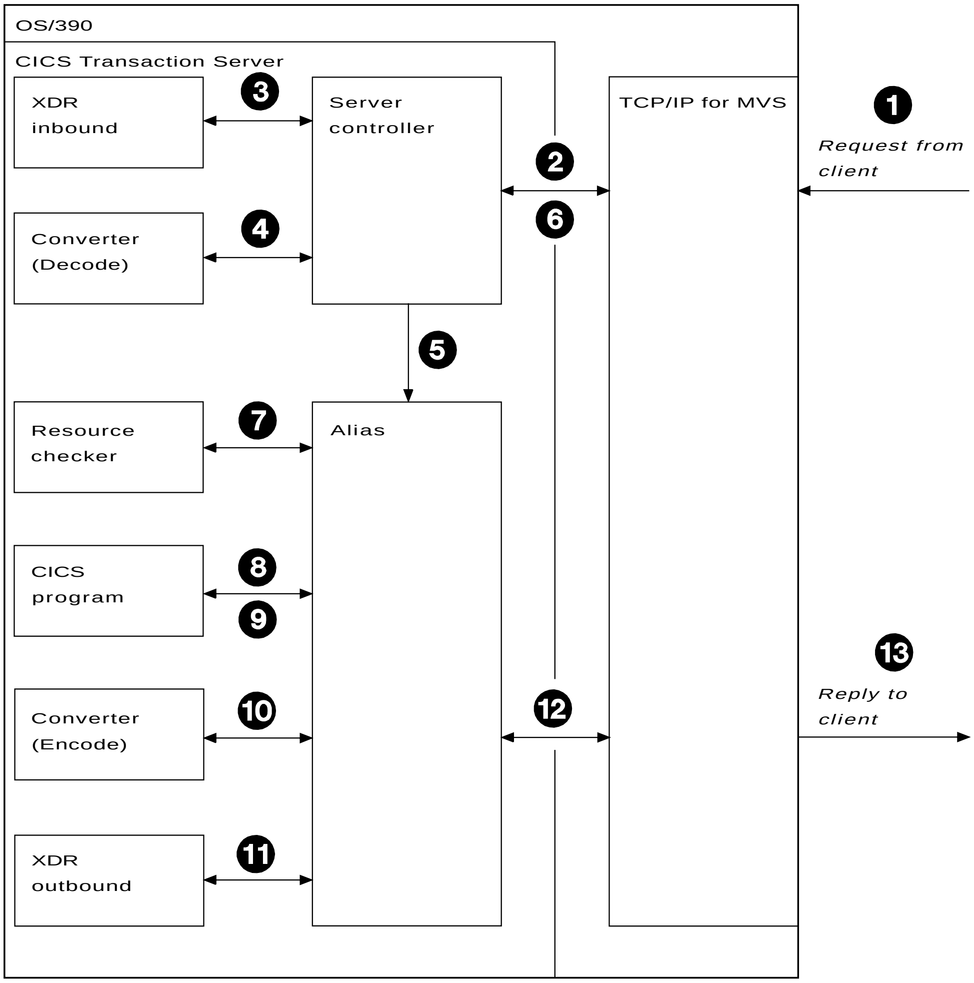


Figure 48. Call processing

Client requests are processed in the following steps:

1. A request from a client arrives in TCP/IP for MVS.
2. The server controller monitors the TCP/IP for MVS interface for incoming client requests, and the client request is passed to it. (From the 4-tuple for the request, the server controller can find the corresponding XDR routine and converter to call.)
3. The server controller invokes the inbound XDR routine.
4. The server controller calls the converter, requesting the **Decode** function, if it is required for the 4-tuple. If **Decode** is not required, the server controller allocates storage for the CICS program communication area.
5. The server controller then starts an alias to deal with all further processing of the request within CICS.
6. The server controller returns to monitor the TCP/IP for MVS interface for client requests.

7. The alias optionally calls a user-written resource checker.
8. The alias issues an EXEC CICS LINK to the CICS program for the 4-tuple. The communication area set up by **Decode** is passed in the LINK command.
9. The CICS program processes the request and returns its output to the alias program in the communication area.
10. The alias calls the **Encode** function, if it is required for the 4-tuple.
11. The alias invokes the outbound XDR routine.
12. The alias returns the reply to TCP/IP for MVS, and ends.
13. The reply is sent back to the client.

Updating recoverable resources

After **Decode** processing, the server controller uses EXEC CICS SYNCPOINT to commit any changes to recoverable resources that **Decode** might have made.

If the CICS program makes updates to recoverable resources, whether the changes are committed or backed out depends on the location of the CICS program, and on whether it uses the EXEC CICS SYNCPOINT command.

- If the CICS program is in a CICS region different from the one in which CICS ONC RPC is operating, the updates are committed when the CICS program returns control to the alias.
- If the CICS program is in the same CICS region as CICS ONC RPC, and it uses EXEC CICS SYNCPOINT, the updates are committed when the syncpoint is processed.
- If the CICS program is in the same CICS region as CICS ONC RPC, but it does *not* use EXEC CICS SYNCPOINT, the updates are committed when the alias transaction ends normally, or are backed out when the alias transaction abends.

Data flow in a client request

This section describes data flow from a client to a CICS program, and from a CICS program back to the client.

From client to CICS program

Figure 49 on page 243 shows the progress of data from the client to the CICS program during a remote procedure call.

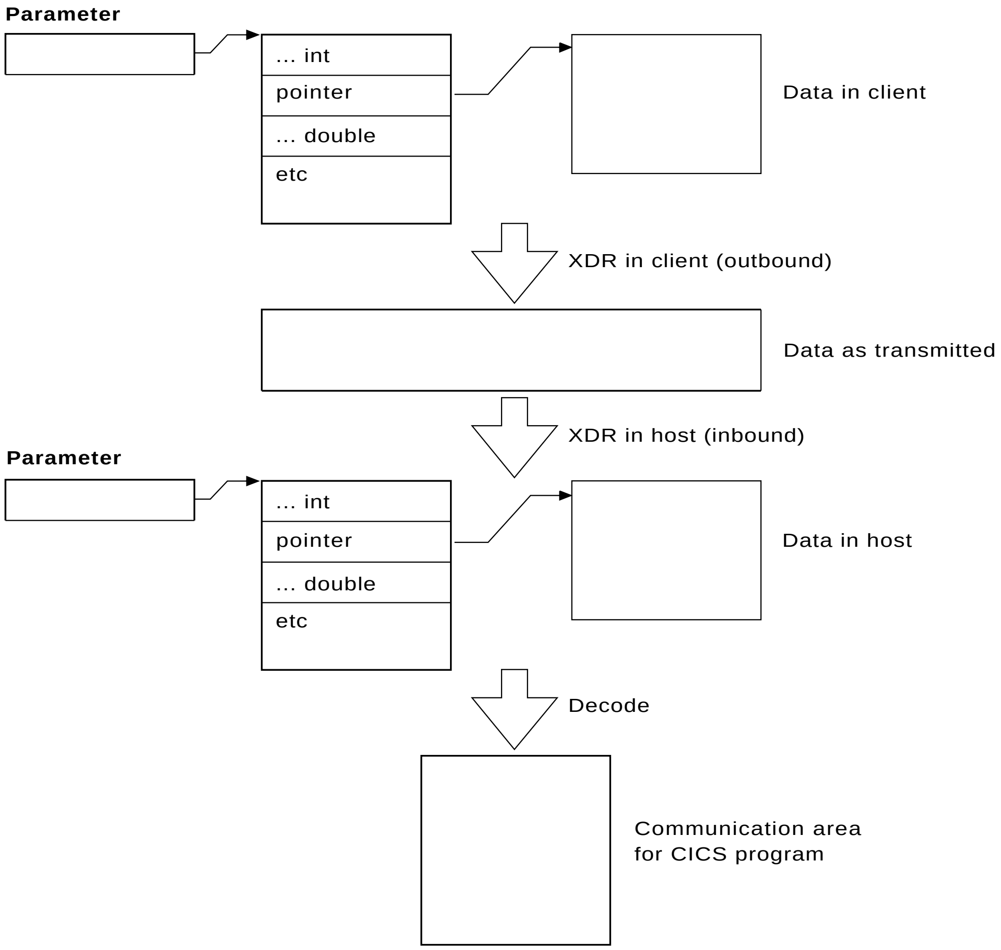


Figure 49. Data flow from client to CICS program

In this example the processing is as follows:

1. The client call has a parameter which includes a pointer to data that is to be passed to the CICS program. The client's outbound XDR routine packages the parameter and the indirect data for transmission to the host.
2. The data is transmitted over the network to the host.
3. In the host, the inbound XDR routine rebuilds the data as it was in the client.
4. The **Decode** function of the converter reorganizes the data into a communication area for the CICS program.

Data format in the CICS program communication area

If the call is a blocking call, the position in the CICS program's communication area of data to be returned to the client has to be specified. The data in the CICS program's communication area can be organized in two ways:

- Contiguous—the data to be returned to the client does not start at the beginning of the communication area, but at some offset into it.
- Overlaid—the data to be returned starts at the beginning of the communication area. The CICS program overwrites the inbound client data in this area with any data to be returned to the client.

Figure 50 illustrates these two possibilities.

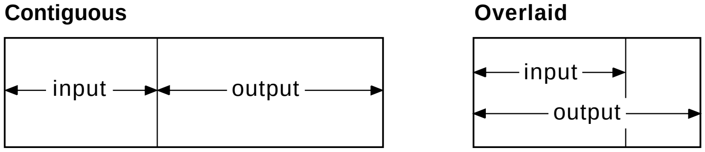


Figure 50. Use of communication area according to data format

From CICS program to client

Figure 51 shows the progress of data from the CICS program back to the client.

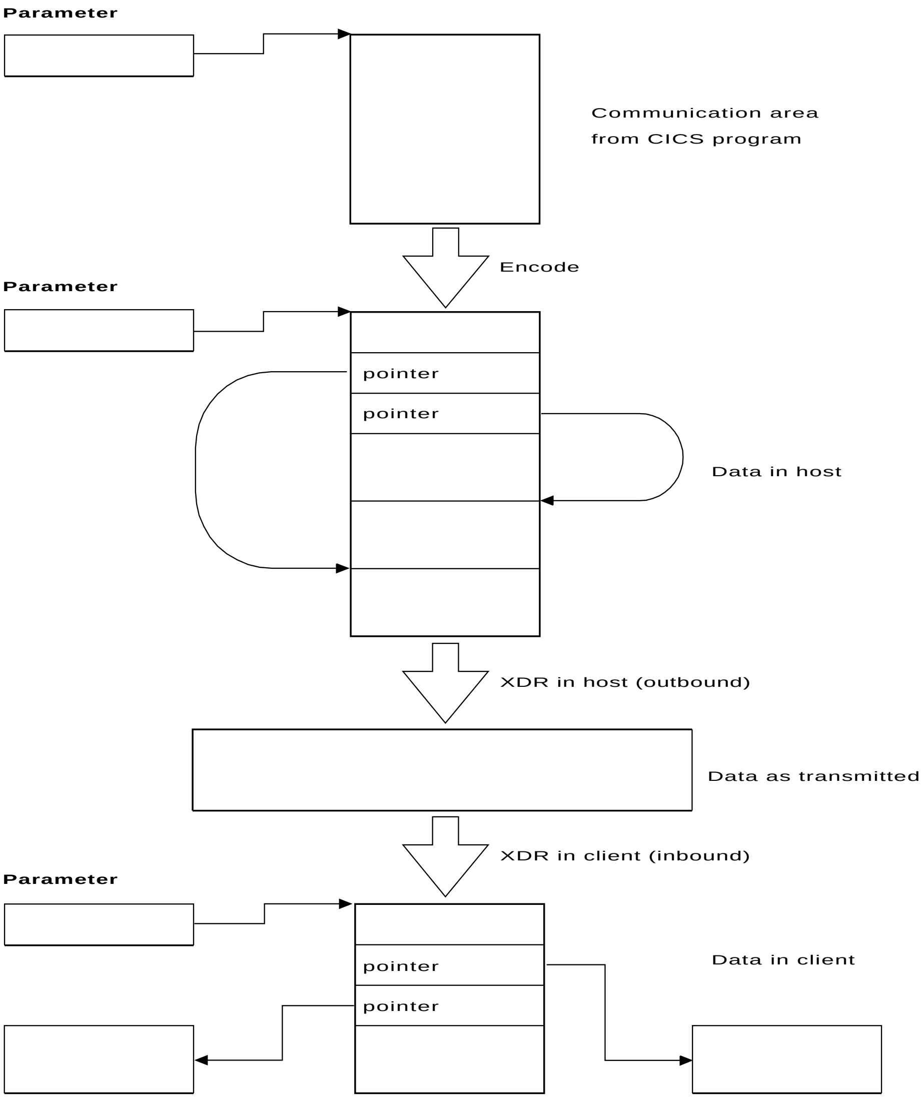


Figure 51. Data flow from CICS program to client

The processing is as follows:

1. The CICS program's output is in the communication area that was created by the **Decode** function. The **Encode** function reorganizes the data in the manner

that the client expects. In this case the client is expecting to get back a structure including two pointers to indirect data. The **Encode** function puts the data in a single area of storage to simplify storage management processing when the area is to be freed.

2. The outbound XDR routine packages the data for transmission.
3. The data is transmitted over the network to the client.
4. In the client, the inbound XDR routine rebuilds the data as it was in the host.

Chapter 21. Setting up CICS ONC RPC

This chapter describes how to set up your CICS system to exploit CICS ONC RPC. It contains the following sections:

- “Prerequisites for using CICS ONC RPC”
- “CICS ONC RPC setup tasks” on page 248
- “Defining CICS ONC RPC resources to CICS” on page 249
- “Modifying TCP/IP for MVS data sets” on page 252

Prerequisites for using CICS ONC RPC

This section discusses software prerequisites and storage requirements.

Clients

Clients must access servers on CICS ONC RPC over a TCP/IP network.

Client systems must use a library compatible with the library for ONC RPC Version 3.9, as this is the ONC RPC version supported by TCP/IP for MVS (Versions 2.2.1 and 3.1). To communicate over a TCP/IP network, appropriate hardware and software must be in place.

MVS

The following items are prerequisite, that is, must be installed on the MVS system for CICS ONC RPC to run.

- TCP/IP for MVS Version 2.2.1 or above. TCP/IP for MVS ports must be made available for use by the CICS region involved.
- Language Environment/370. This provides the C run-time libraries that are a prerequisite for running CICS ONC RPC.
- If you are using RPCGEN, or writing your own XDR routines, you need a C compiler to compile RPCGEN output and your XDR routines.

CICS

CICS must be set up for Language Environment/370 support, as described in the *CICS System Definition Guide* and in the *Language Environment/370 Installation and Customization Guide*.

Note: TCP/IP for MVS CICS Sockets is not a prerequisite for CICS ONC RPC.

TCP/IP for MVS

CICS ONC RPC and TCP/IP for MVS CICS Sockets Version 2.2.1 cannot operate together from one CICS region to one TCP/IP for MVS region. You are advised to run CICS Sockets and CICS ONC RPC in different CICS regions.

TCP/IP for MVS Version 3.1 users do not have this problem; CICS Sockets and CICS ONC RPC can both be run from the same CICS region.

© Copyright IBM Corp. 1994, 1998

TCP/IP for MVS 2.2.1

There are no prerequisites for running CICS ONC RPC.

Note: CICS ONC RPC and TCP/IP for MVS CICS Sockets Version 2.2.1 cannot operate together from one CICS region to one TCP/IP for MVS region. You are advised to run CICS Sockets and CICS ONC RPC from different CICS regions.

TCP/IP for MVS 3.1

The following PTF is a prerequisite for running CICS ONC RPC:

- A PTF, number UN79963, related to the use of the **xdr_text_char** XDR library function.

Note: CICS ONC RPC and TCP/IP for MVS CICS Sockets Version 2.2.1 cannot operate together from one CICS region to one TCP/IP for MVS region. You are advised to run CICS Sockets and CICS ONC RPC from different CICS regions.

Storage requirements

Except where otherwise noted, the storage used by CICS ONC RPC is obtained from CICS subpools.

When CICS ONC RPC is enabled, its storage requirements are as follows:

- 40 KB base storage
- 100 bytes for each registered 4-tuple.

For each client request being processed the following storage is required:

- MVS-controlled storage used by the inbound XDR routine for internal data structures
- Storage used by the inbound XDR routine for the data structure it builds for the **Decode** function
- Storage for the CICS program communication area
- Storage used by the alias transaction while running the CICS program
- Storage used by the **Encode** function to create a data structure for the outbound XDR routine
- MVS-controlled storage used by the outbound XDR routine

CICS ONC RPC setup tasks

There are tasks associated with the CICS ONC RPC data set, dump formatting, and a warning about migration.

Creating the CICS ONC RCP data set

JCL is provided in the DFHCOMDS job to create the CICS ONC RPC data set. The data set is defined as a VSAM key-sequenced data set by a DEFINE CLUSTER command like the following:

```
DEFINE CLUSTER (                    -
  NAME( xxxxxxxx.CICSONC.RESOURCE ) -
  CYL ( 2 1  )                      -
  KEYS( 19 0 )                      -
  INDEXED                           -
  VOLUME ( vvvvvv )                 -
  RECORDSIZE( 150 150 )             -
  FREESPACE( 5 5 )                  -
  SHAREOPTIONS( 1 )                 -
  )
```

The job to define the data set must be run before you start the connection manager for the first time.

JCL entry for dump formatting

To switch dump formatting on for CICS ONC RPC (and for all running features), change the IPCS VERBEXIT control statement as follows:

```
IPCS VERBEXIT CICS530 FT=2
```

The VERBEXIT provides a formatted dump of CICS ONC RPC control blocks.

Migrating between CICS versions

CICS ONC RPC was a separately-installed feature of CICS/ESA 3.3 and CICS for MVS/ESA 4.1, but is part of the CICS Transaction Server base. None of the IBM-supplied programs for CICS ONC RPC should be moved to CICS Transaction Server from earlier releases.

Defining CICS ONC RPC resources to CICS

CICS ONC RPC provides two RDO groups defining CICS resources used by CICS ONC RPC: DFHRP and DFHRPF.

Transaction definitions for CICS ONC RPC transactions

The following CICS ONC RPC transactions are defined in the locked group DFHRP:

CRPA Alias

CRPC Connection manager

CRPM Server controller

These definitions cannot be changed.

Transaction definitions for extra alias transactions

You may want to use other alias transaction names for various reasons:

- Auditing purposes
- Resource and command checking
- Allocating initiation priorities
- Allocating database plan selection
- Assigning different runaway values for different CICS programs

If you do, you must also define these to CICS, copying the definition from CRPA, and making amendments as necessary. The CRPA definition is as follows:

```
DEFINE  TRANSACTION(CRPA)       GROUP(DFHRP)
        PROGRAM(DFHRPAS)        TWASIZE(0)
        PROFILE(DFHCICST)       STATUS(ENABLED)
        TASKDATALOC(BELOW)      TASKDATAKEY(USER)
        RUNAWAY(SYSTEM)         SHUTDOWN(ENABLED)
        PRIORITY(1)             TRANCLASS(DFHTCL00)
        DTIMOUT(NO)             INDOUBT(BACKOUT)
        SPURGE(YES)             TPURGE(NO)
        RESSEC(NO)              CMDSEC(NO)
```

If you want a CICS program to run under an alias with a name other than CRPA, you can enter this in the connection manager when defining the attributes of the 4-tuple associated with the CICS program, as described in “Defining the attributes of a 4-tuple” on page 262. The name of the alias can also be changed by the **Decode** function, as described in “Changing the alias and CICS program” on page 284.

Changing the CMDSEC and RESSEC values

You might wish to define new alias transactions with CMDSEC(YES) or RESSEC(YES) in order to enforce security checking on the programs run under the alias transaction, including the CICS program that services the client request. The effect of these options is described in “Security in CICS and its effect on CICS ONC RPC operations” on page 305. None of the IBM-supplied programs used by the alias use any of system programmer interface (SPI) commands, so CMDSEC need not be changed. However, if you wish to oversee the use of SPI commands by the CICS program, resource checker, or **Encode** function of the converter, CMDSEC(YES) is required.

Program definitions for CICS ONC RPC programs

All the CICS ONC RPC programs are defined in the locked group DFHRP.

Program definitions for user-written programs

You need to make definitions for:

- CICS programs
- Converters
- User-written XDR routines
- Resource checker

LANGUAGE option

User-written XDR routines should be defined with LANGUAGE(C). Converters and CICS programs should be defined with an appropriate LANGUAGE.

CEDF option

Program definitions for CICS programs must include CEDF(YES) if EDF is required for debugging. If you wish to use EDF, you must enter a terminal ID in the connection manager when defining the attributes of the 4-tuple associated with the CICS program, as described in “Defining the attributes of a 4-tuple” on page 262.

EXECKEY option

CICS operates with storage protection only if the SIT parameter STGPROT is set to YES, and the system has the required hardware and software.

Converters and the resource checker should not be regarded as application programs when defining storage. You are recommended to define them as EXECKEY(CICS). This allows them to modify CICS-key storage.

When the **Decode** and **Encode** functions allocate storage to hold the converted data, that storage should be allocated as CICS-key.

User-written XDR routines must be defined as EXECKEY(CICS).

CICS programs should be defined as EXECKEY(USER), unless there is some reason for defining them as CICS-key in your CICS system. Defining programs as EXECKEY(USER) prevents them from overwriting CICS.

If you specify EXECKEY(USER) for the CICS program, ensure that TASKDATAKEY(USER) is specified for the alias. USER is the default TASKDATAKEY setting in the alias definition in the supplied group DFHRP.

If you have CICS programs that need to be specified with EXECKEY(CICS), you are advised to specify TASKDATAKEY(CICS) for the alias that will execute them.

RELOAD option

You should specify RELOAD(YES) for any user-written XDR routines to prevent errors in CICS ONC RPC disable processing.

Definitions for remote CICS programs

If a CICS program that is to service a remote procedure call runs in a different CICS system from CICS ONC RPC, a program definition is required on both the local system and the remote system. The program resides on the remote system, so its definition there is straightforward. The program definition on the local system:

- Must include a REMOTESYSTEM parameter to specify the system on which the program resides.
- Can optionally include a REMOTENAME parameter if you want the names on the local system and remote system to be different.
- Can optionally include a TRANSID parameter:
  – If TRANSID is not specified, the CICS program runs under the CICS mirror transaction on the remote CICS system.
  – If TRANSID is specified, the program in the remote CICS system runs under the transaction name given. See “Transaction definitions for extra alias transactions” on page 249 for reasons why you may want a different name.

    If the remote transaction ID is specified, you must provide a matching transaction definition in the remote CICS system. This definition must specify the appropriate mirror program for the remote system (DFHMIRS for CICS for MVS/ESA and CICS Transaction Server for OS/390 systems).

If a CICS program is running on a CICS platform other than CICS for MVS/ESA or CICS Transaction Server for OS/390 similar considerations apply, but you should refer to the DPL details for that platform.

Mapset definition

Mapset definitions are supplied in the group DFHRP for the connection manager mapsets. The definitions cannot be changed.

Transient data definitions

You must supply DCT definitions for the CICS ONC RPC message transient data queue. The following sample is supplied with CICS ONC RPC as member DFHRPDCT of the SDFHSAMP target library:

```
DFHDCT TYPE=SDSCI,
       BLKSIZE=137,
       RECSIZE=133,
       DSCNAME=CRPO,
       RECFORM=VARUNB,
       BUFNO=1,
       TYPEFLE=OUTPUT

DFHDCT TYPE=EXTRA,
       DESTID=CRPO,
       DSCNAME=CRPO
```

If you define the destination in the manner of the sample, you must also add a suitable DD statement for the extrapartition queue in the CICS JCL, for instance:

```
//CRPO   DD   SYSOUT=A
```

The destination could also be made intrapartition or indirect.

XLT definitions

The XLT system initialization parameter and its associated transaction list should allow the connection manager, CRPC, to be started during normal CICS shutdown. If CICS ONC RPC is delaying shutdown, the connection manager can be used to force an immediate disable of CICS ONC RPC.

Modifying TCP/IP for MVS data sets

You can define the CICS TS region to TCP/IP for MVS in the tcpip.PROFILE.TCPIP data set to reserve specific ports for ONC RPC applications (described in *TCP/IP for MVS: Customization and Administration Guide*).

Chapter 22. Configuring CICS ONC RPC using the connection manager

This chapter describes how you use to connection manager to control CICS ONC RPC. It contains the followgin

What the connection manager is for

The operating environment of CICS ONC RPC has two aspects:

- Operating options—tracing, and other problem determination techniques.
- 4-tuple information—determines which client requests can be processed and what CICS resources are needed.

The CICS ONC RPC data set is a store of operating environment information. It contains two kinds of records: the CICS ONC RPC definition record contains the operating options, and 4-tuple records contain the 4-tuple information.

The connection manager has four main functions:

- Enabling CICS ONC RPC
- Disabling CICS ONC RPC
- Controlling the operating options and 4-tuple information stored in the CICS ONC RPC data set
- Controlling the operating options and 4-tuple information in current use when CICS ONC RPC is enabled

When CICS ONC RPC is disabled

When CICS ONC RPC is disabled, the connection manager allows you to:

- Create or update the CICS ONC RPC definition record in the data set
- Add, delete, and change 4-tuple records in the data set
- Enable CICS ONC RPC

You can use the connection manager to enable CICS ONC RPC in two ways:

- Operator-assisted enable—before you enable CICS ONC RPC, you can:
  - Modify any or all of the options

© Copyright IBM Corp. 1994, 1998

  - Select which 4-tuples are to be registered
  - Modify the attributes of 4-tuples before registration

  When you enable CICS ONC RPC, options to control its operation come into play, and 4-tuples can be registered.

  The changes you make during an operator-assisted enable can be temporary, lasting only until the next time you disable CICS ONC RPC, or you can store them into the CICS ONC RPC data set, and use them the next time you enable CICS ONC RPC.
- Automatic enable—the contents of the CICS ONC RPC definition record determine the options to control the operation of CICS ONC RPC until the next time you disable it. Some 4-tuples might be registered, depending on an attribute in the 4-tuple definition.

When CICS ONC RPC is enabled

When CICS ONC RPC is enabled, the connection manager allows you to:

- Update the CICS ONC RPC definition record in the data set
- Add, delete, and change 4-tuple records in the data set
- Change the options being used to control the operation of CICS ONC RPC
- Register 4-tuple definitions from the data set
- Create temporary 4-tuple definitions and register them
- Unregister 4-tuple definitions
- Disable CICS ONC RPC

There are two ways of disabling CICS ONC RPC: normal, and immediate. The effects of disable processing are described in “Disabling CICS ONC RPC” on page 270 .

Starting the connection manager

You can start the connection manager in various ways:

- From a terminal that supports BMS maps. You can work with the connection manager panels described in this chapter.
- From a CICS console.
- Using an EXEC CICS START command.
- From a sequential terminal.

The effect of starting the connection manager depends on:

- Whether CICS ONC RPC is enabled or disabled
- Whether you start the connection manager from a terminal that permits the use of BMS
- Whether you enter additional data with the transaction name
- Whether the Automatic Enable option in the CICS ONC RPC definition record is set to YES

When CICS ONC RPC is disabled, the effect of entering the transaction name (and optional additional data) on a terminal that supports BMS is as follows:

CRPC

- If Automatic Enable is YES, automatic enable processing occurs.
- If Automatic Enable is NO, a BMS panel (DFHRP01) is shown.
- If there is no CICS ONC RPC definition record yet, a BMS panel (DFHRP01) is shown.

CRPC E A(N)

- A BMS panel (DFHRP01) is shown.

CRPC E A(Y)

- Automatic enable processing occurs. If there is no CICS ONC RPC definition record, one is created using default values for the options, but no 4-tuples are registered.

If you start the connection manager in a way that does not allow panels to be shown (EXEC CICS START, or non-BMS terminal, for example) and the action is to show a panel, error message DFHRP1505 is produced.

When CICS ONC RPC is enabled, the effect of entering the transaction name (and optional additional data) is as follows:

- CRPC displays panel DFHRP04, or produces error message DFHRP1505 if panels cannot be shown.
- CRPC D(N) causes normal disable processing.
- CRPC D(I) causes immediate disable processing.

The forms CRPC E A(N), CRPC E A(Y), CRPC D(N), and CRPC D(I) are called fast-path commands.

TCP/IP for MVS should be started before you try to enable CICS ONC RPC with the connection manager, otherwise you cannot register 4-tuples, and you have to reenable CICS ONC RPC after starting TCP/IP for MVS.

Using the connection manager BMS panels

All leading and trailing blanks are ignored on BMS input.

At the top of all panels is a panel identifier in the right corner (for example, DFHRP02) and CRPC in the left corner.

On the bottom of all panels, the fourth line from the bottom gives the status of CICS ONC RPC, the third line from the bottom is a prompt line, while the bottom line lists the available PF keys, which can include:

PF1 Help information (all panels)

PF2 Delete definition from the CICS ONC RPC data set (only where shown)

PF3 Exit CRPC (you are prompted to confirm by using PF3 again)

PF4 Write fields to the CICS ONC RPC data set (only where shown)

PF7 Scroll up (only where shown)

PF8 Scroll down (only where shown)

PF9 Display messages relating to current input

PF12 Cancel this panel and return to the previous panel

Connection manager error message output

The destination of connection manager messages depends on the nature of the message:

- Severe errors requiring operator intervention are sent to the console. No other messages go to the console.
- Messages relating to invalid input on the panel can be displayed by pressing PF9.
- Messages reporting internal errors are sent to CRPO, and in most cases they can be displayed on the terminal by pressing PF9.

Using PF9 to display messages

During the operation of the connection manager, error messages might be issued. These are not displayed immediately on the screen, but a prompt appears on the prompt line to say that messages are waiting to be viewed. To see the messages, press PF9. The number and text of the messages is displayed. You can look up the messages in *CICS Messages and Codes* for more information about errors, and for advice about what to do next.

When you have read the messages, you can press Enter, PF3, or PF12 to return to the input panel.

Starting the connection manager when CICS ONC RPC is disabled

If CICS ONC RPC is disabled, panel DFHRP01 is shown. (See Figure 52 on page 257.)

Select an option, then press Enter.

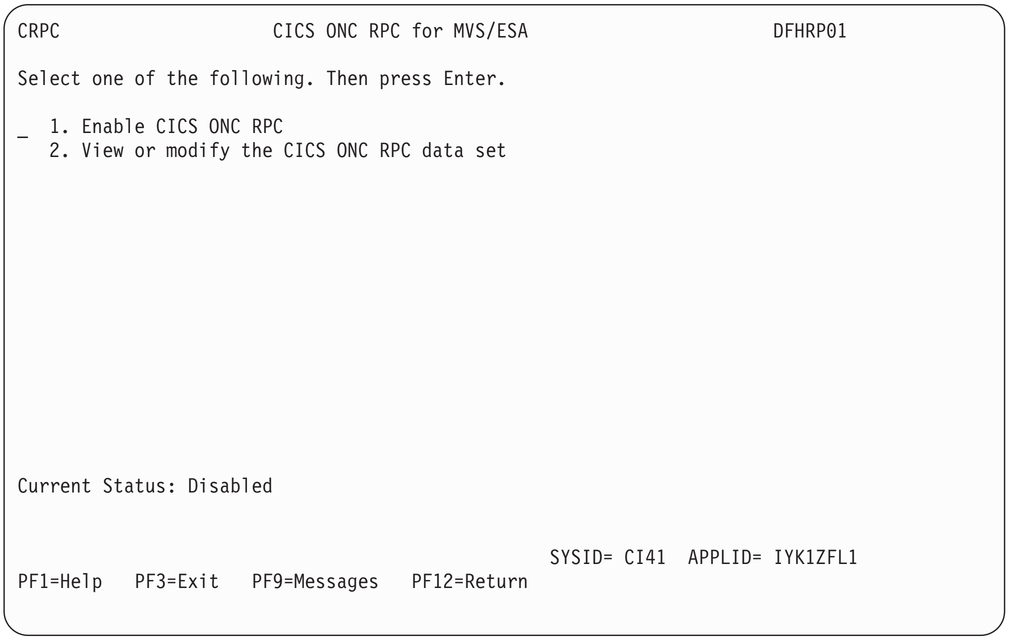
Option	**For more information see:**
1	“Enabling CICS ONC RPC” on page 259
2	“Updating the CICS ONC RPC data set” on page 272

Starting the connection manager when CICS ONC RPC is enabled

If CICS ONC RPC is enabled, panel DFHRP04 is shown. (See Figure 53 on page 257.)

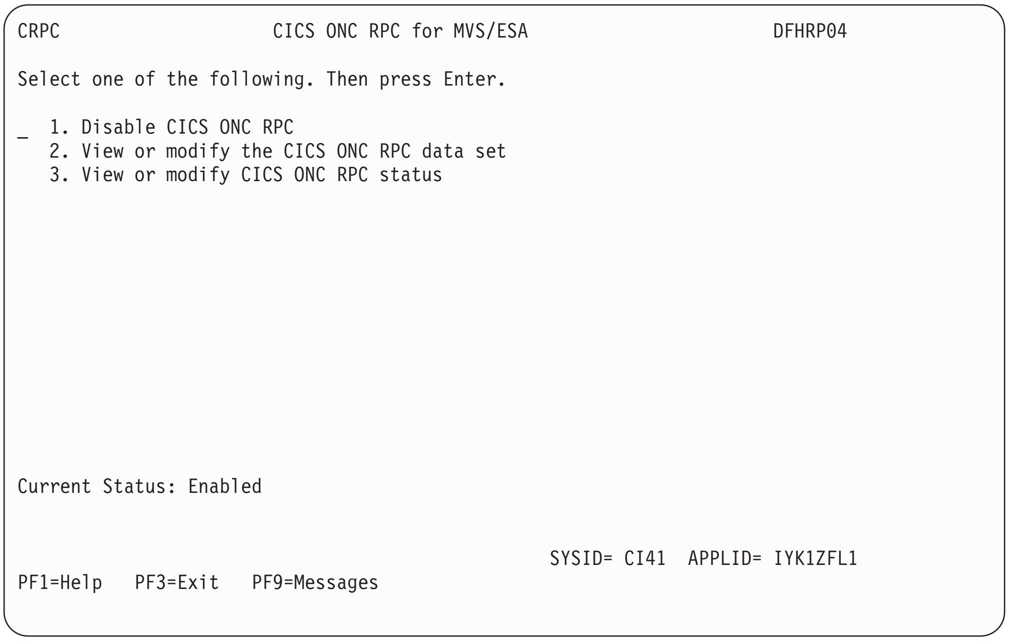
Select an option, then press Enter.

Option	**For more information see:**
1	“Disabling CICS ONC RPC” on page 270
2	“Updating the CICS ONC RPC data set” on page 272
3	“Updating CICS ONC RPC status” on page 257



```
CRPC                    CICS ONC RPC for MVS/ESA                       DFHRP01

Select one of the following. Then press Enter.

_  1. Enable CICS ONC RPC
   2. View or modify the CICS ONC RPC data set















Current Status: Disabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 52. Panel DFHRP01

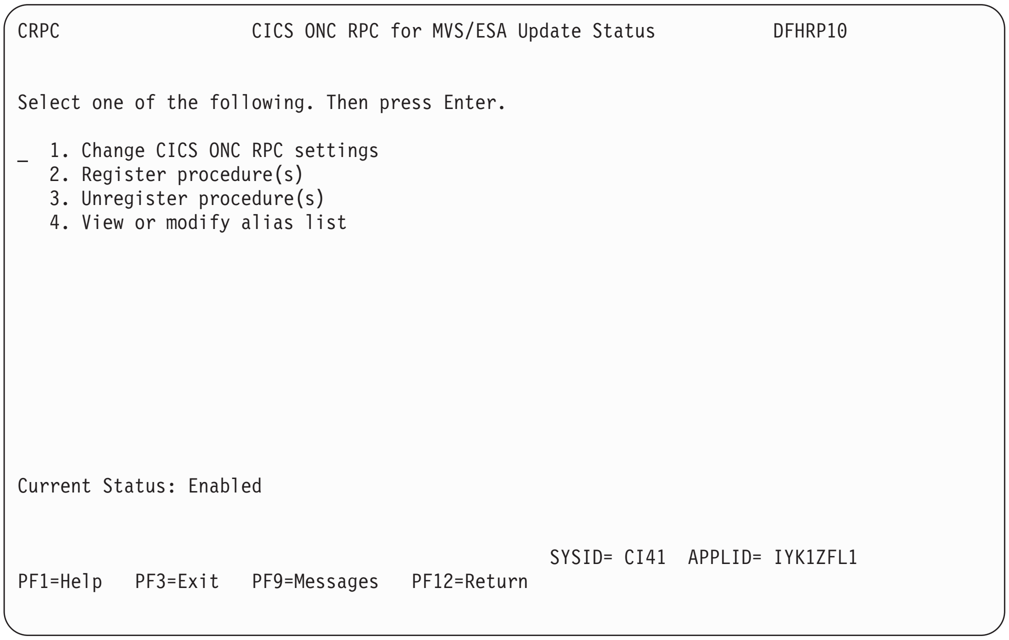


```
CRPC                    CICS ONC RPC for MVS/ESA                       DFHRP04

Select one of the following. Then press Enter.

_  1. Disable CICS ONC RPC
   2. View or modify the CICS ONC RPC data set
   3. View or modify CICS ONC RPC status














Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages
```

Figure 53. Panel DFHRP04

Updating CICS ONC RPC status

If you select option 3 on panel DFHRP04, panel DFHRP10 is shown.



```
CRPC                 CICS ONC RPC for MVS/ESA Update Status           DFHRP10


Select one of the following. Then press Enter.

_  1. Change CICS ONC RPC settings
   2. Register procedure(s)
   3. Unregister procedure(s)
   4. View or modify alias list











Current Status: Enabled


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

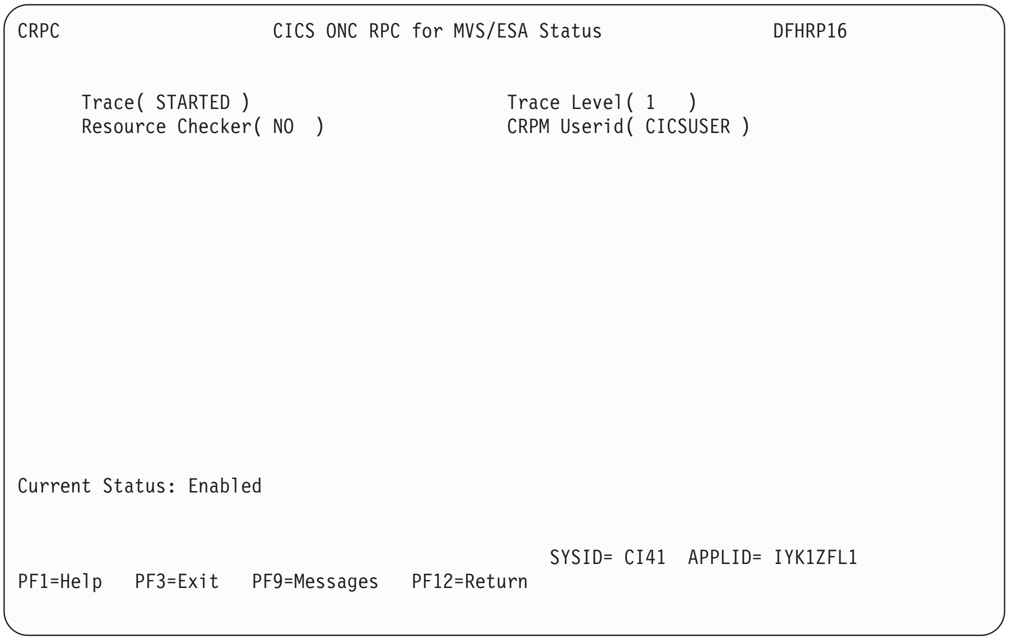
Figure 54. Panel DFHRP10

Select an option, then press Enter.

Option	For more information see:
1	“Changing the CICS ONC RPC status”
2	“Defining, saving, modifying, and deleting 4-tuples” on page 261
3	“Unregistering 4-tuples” on page 268
4	“Processing the alias list” on page 277

Changing the CICS ONC RPC status

If you select option 1 on panel DFHRP10, panel DFHRP16 is shown. (See Figure 55 on page 259.) You can type over any of the entries except CRPM Userid to change the values currently used by CICS ONC RPC. CRPM Userid is displayed only for information. CRPM Userid cannot be changed without first disabling CICS ONC RPC.



```
CRPC                    CICS ONC RPC for MVS/ESA Status                DFHRP16


      Trace( STARTED )                        Trace Level( 1   )
      Resource Checker( NO  )                 CRPM Userid( CICSUSER )
















Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

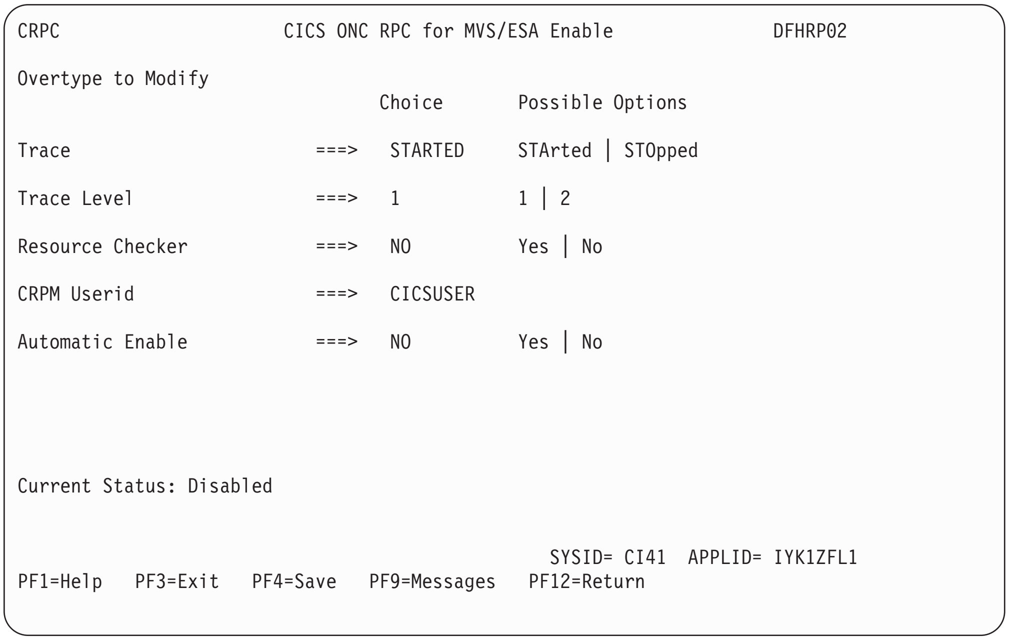
Figure 55. Panel DFHRP16

Enabling CICS ONC RPC

Before enabling CICS ONC RPC, you must accept or modify some options as described in the next section.

Setting and modifying options

If you start the connection manager when CICS ONC RPC is disabled, and select option 1 on panel DFHRP01, panel DFHRP02 is shown. (See Figure 56 on page 260.)



```
CRPC                     CICS ONC RPC for MVS/ESA Enable              DFHRP02

Overtype to Modify
                                  Choice       Possible Options

Trace                       ===>   STARTED     STArted | STOpped

Trace Level                 ===>   1           1 | 2

Resource Checker            ===>   NO          Yes | No

CRPM Userid                 ===>   CICSUSER

Automatic Enable            ===>   NO          Yes | No




Current Status: Disabled


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF4=Save   PF9=Messages   PF12=Return
```

Figure 56. Panel DFHRP02

The values displayed in the Choice column are those stored in the CICS ONC RPC data set. The data set is initialized with the values shown in Figure 56, except that the value displayed for CRPM Userid is the default CICS user ID for the CICS system in which CICS ONC RPC is operating.

You can make entries in the fields listed below. Entries may be in lowercase or uppercase. Where entries to a field are restricted (for example, YES or NO) you can enter the whole option (YES) or the minimum (Y). In the panels, the minimum entry is shown in uppercase in the Possible Options column. In the reference material in this manual, the minimum entry is given in parentheses after the full entry.

Trace Specifies whether CICS ONC RPC tracing is active. STARTED (STA) means it is active, STOPPED (STO) means it is not. The default value is STARTED.

CICS ONC RPC exception trace entries are always written to CICS internal trace whatever the setting of this option. To get non-exception trace entries written, CICS trace must be started, and this option must be set to STARTED.

Trace Level
Specifies the trace level for CICS ONC RPC. The value 1 means that level 1 trace points are traced, and 2 means that both level 1 and 2 are traced. The default value is 1.

Resource Checker
YES (Y) means that CICS ONC RPC is to call the user-written resource-checking module on receipt of every incoming RPC request. NO (N) means the resource checker is not to be called. The default is NO.

CRPM Userid
Specifies the CICS user ID under which the server controller is to run. The default is the default user ID for the CICS system in which CICS ONC RPC is operating.

Automatic Enable

Enter YES (Y) or NO (N). If YES is stored in the CICS ONC RPC data set, you can enable CICS ONC RPC by just typing CRPC; all values are defaulted from the CICS ONC RPC data set, CICS ONC RPC becomes enabled without further user input, and all the 4-tuples with YES for their Register from Data Set option are registered. The default value is NO.

Setting this field has an effect only when you enable CICS ONC RPC. If you use PF4 to save the values to the CICS ONC RPC data set, this value will be effective the next time you enable, unless you override it. A YES in this field in the CICS ONC RPC data set may be overridden by the fast path command CRPC E A(N).

Validating, saving, and activating options

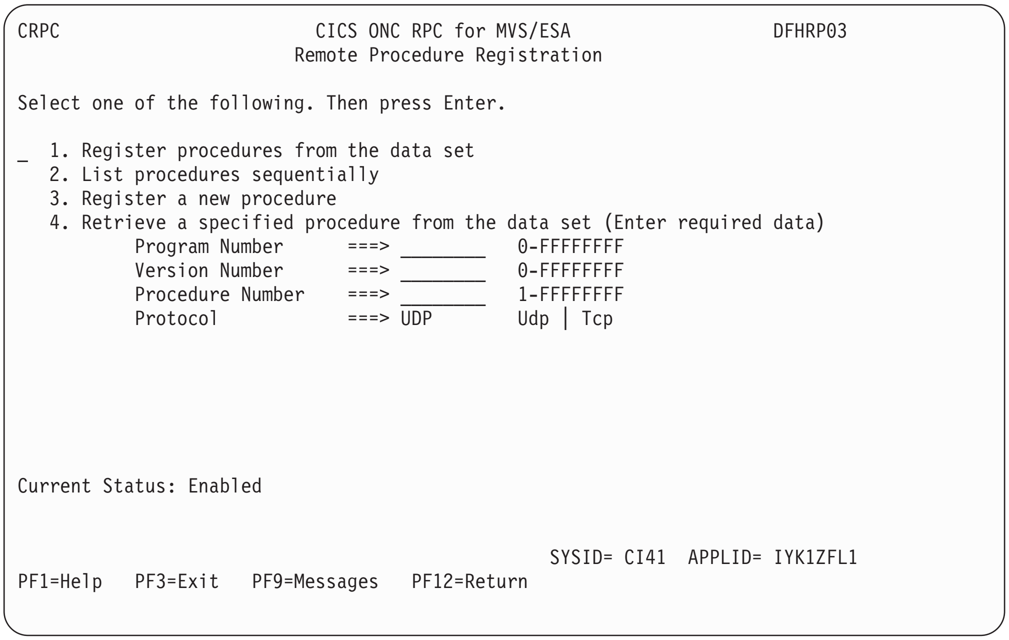
After you have made your changes on panel DFHRP02, press Enter to get them validated by the connection manager.

If you wish to save the new values in the CICS ONC RPC data set, press PF4.

If you press Enter a second time, CICS ONC RPC becomes enabled, and panel DFHRP03 is shown, as described in “Defining, saving, modifying, and deleting 4-tuples”.

Defining, saving, modifying, and deleting 4-tuples

The first panel for defining, saving, modifying, and deleting 4-tuples is DFHRP03. (See Figure 57.) This panel is shown as soon as you have enabled CICS ONC RPC, or if you choose option 2 on panel DFHRP10.



```
CRPC                        CICS ONC RPC for MVS/ESA                   DFHRP03
                          Remote Procedure Registration

Select one of the following. Then press Enter.

_  1. Register procedures from the data set
   2. List procedures sequentially
   3. Register a new procedure
   4. Retrieve a specified procedure from the data set (Enter required data)
           Program Number      ===> ________   0-FFFFFFFF
           Version Number      ===> ________   0-FFFFFFFF
           Procedure Number    ===> ________   1-FFFFFFFF
           Protocol            ===> UDP        Udp | Tcp







Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 57. Panel DFHRP03

If you wish to select option 4, you must first supply the following information:

Program Number
The program number of the 4-tuple whose definition is to be retrieved.

Version Number
The version number of the 4-tuple whose definition is to be retrieved.

Procedure Number
The procedure number of the 4-tuple whose definition is to be retrieved.

Protocol
The protocol of the 4-tuple whose definition is to be retrieved.

Select an option, then press Enter.

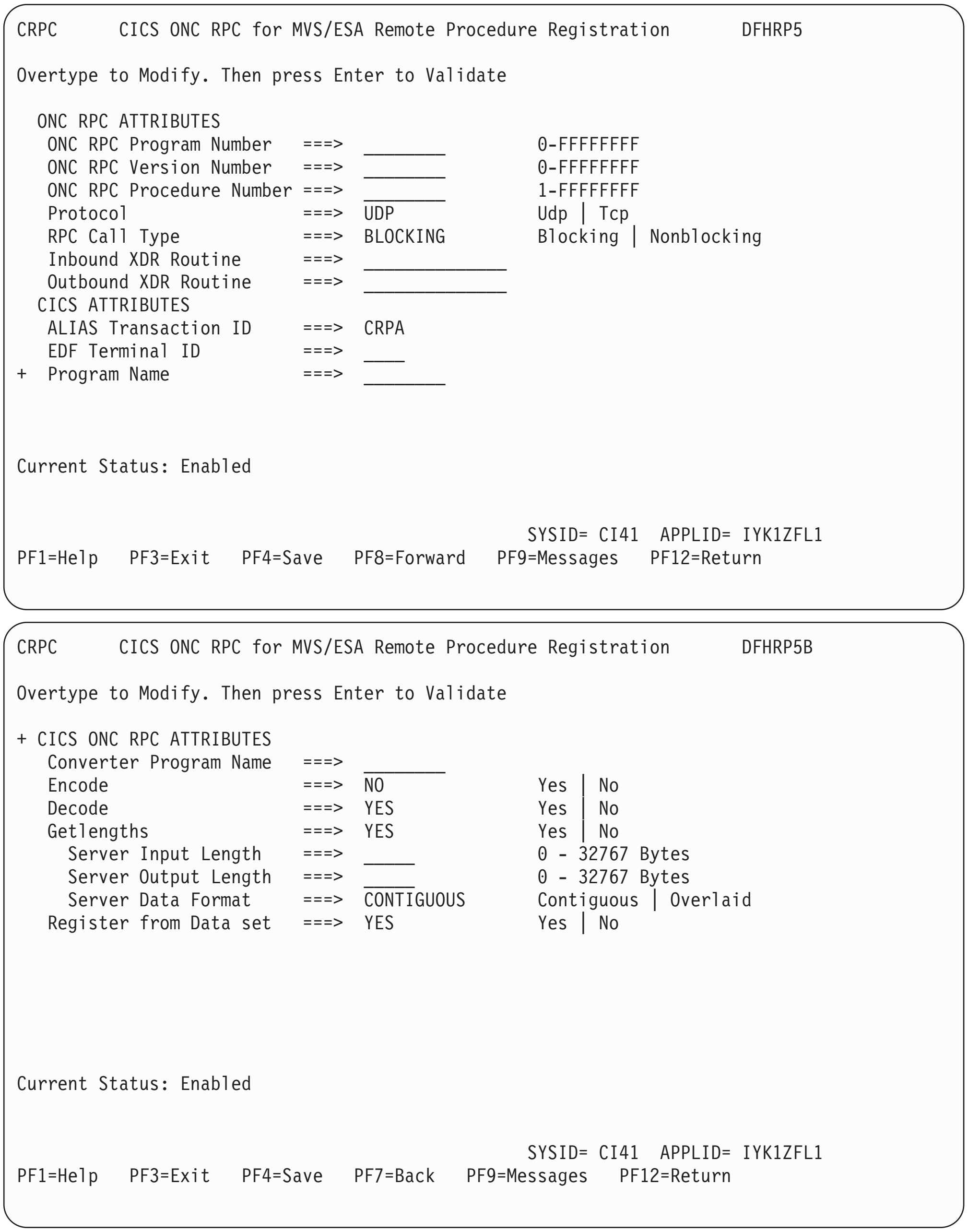
Option	**For more information see:**
1	See below.
2	"Defining the attributes of a 4-tuple"
3	"Unregistering 4-tuples" on page 268
4	See below.

If you select option 1, the 4-tuples in the CICS ONC RPC data set that have YES for their Register from Data Set attribute are all registered.

If you specify a 4-tuple for which there is no definition in the CICS ONC RPC data set, a message is issued when you press Enter, and panel DFHRP03 remains on the screen.

Defining the attributes of a 4-tuple

When you select option 3 or option 4 on panel DFHRP03, panel DFHRP5 is shown. (See Figure 58 on page 263.) If you chose option 3, some of the fields are empty, but if you chose option 4, the details of the selected 4-tuple are shown. You have to supply more information on panel DFHRP5B.



```
CRPC      CICS ONC RPC for MVS/ESA Remote Procedure Registration       DFHRP5

Overtype to Modify. Then press Enter to Validate

  ONC RPC ATTRIBUTES
   ONC RPC Program Number   ===>  ________         0-FFFFFFFF
   ONC RPC Version Number   ===>  ________         0-FFFFFFFF
   ONC RPC Procedure Number ===>  ________         1-FFFFFFFF
   Protocol                 ===>  UDP              Udp | Tcp
   RPC Call Type            ===>  BLOCKING         Blocking | Nonblocking
   Inbound XDR Routine      ===>  ______________
   Outbound XDR Routine     ===>  ______________
  CICS ATTRIBUTES
   ALIAS Transaction ID     ===>  CRPA
   EDF Terminal ID          ===>  ____
+  Program Name             ===>  ________



Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF4=Save   PF8=Forward   PF9=Messages   PF12=Return
```

```
CRPC      CICS ONC RPC for MVS/ESA Remote Procedure Registration       DFHRP5B

Overtype to Modify. Then press Enter to Validate

+ CICS ONC RPC ATTRIBUTES
   Converter Program Name   ===>  ________
   Encode                   ===>  NO               Yes | No
   Decode                   ===>  YES              Yes | No
   Getlengths               ===>  YES              Yes | No
     Server Input Length    ===>  _____            0 - 32767 Bytes
     Server Output Length   ===>  _____            0 - 32767 Bytes
     Server Data Format     ===>  CONTIGUOUS       Contiguous | Overlaid
   Register from Data set   ===>  YES              Yes | No







Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF4=Save   PF7=Back   PF9=Messages   PF12=Return
```

Figure 58. Panels DFHRP5 and DFHRP5B

After you have made your modifications to panel DFHRP5, you should press PF8 to move to panel DFHRP5B. From panel DFHRP5B you can press PF7 if you wish to go back to panel DFHRP5. After you have made your modifications to the panels, you press Enter to get all the modifications validated.

The attributes of a 4-tuple are divided into three categories:

- ONC RPC attributes
- CICS attributes
- CICS ONC RPC attributes

ONC RPC attributes

The first four options establish the 4-tuple whose attributes are being defined.

ONC RPC Program Number
Specifies the program number of the 4-tuple as a hexadecimal string of 1 through 8 characters. You are advised not to use numbers in the range 0 through 1FFFFFFF, as these numbers are reserved for public network services and are allocated by Sun Microsystems.

ONC RPC Version Number
Specifies the version number of the 4-tuple as a hexadecimal string of 1 through 8 characters.

ONC RPC Procedure Number
Specifies the procedure number of the 4-tuple as a hexadecimal string of 1 through 8 characters. Procedure 0 is reserved by TCP/IP for MVS for a procedure with no parameters and no processing that returns an empty reply.

Protocol
Specifies the protocol of the 4-tuple. UDP (U) for UDP, or TCP (T) for TCP.

The remaining options specify the attributes of the 4-tuple.

RPC Call Type
Specifies whether CICS ONC RPC is to treat calls from clients as BLOCKING (B) or NONBLOCKING (N). If NONBLOCKING is specified, the outbound XDR routine cannot be specified, and no reply is sent to the client. The default is BLOCKING.

Inbound XDR Routine
Specifies the name of the inbound XDR routine. If an XDR library function is used, its full name is specified. See Table 20 on page 282 to find out which library routines can be specified here. If a user-defined routine is used, its name (maximum 8 characters) is specified.

Outbound XDR Routine
Specifies the name of the outbound XDR routine, if RPC Call Type is BLOCKING. If an XDR library function is used, its full name is specified. See Table 20 on page 282 to find out which library routines can be specified here. If a user-defined routine is used, its name (maximum 8 characters) is specified. A blank input is valid only if RPC Call Type is NONBLOCKING.

CICS attributes

ALIAS Transaction ID
Specifies the transaction ID to be used for the alias. If this is omitted, and not provided by the **Decode** function, the alias transaction ID is CRPA. For reasons why you might want a different name from CRPA, see “Transaction definitions for extra alias transactions” on page 249.

EDF Terminal ID
Specifies the terminal ID to be used for the alias. You need a terminal ID only if you want to use execution diagnostic facility (EDF) to debug the resource checker, CICS program, or **Encode** function of the converter. A blank means that you cannot use EDF. EDF setup is described in “Using EDF” on page 316.

Program Name
Specifies the name of the CICS program that is to be called to service a request for this 4-tuple.

CICS ONC RPC attributes

Converter Program Name
Specifies the name of the converter program. This name must be specified.

Encode
YES (Y) means that CICS ONC RPC must call the **Encode** function of the converter when servicing a client request for this 4-tuple; NO (N) means that it must not. The default is NO.

Decode
YES (Y) means that CICS ONC RPC must call the **Decode** function of the converter when servicing a client request for this 4-tuple; NO (N) means that it must not. The default is YES.

Getlengths
YES (Y) means that the connection manager must call the **Getlengths** function of the converter before registering this 4-tuple. NO (N) means that it must not. If you specify YES here, you should ignore the next two attributes, but you can set Server Data Format. If you specify NO here, you must specify the next three attributes. The default is YES.

Server Input Length
For the use of this option, see the description of Server Data Format.

If you specified YES for the Getlengths option, leave this field blank.

Server Output Length
For the use of this option, see the description of Server Data Format.

If you specified YES for the Getlengths option, leave this field blank.

Server Data Format
A value that controls:

- How the input data pointer for **Encode** will be set up
- How the communication area length to be checked by the connection manager is calculated

The values you can specify are as follows:

CONTIGUOUS
The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area plus the value of Server Input Length, though **Decode** can modify this offset.

The connection manager calculates a communication area length by adding the values of Server Input Length and Server Output Length. If this length exceeds 32 767 bytes, message DFHRP1965 is issued. If this length is different from the actual length of the communication area passed from **Decode** to the CICS program, errors might occur in the processing of client requests.

OVERLAID
The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area.

The connection manager calculates a communication area length by taking the larger of the output values of Server Input Length and Server Output Length. If this length is different from the actual length of the communication area passed to the CICS program, errors might occur in the processing of client requests.

If you specified YES for the Getlengths option, the value in this field is used as an input to the **Getlengths** function of the converter.

Register from Data Set

YES (Y) means that the 4-tuple is to be registered:

- During automatic enable processing
- When option 1 is selected on panel DFHRP03, as described in "Registering the 4-tuples" on page 267

NO (N) means that it is not. The default is YES. Entries specified as NO can be stored in the CICS ONC RPC data set and you can register them at any time when CICS ONC RPC is enabled.

Saving new 4-tuple definitions

There are five ways of doing this:

- On panel DFHRP03, select option 3. Complete panels DFHRP5 and DFHRP5B, and validate your input as described in "Defining the attributes of a 4-tuple" on page 262. Press PF4 to save the definition in the CICS ONC RPC data set.
- On panel DFHRP03, select option 4. Modify the panels DFHRP5 and DFHRP5B, and validate your input as described in "Defining the attributes of a 4-tuple" on page 262. Press PF4 to save the definition in the CICS ONC RPC data set.
- On panel DFHRP20, select option 3. Complete panels DFHRP21 and DFHRP2B, and validate your input as described in "Changing the attributes of a 4-tuple" on page 275. Press Enter to save the definition in the CICS ONC RPC data set.
- On panel DFHRP20, select option 4. Modify the panels DFHRP21 and DFHRP2B, and validate your input as described in "Changing the attributes of a 4-tuple" on page 275. Press Enter to save the definition in the CICS ONC RPC data set.
- On panel DFHRP03, select option 2. Then on panel DFHRP14, enter command **M** against a 4-tuple. Modify the panels DFHRP21 and DFHRP2B, and validate your input as described in "Changing the attributes of a 4-tuple" on page 275. Press Enter to save the definition in the CICS ONC RPC data set.

Modifying existing 4-tuple definitions

To change some of the attributes of a 4-tuple that already has a definition in the CICS ONC RPC data set, select option 4 on panel DFHRP03 or panel DFHRP20. Change the attributes and validate your input as described in "Defining the attributes of a 4-tuple" on page 262, and press PF4, or Enter, to save the definition in the data set.

Deleting existing 4-tuple definitions

You can delete existing 4-tuple definitions from the CICS ONC RPC data set in either of the following ways:

- On panel DFHRP03, select option 2. Then on panel DFHRP14 you can enter **D** against 4-tuples in the list, and they are deleted from the data set when you press Enter.
- On panel DFHRP21, by using key PF2, as described in “Changing the attributes of a 4-tuple” on page 275.

Registering the 4-tuples

You can register 4-tuples in any of the following ways:

- You can register all the 4-tuples in the CICS ONC RPC data set that are defined with YES specified for Register from Data Set. To do this, select option 1 on panel DFHRP03, and press Enter. After these 4-tuples have been registered, panel DFHRP03 is still displayed, so you can make other selections.
- You can register 4-tuple definitions one at a time. To do this, you use option 3 or option 4 on panel DFHRP03. Make changes, if you need any, to panels DFHRP5 and DFHRP5B and get them validated as described in “Defining the attributes of a 4-tuple” on page 262. To register the definition, press Enter.
- You can register 4-tuples from a list. See “Working with a list of 4-tuples” on page 274.
- When CICS ONC RPC is disabled, you can register all the 4-tuples in the CICS ONC RPC data set that have YES for their Register from Data Set attribute by initiating automatic enable processing.

When a 4-tuple is registered, two things happen:

- If the program-version-protocol 3-tuple has not yet been registered with TCP/IP for MVS, it is registered. The Portmapper assigns a port number to this combination, and that port number is the one that clients use to request the service represented by this 4-tuple. Procedure 0 for the program, version, and protocol becomes available to callers.
- The resources associated with the 4-tuple become available to service client requests. When a client request arrives in CICS ONC RPC, the resources used to service it are those of the 4-tuple whose program, version, and procedure numbers match those of the request, and whose protocol matches the protocol used to transmit the request from the client to the server.

Limits on registration

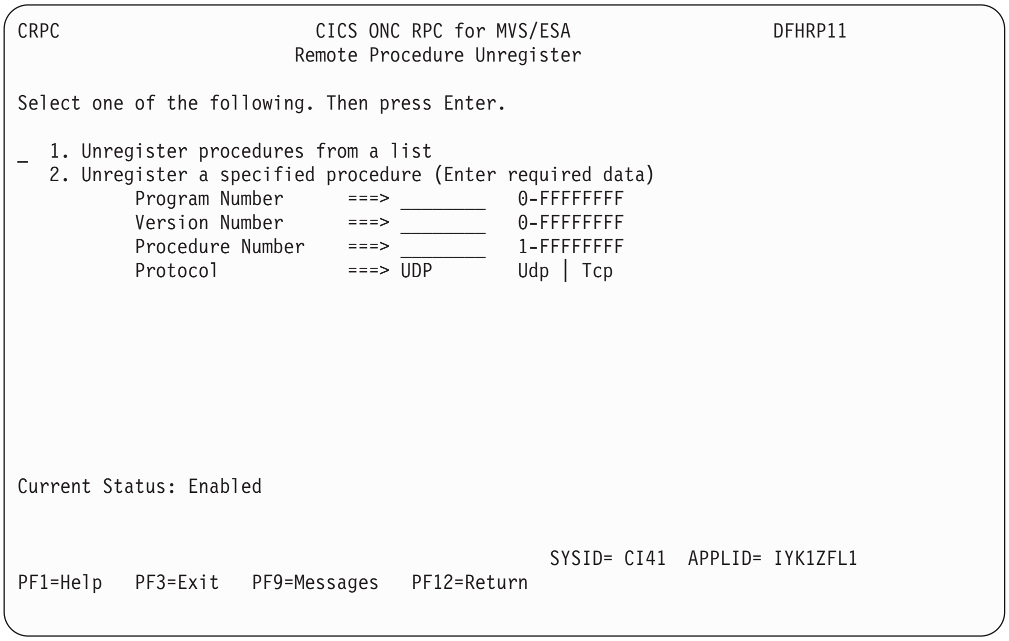
CICS ONC RPC makes 252 sockets available for use as follows:

- One socket is used by each program/version/protocol 3-tuple from the time the first 4-tuple for that program, version and protocol is registered. This socket remains in use until the last 4-tuple with that program and version is unregistered.
- One socket is used by each TCP call for the duration of the call.

If you register too many 4-tuples, you reduce the service that CICS ONC RPC can give to incoming client requests. If you attempt to register more than 252 program-version-protocol 3-tuples with TCP/IP for MVS, the results are unpredictable.

Unregistering 4-tuples

You can unregister 4-tuples that have previously been registered with CICS ONC RPC only when CICS ONC RPC is already enabled. From panel DFHRP10, if you select option 3, panel DFHRP11 is shown. (See Figure 59.)



```
CRPC                        CICS ONC RPC for MVS/ESA                  DFHRP11
                          Remote Procedure Unregister

Select one of the following. Then press Enter.

_  1. Unregister procedures from a list
   2. Unregister a specified procedure (Enter required data)
           Program Number      ===> ________   0-FFFFFFFF
           Version Number      ===> ________   0-FFFFFFFF
           Procedure Number    ===> ________   1-FFFFFFFF
           Protocol            ===> UDP        Udp | Tcp








Current Status: Enabled


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 59. Panel DFHRP11

Select an option, then press Enter.

Option For more information see:

1 “Unregistering 4-tuples from a list” on page 269

2 “Unregistering 4-tuples one by one”

Unregistering 4-tuples one by one

Before you select option 2 on panel DFHRP11, you must supply the following information:

Program Number
The program number of the 4-tuple to be unregistered.

Version Number
The version number of the 4-tuple to be unregistered.

Procedure Number
The procedure number of the 4-tuple to be unregistered.

Protocol
The protocol of the 4-tuple to be unregistered.

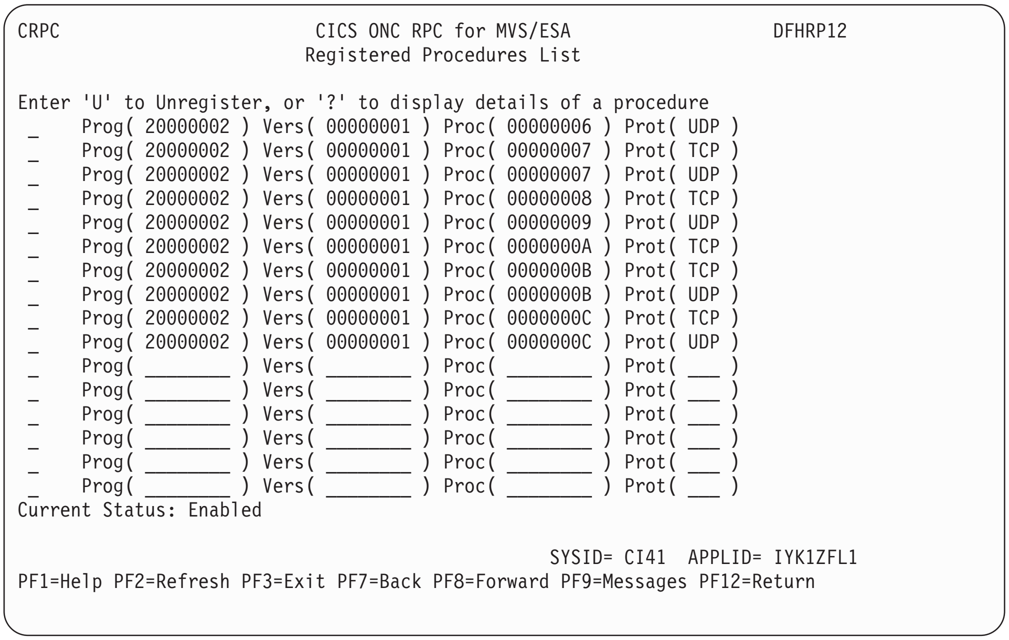
If you specify a 4-tuple that is registered, it is unregistered when you press Enter, and panel DFHRP11 remains on the screen.

If you specify a 4-tuple that is not registered, a message is issued when you press Enter, and panel DFHRP11 remains on the screen.

Unregistering 4-tuples from a list

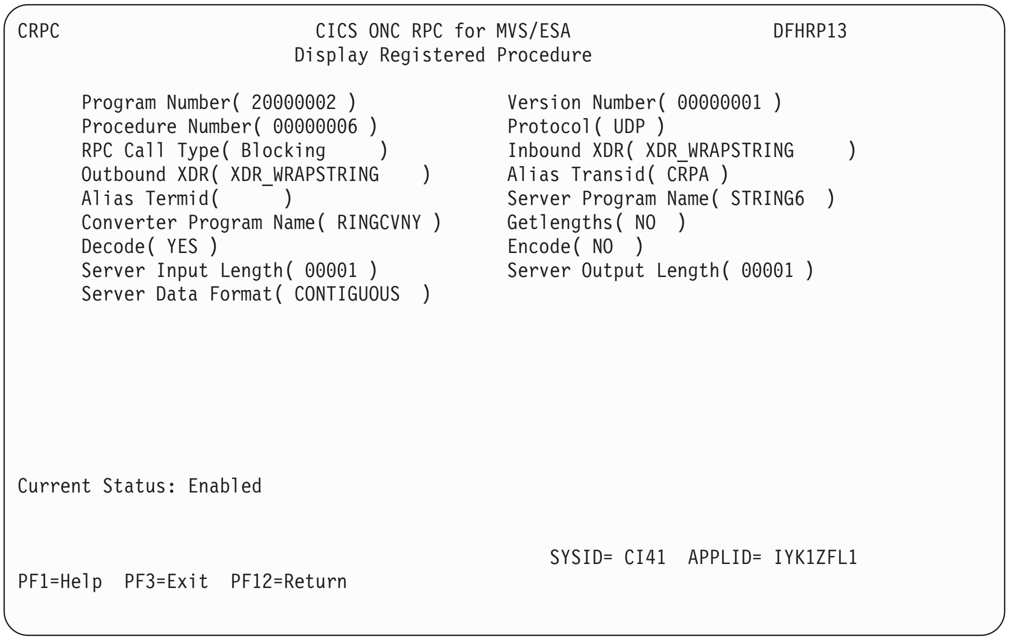
If you select option 1 on panel DFHRP11, the panel DFHRP12 is shown. (See Figure 60.)

This panel presents a list of 4-tuples currently registered with CICS ONC RPC. If you enter **U** against 4-tuples in the list, they are unregistered when you press Enter. You can display the attributes of a 4-tuple by entering **?** against it, and pressing Enter. Panel DFHRP13 is shown. (See Figure 61 on page 270.)



```
CRPC                         CICS ONC RPC for MVS/ESA                  DFHRP12
                            Registered Procedures List

Enter 'U' to Unregister, or '?' to display details of a procedure
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000006 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000007 ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000007 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000008 ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000009 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000A ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000B ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000B ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000C ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000C ) Prot( UDP )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
Current Status: Enabled

                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help PF2=Refresh PF3=Exit PF7=Back PF8=Forward PF9=Messages PF12=Return
```

Figure 60. Panel DFHRP12

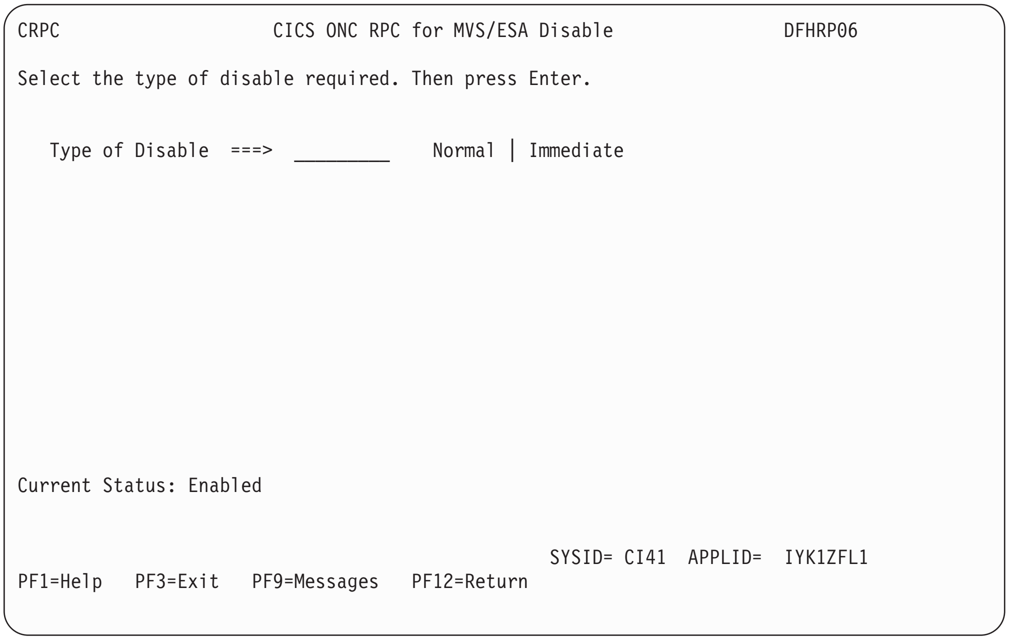


```
CRPC                        CICS ONC RPC for MVS/ESA                  DFHRP13
                          Display Registered Procedure

      Program Number( 20000002 )              Version Number( 00000001 )
      Procedure Number( 00000006 )            Protocol( UDP )
      RPC Call Type( Blocking     )           Inbound XDR( XDR_WRAPSTRING     )
      Outbound XDR( XDR_WRAPSTRING    )       Alias Transid( CRPA )
      Alias Termid(      )                    Server Program Name( STRING6  )
      Converter Program Name( RINGCVNY )      Getlengths( NO  )
      Decode( YES )                           Encode( NO  )
      Server Input Length( 00001 )            Server Output Length( 00001 )
      Server Data Format( CONTIGUOUS  )








Current Status: Enabled



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help  PF3=Exit  PF12=Return
```

Figure 61. Panel DFHRP13

Disabling CICS ONC RPC

From panel DFHRP04, select option 1; panel DFHRP06 is shown. (See Figure 62.)



```
CRPC                  CICS ONC RPC for MVS/ESA Disable                DFHRP06

Select the type of disable required. Then press Enter.


   Type of Disable  ===>  _________    Normal | Immediate













Current Status: Enabled



                                                  SYSID= CI41  APPLID=  IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 62. Panel DFHRP06

In this panel there is only one field to enter.

Type of Disable

NORMAL (N)

Normal disable processing is started.

- All program-version pairs are unregistered from TCP/IP for MVS.
- All work that has already entered CICS ONC RPC is allowed to run to completion, and replies are sent to the relevant client.

IMMEDIATE (I)

Immediate disable processing is started.

- Aliases not yet started do not start at all.
- CICS programs running under aliases are allowed to end, and then the alias abends. If the CICS program ends normally, and was called using DPL, the changes it makes to recoverable resources are committed. If the CICS program is a local program, the changes it makes to recoverable resources are backed out unless the CICS program takes a syncpoint with EXEC CICS SYNCPOINT.
- All the program-version pairs are unregistered from TCP/IP for MVS.
- No replies are sent to clients, so they do not know whether the CICS program has run or not.

Pressing Enter causes the entry you have made to be validated. Pressing Enter a second time begins disable processing. The Current Status is changed to Disabling or Disabled, depending on the progress of disable processing. When disable processing is complete, pressing Enter will change the Current Status to Disabled.

The panel is displayed until you use PF3 or PF12.

On CICS normal shutdown

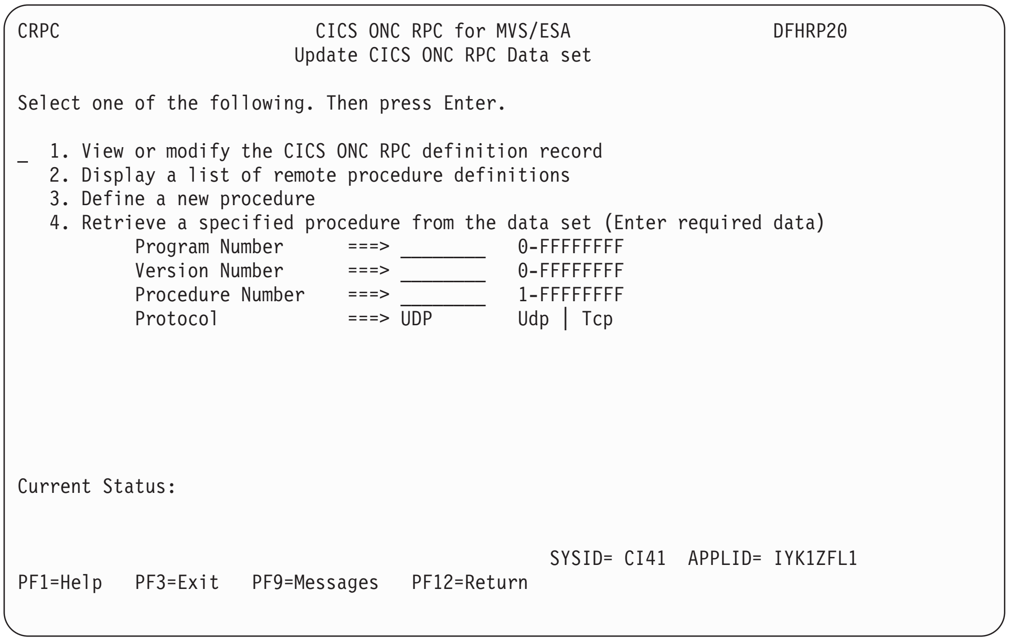
CICS normal shutdown starts normal disable processing for CICS ONC RPC.

On CICS immediate shutdown

On CICS immediate shutdown, all transactions are terminated. Clients are not informed of the shutdown or its effects. The program-version-protocol 3-tuples that are registered with TCP/IP for MVS might remain registered.

Updating the CICS ONC RPC data set

If you select option 2 on panel DFHRP01, or option 2 on panel DFHRP04, panel DFHRP20 is shown. (See Figure 63.)



```
CRPC                        CICS ONC RPC for MVS/ESA                  DFHRP20
                          Update CICS ONC RPC Data set

Select one of the following. Then press Enter.

_  1. View or modify the CICS ONC RPC definition record
   2. Display a list of remote procedure definitions
   3. Define a new procedure
   4. Retrieve a specified procedure from the data set (Enter required data)
           Program Number      ===> ________   0-FFFFFFFF
           Version Number      ===> ________   0-FFFFFFFF
           Procedure Number    ===> ________   1-FFFFFFFF
           Protocol            ===> UDP        Udp | Tcp







Current Status:



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 63. Panel DFHRP20

The Current Status field in this panel might show Enabled or Disabled, depending on which panel you came from.

Before selecting option 4, you must supply the following information:

Program Number
The program number of the 4-tuple whose definition is to be retrieved.

Version Number
The version number of the 4-tuple whose definition is to be retrieved.

Procedure Number
The procedure number of the 4-tuple whose definition is to be retrieved.

Protocol
The protocol of the 4-tuple whose definition is to be retrieved.

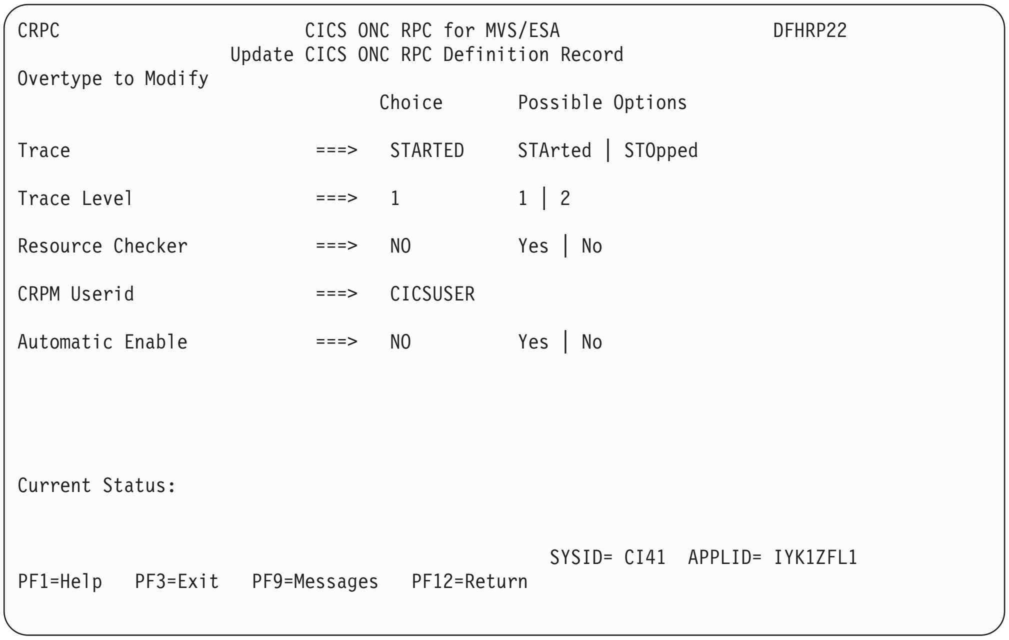
Select an option, then press Enter.

Option	**For more information see:**
1	“Updating the CICS ONC RPC definition record” on page 273
2	“Working with a list of 4-tuples” on page 274
3	“Changing the attributes of a 4-tuple” on page 275
4	“Changing the attributes of a 4-tuple” on page 275

If you specify a 4-tuple which is not defined in the CICS ONC RPC data set, a message is issued when you press Enter, and panel DFHRP20 remains on the screen.

Updating the CICS ONC RPC definition record

If you select option 1 on panel DFHRP20, panel DFHRP22 is shown. (See Figure 64.)



```
CRPC                       CICS ONC RPC for MVS/ESA                   DFHRP22
                    Update CICS ONC RPC Definition Record
Overtype to Modify
                                  Choice       Possible Options

Trace                       ===>   STARTED     STArted | STOpped

Trace Level                 ===>   1           1 | 2

Resource Checker            ===>   NO          Yes | No

CRPM Userid                 ===>   CICSUSER

Automatic Enable            ===>   NO          Yes | No




Current Status:


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF3=Exit   PF9=Messages   PF12=Return
```

Figure 64. Panel DFHRP22

The values displayed in the Choice column are those stored in the CICS ONC RPC data set.

After you have made your changes you should press Enter to get them validated. You can then press Enter again to update the CICS ONC RPC data set with the values you have supplied. The next time you start the connection manager, the saved options are used to set up panel DFHRP02

Trace Specifies whether CICS ONC RPC tracing is active. STARTED (STA) means it is active, STOPPED (STO) means it is not. The default value is STARTED.

CICS ONC RPC exception trace entries are always written to CICS internal trace whatever the setting of this option. To get non-exception trace entries written, CICS trace must be started, and this option must be set to STARTED.

Trace Level
Specifies the trace level for CICS ONC RPC. The value 1 means that level 1 trace points are traced, 2 means that both level 1 and level 2 are traced. The default value is 1.

Resource Checker
YES (Y) means that CICS ONC RPC is to call the user-written

resource-checking module on receipt of every incoming RPC request. NO (N) means the resource checker is not to be called. The default is NO.

CRPM Userid

Specifies the CICS user ID under which the server controller is to operate. The default is the default user ID for the CICS system in which CICS ONC RPC is operating.

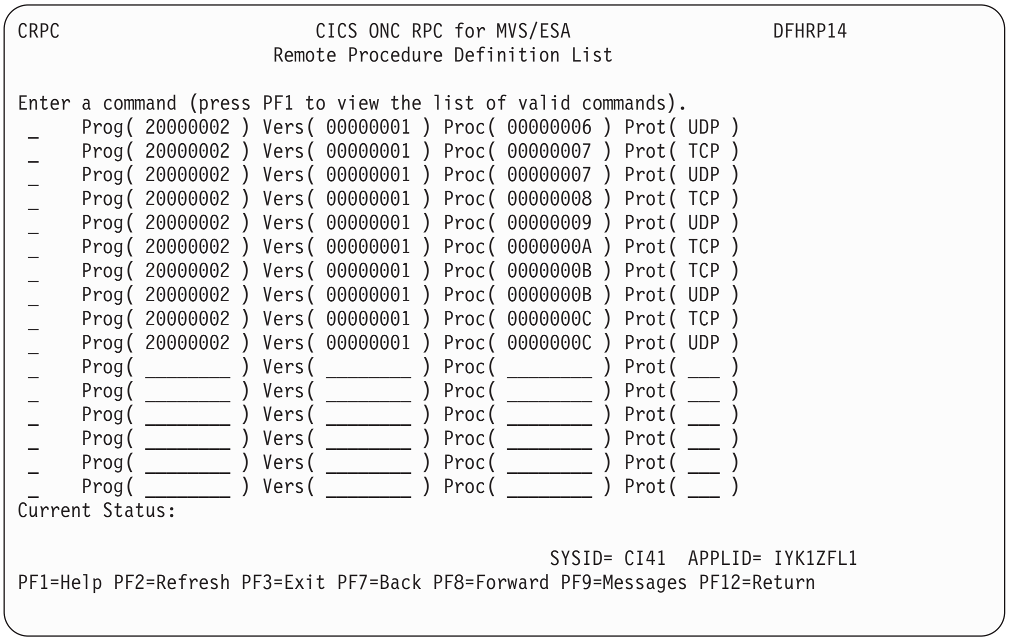
Automatic Enable

Enter YES (Y) or NO (N). If YES is stored in the CICS ONC RPC data set, you can enable CICS ONC RPC by just typing CRPC; all values are defaulted from the CICS ONC RPC data set, CICS ONC RPC becomes enabled without further user input, and all the 4-tuples with YES for their Register from Data Set option are registered. The default value is NO.

Setting this field has an effect only when you enable CICS ONC RPC. If you save the values to the CICS ONC RPC data set, this value will be effective the next time you enable, unless you override it. The value of this field in the CICS ONC RPC data set may be overridden by the fast path command CRPC E A(N).

Working with a list of 4-tuples

If you select option 2 on panel DFHRP03, or option 2 on panel DFHRP20, panel DFHRP14 is shown. (See Figure 65.)



```
CRPC                        CICS ONC RPC for MVS/ESA                  DFHRP14
                        Remote Procedure Definition List

Enter a command (press PF1 to view the list of valid commands).
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000006 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000007 ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000007 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000008 ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 00000009 ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000A ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000B ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000B ) Prot( UDP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000C ) Prot( TCP )
 _    Prog( 20000002 ) Vers( 00000001 ) Proc( 0000000C ) Prot( UDP )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
Current Status:

                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help PF2=Refresh PF3=Exit PF7=Back PF8=Forward PF9=Messages PF12=Return
```

Figure 65. Panel DFHRP14

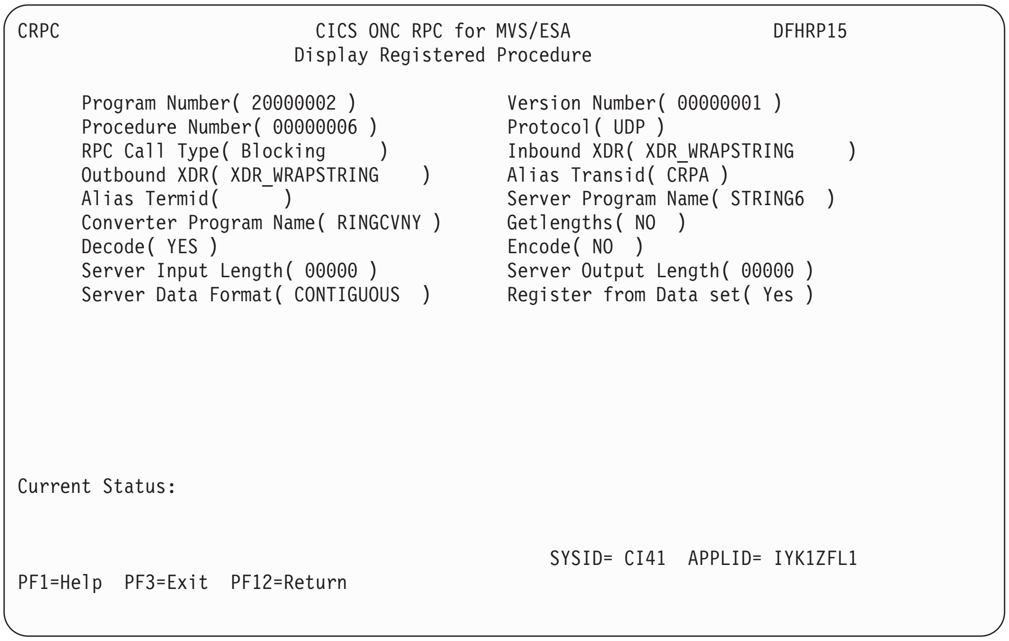
This panel presents a list of 4-tuples currently defined in the CICS ONC RPC data set. If CICS ONC RPC is enabled, the 4-tuples that are currently registered are shown highlighted. You can put a command against a 4-tuple, and it takes effect when you press Enter. The following commands can be entered against a 4-tuple:

D Deletes the definition from the data set.

R If CICS ONC RPC is enabled, registers the 4-tuple with CICS ONC RPC. If CICS ONC RPC is disabled, this command produces an error message.

M Shows panel DFHRP21. See “Changing the attributes of a 4-tuple” for details.

? Shows panel DFHRP15, which displays the attributes of a 4-tuple, but does not allow changes.



```
CRPC                        CICS ONC RPC for MVS/ESA                   DFHRP15
                          Display Registered Procedure

      Program Number( 20000002 )              Version Number( 00000001 )
      Procedure Number( 00000006 )            Protocol( UDP )
      RPC Call Type( Blocking     )           Inbound XDR( XDR_WRAPSTRING     )
      Outbound XDR( XDR_WRAPSTRING    )       Alias Transid( CRPA )
      Alias Termid(      )                    Server Program Name( STRING6  )
      Converter Program Name( RINGCVNY )      Getlengths( NO  )
      Decode( YES )                           Encode( NO  )
      Server Input Length( 00000 )            Server Output Length( 00000 )
      Server Data Format( CONTIGUOUS  )       Register from Data set( Yes )









Current Status:



                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help  PF3=Exit  PF12=Return
```

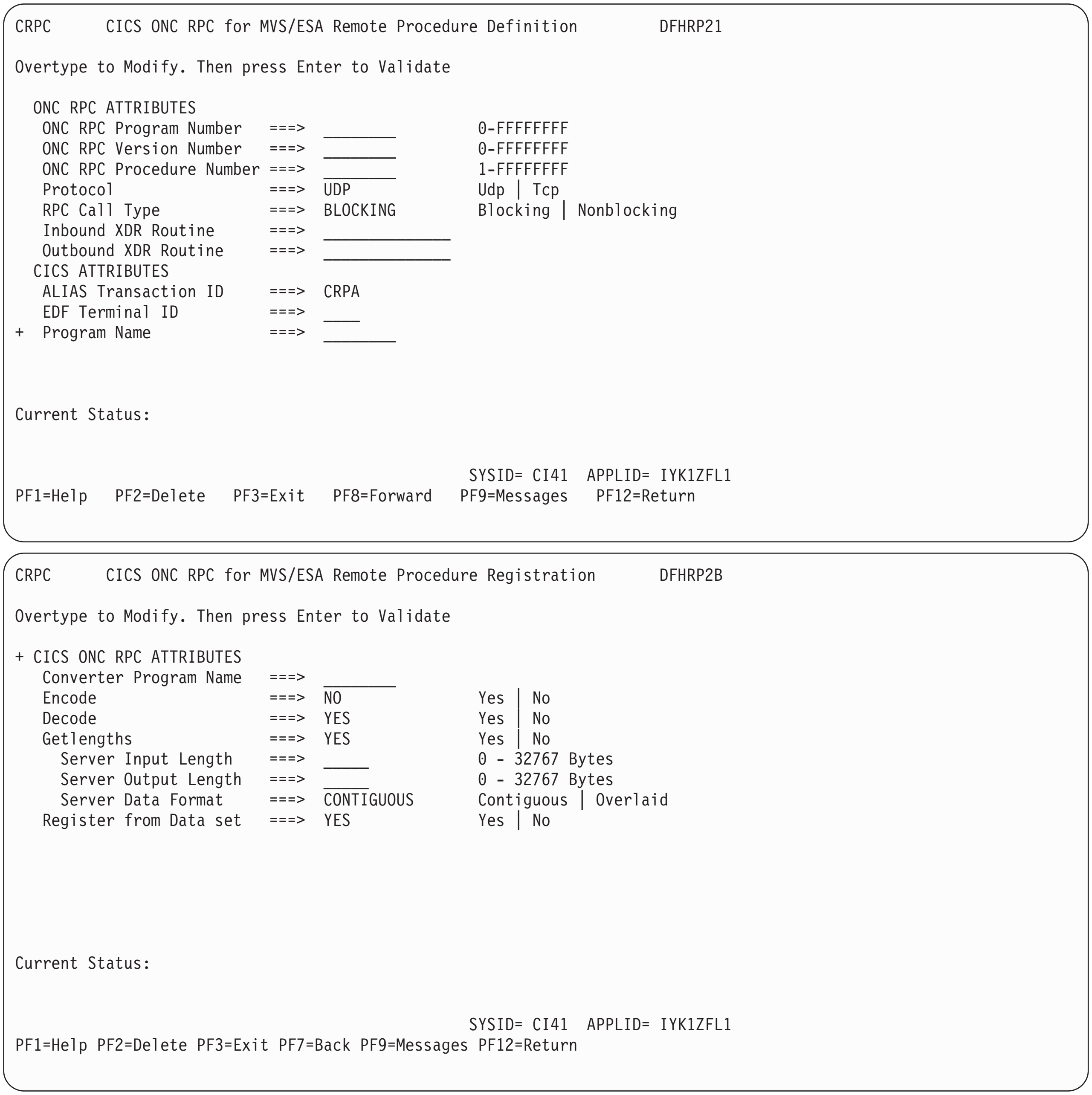
Figure 66. Panel DFHRP15

Changing the attributes of a 4-tuple

If you select option 3 or 4 on panel DFHRP20, or if you enter the **M** command on panel DFHRP14, panel DFHRP21 is shown. (See Figure 67 on page 276.)

The attributes of a 4-tuple are divided into three categories:

- ONC RPC attributes—see “ONC RPC attributes” on page 264.
- CICS attributes—see “CICS attributes” on page 264.
- CICS ONC RPC attributes—see “CICS ONC RPC attributes” on page 265.



```
CRPC      CICS ONC RPC for MVS/ESA Remote Procedure Definition        DFHRP21

Overtype to Modify. Then press Enter to Validate

  ONC RPC ATTRIBUTES
   ONC RPC Program Number   ===>  ________         0-FFFFFFFF
   ONC RPC Version Number   ===>  ________         0-FFFFFFFF
   ONC RPC Procedure Number ===>  ________         1-FFFFFFFF
   Protocol                 ===>  UDP              Udp | Tcp
   RPC Call Type            ===>  BLOCKING         Blocking | Nonblocking
   Inbound XDR Routine      ===>  ______________
   Outbound XDR Routine     ===>  ______________
  CICS ATTRIBUTES
   ALIAS Transaction ID     ===>  CRPA
   EDF Terminal ID          ===>  ____
+  Program Name             ===>  ________



Current Status:


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help   PF2=Delete   PF3=Exit   PF8=Forward   PF9=Messages   PF12=Return
```

```
CRPC      CICS ONC RPC for MVS/ESA Remote Procedure Registration      DFHRP2B

Overtype to Modify. Then press Enter to Validate

+ CICS ONC RPC ATTRIBUTES
   Converter Program Name   ===>  ________
   Encode                   ===>  NO               Yes | No
   Decode                   ===>  YES              Yes | No
   Getlengths               ===>  YES              Yes | No
     Server Input Length    ===>  _____            0 - 32767 Bytes
     Server Output Length   ===>  _____            0 - 32767 Bytes
     Server Data Format     ===>  CONTIGUOUS       Contiguous | Overlaid
   Register from Data set   ===>  YES              Yes | No






Current Status:


                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help PF2=Delete PF3=Exit PF7=Back PF9=Messages PF12=Return
```

Figure 67. Panels DFHRP21 and DFHRP2B

You can use these panels to delete a 4-tuple definition from the CICS ONC RPC data set by pressing PF2.

If you wish to modify the 4-tuple definition, you should first make modifications to panel DFHRP21, and then press PF8 to move to panel DFHRP2B. From panel DFHRP2B you can press PF7 if you wish to go back to panel DFHRP21. After you have made your modifications to the panels, you should press Enter to get all the modifications validated, and then press Enter again to get the definition changed.

Processing the alias list

If you select option 4 on panel DFHRP10, panel DFHRP17 is shown. (See Figure 68.)

This panel gives a list of the aliases that have been started, or scheduled, by the server controller, but have not yet ended. Each alias has two lines on the panel.

- The first line shows the 4-tuple for the client request.
- The second line shows the CICS task number of the alias that is processing the client request.

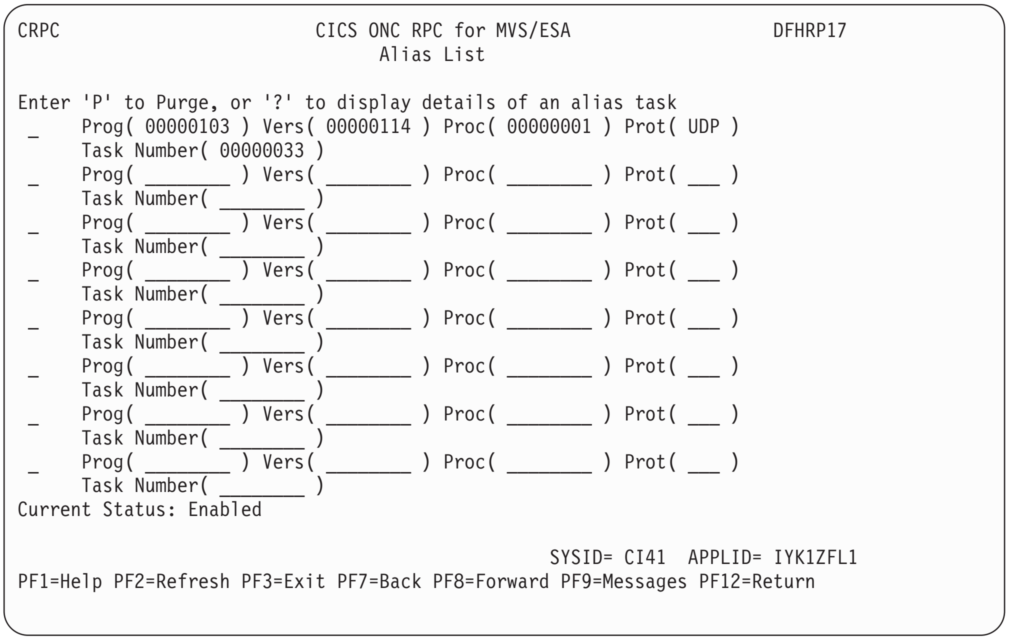
If the alias is scheduled, but not yet started, the task number is blank. If the alias has started, a task number is given and the line is highlighted.

You can enter the following commands against an alias:

P Purges the alias.

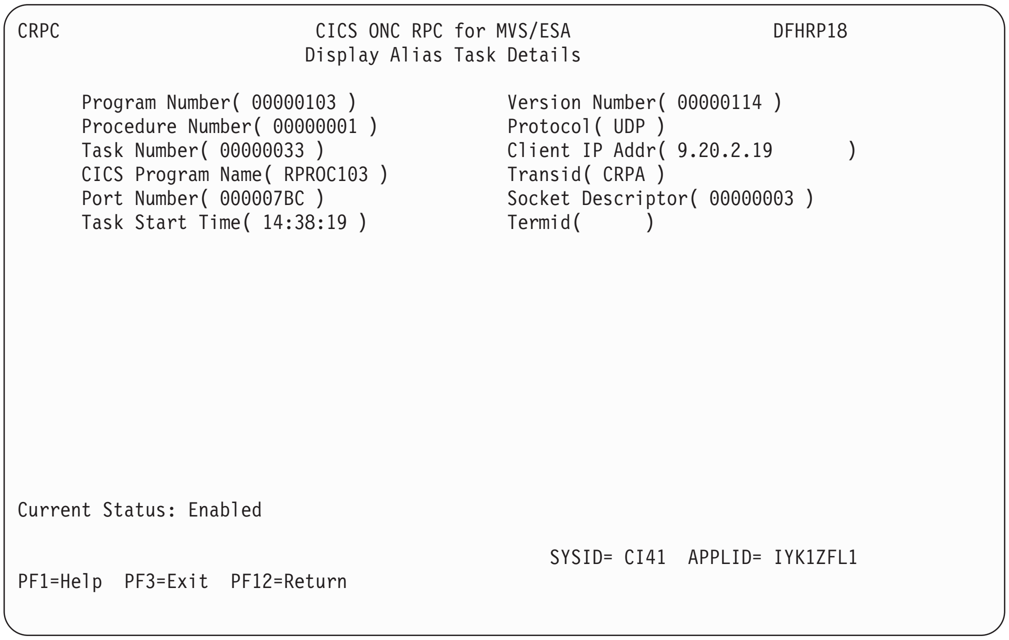
? Shows panel DFHRP18, which displays details of the alias and the associated client request. (See Figure 69 on page 278.)

If the alias is scheduled, but not yet started, the task number and start time are blank. If the alias has started, a task number and start time are given.



```
CRPC                        CICS ONC RPC for MVS/ESA                  DFHRP17
                                  Alias List

Enter 'P' to Purge, or '?' to display details of an alias task
 _    Prog( 00000103 ) Vers( 00000114 ) Proc( 00000001 ) Prot( UDP )
      Task Number( 00000033 )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
 _    Prog( ________ ) Vers( ________ ) Proc( ________ ) Prot( ___ )
      Task Number( ________ )
Current Status: Enabled

                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help PF2=Refresh PF3=Exit PF7=Back PF8=Forward PF9=Messages PF12=Return
```

Figure 68. Panel DFHRP17



```
CRPC                         CICS ONC RPC for MVS/ESA                  DFHRP18
                            Display Alias Task Details

      Program Number( 00000103 )              Version Number( 00000114 )
      Procedure Number( 00000001 )            Protocol( UDP )
      Task Number( 00000033 )                 Client IP Addr( 9.20.2.19       )
      CICS Program Name( RPROC103 )           Transid( CRPA )
      Port Number( 000007BC )                 Socket Descriptor( 00000003 )
      Task Start Time( 14:38:19 )             Termid(      )














Current Status: Enabled

                                                  SYSID= CI41  APPLID= IYK1ZFL1
PF1=Help  PF3=Exit  PF12=Return
```

Figure 69. Panel DFHRP18

Chapter 23. Programming with CICS ONC RPC

This chapter tells you how to write the user-replaceable programs that were described in “User-replaceable programs” on page 239. It describes the general process of development, including details of the interfaces to the converter functions.

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

This chapter is organized as follows:

Developing an ONC RPC application for CICS ONC RPC

ONC RPC applications are always developed as client/server pairs. The process described in this section takes account of this, but concentrates on the server, because CICS ONC RPC affects this and not the client. For details of the client development process, read the documentation of the ONC RPC system running on the client machine.

The process of developing all the material needed for an ONC RPC application using CICS ONC RPC is summarized in Figure 70 on page 280, which should be compared with Figure 44 on page 233, which showed the process for ONC RPC without CICS ONC RPC.

© Copyright IBM Corp. 1994, 1998

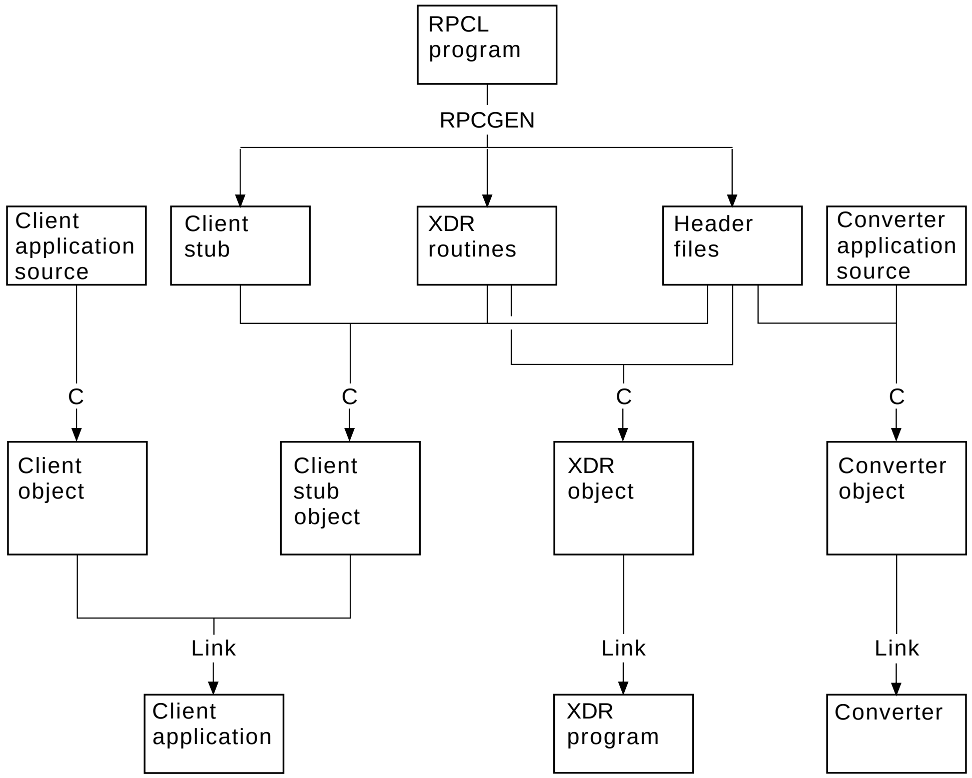


Figure 70. Program development with CICS ONC RPC

The figure shows the development process when RPCGEN is used to create source text from the interface definition in the RPCL program. If you do not use RPCGEN, you must supply some of its output—XDR routines and header files—yourself. The development of the CICS program to service client requests is not shown.

The sequence of development of an ONC RPC application is summarized below. Each step is described in detail in the sections following the summary.

1. Decide what data is to be sent from client to server and what is to be returned. If the data structures the client uses are not simple, you might choose to use RPCGEN to help with managing the data. If you choose to use RPCGEN, some of its output is useful for writing the user-replaceable programs for CICS ONC RPC.
2. Decide the format of the communication area to be used by the CICS program. If the client is to use an existing CICS program, the format is already decided.
3. Write the XDR routines. If the translations you need can be done by an XDR library function supported by the connection manager (see Table 20 on page 282), you do not need to write an XDR routine. If you used RPCGEN, it has generated source for XDR routines. In any other case you must write the XDR routines yourself.

   XDR routines must be written in C.
4. Write the converter. If you used RPCGEN, and you are going to write your converter in C, the header files produced by RPCGEN describe the data structures that **Decode** receives and **Encode** returns. The format of the CICS program communication area is also used by **Decode** and **Encode**.
5. Write the resource checker, if required. You may want to write your own resource checker to validate incoming client requests. “Chapter 24. Security” on page 305

page 305 tells you about this and other security facilities available for use with CICS ONC RPC. “Writing the resource checker” on page 307 gives you details on writing a resource checker.

6. Compile and link the user-replaceable programs. If you used RPCGEN, the header files are needed for the compilation of the XDR routines and the converter if it is in C.
7. Define the server application set to CICS. This means defining programs for the CICS program, any XDR routines that are not just XDR library functions, and the converter. One or more alias transaction definitions may also be required, see “Defining CICS ONC RPC resources to CICS” on page 249.
8. Use the connection manager to define a 4-tuple and save it in the CICS ONC RPC data set. The definition specifies the CICS program, XDR routines, and converter, as described in “Defining the attributes of a 4-tuple” on page 262.

Step 1—Decide what data is to be sent

This step is outside the scope of this manual. What you do depends on the nature of the data to be sent with the request and with the reply. Defining data with RPCL and the use of RPCGEN are described in Sun Microsystems’ publication *Network Programming*.

Step 2—Decide the format of the communication area

This step is also outside the scope of this manual. You are reminded that if the CICS program that services a client request is not in the same CICS region as CICS ONC RPC, the maximum communication area length is 35 000 bytes. If the CICS program resides in a server other than CICS Transaction Server for OS/390, other restrictions might also apply.

Step 3—Write the XDR routines

If you used RPCGEN in Step 1, you use the XDR source programs generated by RPCGEN. If the XDR source uses the **xdr_char** or **xdr_u_char** XDR library functions, you must use the C `#define` directive to make the compiler use the **xdr_text_char** function instead.

If the translations you need can be done by an XDR library function supported by the connection manager (see Table 20 on page 282), you do not need to write an XDR routine. Instead you specify one of the XDR library functions described below when you register a 4-tuple with the connection manager.

If you write your own XDR routine, you need to use the XDR library functions. The full C definitions of these functions are documented in the *TCP/IP for MVS: Programmer’s Reference* .

CICS ONC RPC supports only the functions listed below. You should use only these functions in your own XDR routines. These functions convert C data types to XDR formats, and XDR formats to C data types.

Some of these function names cannot be used in the connection manager when specifying XDR library functions for the inbound and outbound XDR routines for a 4-tuple. In the column headed **CM**, an asterisk means that the XDR library routine can be specified in the connection manager, while a blank means that it cannot.

Table 20. Supported XDR library functions

XDR library function	CM	C type
xdr_int	*	int
xdr_u_int	*	unsigned int
xdr_long	*	long
xdr_u_long	*	unsigned long
xdr_short	*	short int
xdr_u_short	*	unsigned short int
xdr_float	*	float
xdr_bool	*	bool_t (see note)
xdr_double	*	double
xdr_enum		enum
xdr_void	*	void
xdr_array		variable-length array
xdr_opaque		fixed-length uninterrupted data
xdr_bytes		variable-length array of bytes
xdr_pointer		object references, including null pointers
xdr_reference		object references
xdr_char	*	character
xdr_u_char	*	unsigned character
xdr_text_char	*	text character
xdr_string		null-terminated character arrays
xdr_vector		fixed-length array with arbitrary element size
xdr_wrapstring	*	variable-length null-terminated character arrays
xdr_union		discriminated union

Note: **bool_t** is not a built-in C data type; it is defined in an ONC RPC header (as a C **int**).

Names of user-written XDR routines are subject to the same restrictions as CICS programs.

You must take care when writing your own XDR routines. These run in the CICS address space and can overwrite CICS code and other user application storage, because they are defined with EXECKEY(CICS).

Code page conversions

Conversion between ASCII and EBCDIC (or vice versa) is done by XDR library functions supplied as part of TCP/IP for MVS. The relevant XDR routines are **xdr_text_char**, **xdr_string**, and **xdr_wrapstring**. These routines use EBCDIC-to-ASCII and ASCII-to-EBCDIC translate tables, which are loaded at TCP/IP for MVS initialization from a data set containing one of the possible translate tables provided with TCP/IP for MVS.

Thus all ONC RPC requests from all clients use the same translate table. There is no provision for ONC RPC data from different client workstations or from different client end users to have different character sets.

Various single-byte character set (SBCS) translate tables are provided with TCP/IP for MVS, one of which is generated during TCP/IP for MVS customization. If none of these is suitable, you could provide your own, as described in the *TCP/IP for MVS: Customization and Administration Guide* .

TCP/IP for MVS 3.1 provides several code pages for double-byte character sets (DBCS). If you want to include DBCS in ONC RPC data you have to write your own XDR routines to convert the double-byte characters.

Step 4—Write the converter

This section describes how you can write a converter to perform various tasks. Some of these tasks are required for all 4-tuples, others only for some.

The section describes in turn each of the tasks, indicating the converter function (**Getlengths**, **Decode**, or **Encode**) used.

The parameter details and responses of each of the converter functions are given at the end of the section in “Getlengths” on page 293, “Decode” on page 296, and “Encode” on page 301.

Tasks that can be performed by a converter

The tasks to be performed are:

- Telling the connection manager or the server controller the lengths of the input and output data for the CICS program
- Telling the connection manager the CICS program data format
- Mapping data between client and CICS program formats, as illustrated in Figure 49 on page 243 and Figure 51 on page 244
- Telling the server controller which alias and CICS program are to be used to service a request, if those specified when the 4-tuple was defined are to be changed

Lengths of the CICS program input and output data

CICS ONC RPC needs to know the length of the CICS program input and output data for each 4-tuple. For each 4-tuple, the lengths may be defined in one of three places:

- In the connection manager if the lengths do not vary from call to call. You specify the lengths in the connection manager and specify NO for the Getlengths attribute of the 4-tuple. In this case **Getlengths** is not called.
- In **Getlengths** if the lengths do not vary from call to call, returning the values in the **glength_server_input_data_len** and **glength_server_output_data_len** output fields. In the connection manager you specify YES for the Getlengths attribute of the 4-tuple, and leave the length fields blank.

In either of the above cases, if **Decode** is specified for the 4-tuple, the **Decode** function can change the lengths.

- In **Decode**, if the lengths of the data structures vary from call to call. You return the lengths on each call by using the **decode_server_input_data_len** and **decode_server_output_data_len** output fields. The lengths specified with the connection manager or **Getlengths** are supplied as inputs to **Decode** in these fields.

Setting the CICS program data format

CICS ONC RPC needs to know the CICS program data format for each 4-tuple. The data format defines how the input and output data is arranged in the CICS program communication area. You can set this either in **Getlengths** or in the connection manager. If you choose **Getlengths**, use the output field **glength_server_data_format**. The value specified with the connection manager is supplied as input to **Getlengths** in this field.

Mapping data between client and CICS program formats

You need to map the incoming data intended for the CICS program only if it is not in the format required by the CICS program. This is typically for:

- Client data structures that contain pointers to other data. These are rebuilt by the inbound XDR routine in the same form as they existed in the client. The data for the CICS program must be copied into a single area of storage to be passed to the CICS program as its communication area.
- CICS programs that are written in a language other than C. The incoming client request always has a C data structure. If your CICS program is written in COBOL, for example, you need to perform a C-to-COBOL mapping in **Decode**.

The mapping is always done by **Decode** for the input data for the CICS program. In most cases, the output data needs to be mapped in the opposite direction by **Encode**.

On input, the client data is pointed to by the **Decode** input field **decode_client_data_ptr**. **Decode** maps this data into the form which the CICS program requires.

To achieve the mapping, **Decode** must allocate an area of CICS storage, using EXEC CICS GETMAIN SHARED. **Decode** must set the output field **decode_returned_data_ptr** to the address returned by the GETMAIN command, and put the input data passed from the client into the storage, making changes where applicable.

Changing the alias and CICS program

You can use **Decode** to redirect a client request to another CICS program. CICS ONC RPC then ignores the original program name that was defined in the connection manager for the requested 4-tuple. To reroute a client request, specify a new CICS program name in the **decode_server_program** field in **Decode.** This facility allows a client to pass a CICS program name in the data it sends in the remote procedure call. The new CICS program must work with the same communication area format, converter, and XDR output routine as the original program.

You can use **Decode** to change the name of the alias transaction to run the CICS program by setting the **decode_alias_transid** output field. CICS ONC RPC then ignores the transaction ID that was defined in the connection manager for the requested 4-tuple. This facility allows a client to pass the alias transaction ID in the data it sends with the remote procedure call.

Changing security information

You may want your CICS ONC RPC system to implement security checking on incoming client requests. Such checking usually involves checks on the client user ID and password. One of the ways the client can provide these is by including them in the data structure it sends.

Decode can retrieve this information from the incoming data, and return it in the output fields. The user ID should be returned in the output field **decode_userid**; the password should be returned as part of the data pointed to by the **decode_returned_data_ptr** field. These outputs can either be passed by the client or generated by **Decode** in whatever way you want. For instance, **Decode** can derive the CICS user ID and password for the client request by using the **decode_client_address** field, or the authentication fields **decode_aup_...** that identify the client.

Organizing the converter

You can write converters for any CICS-supported compiler. If you choose a language other than C or COBOL, you must write your own header files to define the CICS ONC RPC data structures and constants.

A converter is passed a communication area that contains a parameter that specifies which of the three functions **Getlengths**, **Decode**, or **Encode** is required, and parameters for the particular function, as described in the reference material: “Getlengths” on page 293, “Decode” on page 296, and “Encode” on page 301.

The following C header files (in the SDFHC370 target library) and COBOL copybooks (in the SDFHCOB target library) are provided to help with writing the converter:

- DFHRPUCH for C (DFHRPUCO for COBOL)—contains definitions of the constants that are used in the interface between CICS ONC RPC and the converter.
- DFHRPCDH for C (DFHRPCDO for COBOL)—defines the format of the communication area that is presented to the converter. The communication area is in two parts. The format of the first part is independent of the function that the converter is being asked to perform, and it contains:
  - The eyecatcher for the requested function
  - The function code for the requested function
  - A response to be supplied by the converter
  - A reason code to be supplied by the converter

  The format of the rest of the communication area depends on the converter function requested.

You need a header file produced by RPCGEN only if you used RPCL to define the data structures, and you are writing **Decode** or **Encode**. If you are writing your converter in a language other than C, you need to rewrite the header file in your chosen language, since RPCGEN produces its output only in C.

You need definitions of the CICS structures that you use, and the definition of the CICS program communication area.

Writing a converter in C

The following discussion is based on a converter that consists of four main parts:

- A routing part that consults the function code in the communication area, and then calls the appropriate function
- A function for **Getlengths** processing
- A function for **Decode** processing

- A function for **Encode** processing

Figure 71 shows how you can route control to the appropriate function.

```
  EXEC CICS ADDRESS EIB(dfheiptr);      /*Get addressability of EIB*/

  EXEC CICS ADDRESS COMMAREA(converter_parms_ptr);

  switch(converter_parms_ptr->converter_function) {

   case URP_GETLENGTHS:
   {
     converter_getlengths();
     break;
   }
   case URP_DECODE:
   {
     converter_decode();
     break;
   }
   case URP_ENCODE:
   {
     converter_encode();
     break;
   }

   default:
   {
     converter_parms_ptr->converter_response = URP_INVALID;
   }

  }   /* end switch */

  EXEC CICS RETURN;

}  /* end main */
```

Figure 71. Routing control to the functions in C

In this program fragment, `converter_parms_ptr` is a locally declared pointer to the `converter_parms` structure declared in DFHRPCDH. All the other names beginning `converter_` are names from this structure.

The processing is as follows:

1. The `converter_parms_ptr` pointer is set by using EXEC CICS ADDRESS COMMAREA.
2. The **switch** statement is used to select the function to be called. If you are not providing all the functions, you need fewer **case** statements.
3. If the function is not valid, the response URP_INVALID is returned from the converter. This test is always advised, especially if the converter does not provide all three functions.

Figure 72 on page 287 is an example of a **Decode** function.

```
void converter_decode(void)
{
  decode_parms *decode_parms_ptr;

  decode_parms_ptr = (decode_parms *)converter_parms_ptr;

  if (strncmp
  (decode_parms_ptr->decode_eyecatcher,DECODE_EYECATCHER_INIT,8)
  == 0)
  {
    EXEC CICS GETMAIN
      SET(decode_parms_ptr->decode_returned_data_ptr)
      FLENGTH(sizeof(rem_proc_parms_103) + PW_LEN)
      SHARED
      NOSUSPEND
      CICSDATAKEY
      RESP(response)
      RESP2(response2);

    if (response != DFHRESP(NORMAL))
    {
      memcpy(outline,errmsg1,strlen(errmsg1));
      EXEC CICS WRITEQ TD QUEUE(tdq) FROM(outline) LENGTH(30);
      decode_parms_ptr->decode_response = URP_EXCEPTION;
      decode_parms_ptr->decode_reason = NO_STORAGE;
    }
    else
    {
      /* move password and data to decode_password and
      decode_server_input_data */

      decode_parms_ptr->decode_response = URP_OK;
    };
  }
  else
    decode_parms_ptr->decode_response = URP_INVALID;
}
```

Figure 72. Example of a Decode function in C

In this program fragment, names beginning decode_, except decode_parms_ptr, are names from the decode_parms structure defined in DFHRPCDH.

The processing is as follows:

1. The pointer decode_parms_ptr is set from converter_parms_ptr.
2. The eyecatcher is checked to see if it agrees with the function code. If it does:
   a. EXEC CICS GETMAIN is used to get storage for the password and for the communication area to be passed to the CICS program. The value of PW_LEN is set elsewhere in the program to 8 by #define. The output parameter decode_returned_data_ptr is used directly in the GETMAIN. In this case there is no conversion of data to be done, and the communication area size is the same as the size of the client data structure. (rem_proc_parms_103 is a structure that defines the input data after XDR conversion.)
   b. If the response to the EXEC CICS GETMAIN is not NORMAL, an error message is directed to a transient data queue, the converter response is set to URP_EXCEPTION, and the reason code is set to NO_STORAGE, which is locally declared.

   c. If the response to the EXEC CICS GETMAIN is NORMAL, the data and password are transferred to the storage acquired by GETMAIN (not shown), and the converter response is set to URP_OK.
3. If the eyecatcher is not the one for the function being called, the converter response is set to URP_INVALID.

Writing a converter in COBOL

In the working storage section of the data division, you should use the COPY statement to copy the copybook DFHRPUCO, and any other copybooks you need. You should also define any other data items you need in working storage.

You use the COPY statement to include the definition of the communication area in the linkage section of the data division.

Figure 73 on page 289 shows the layout of the data division. Comments, which would be part of a well-documented converter, are omitted.

The following discussion is based on a converter that consists of four main parts:

- A routing part that consults the function code in the communication area, and then calls the appropriate function
- A function for **Getlengths** processing
- A function for **Decode** processing
- A function for **Encode** processing

Figure 74 on page 290 shows how you can route control to the appropriate function.

```
DATA DIVISION.

WORKING-STORAGE SECTION.

    COPY DFHRPUCO.

01  RESP                          PIC S9(8) COMP.
01  RESP2                         PIC S9(8) COMP.
01  REM-PROC-COMMSIZE             PIC S9(8) COMP VALUE +12.
01  CLIENT-OUT-SIZE               PIC S9(8) COMP VALUE +8.

LINKAGE SECTION.

 01 DFHCOMMAREA.
   02 COMM-PARMLIST PIC X(1).

 01 CONVERTER-PARMS REDEFINES DFHCOMMAREA.
   02 CONVERTER-EYECATCHER PIC X(8).
   02 CONVERTER-FUNCTION PIC 9(8) COMP.
   02 CONVERTER-RESPONSE PIC 9(8) COMP.
   02 CONVERTER-REASON PIC 9(8) COMP.
   02 CONVERTER-PARMLIST PIC X(1).

 01 GLENGTH-PARMS REDEFINES DFHCOMMAREA.
   02 GLENGTH-EYECATCHER PIC X(8).
   02 GLENGTH-FUNCTION PIC 9(8) COMP.
   02 GLENGTH-RESPONSE PIC 9(8) COMP.
   02 GLENGTH-REASON PIC 9(8) COMP.
   02 GLENGTH-SERVER-INPUT-DATA-LEN PIC S9(8) COMP.
   02 ...

 01 DECODE-PARMS REDEFINES DFHCOMMAREA.
   02 ...

 01 DECODE-RETURNED-DATA.
   02 DECODE-PASSWORD PIC X(8).
   02 DECODE-SERVER-INPUT-DATA PIC X(1).

 01 ENCODE-PARMS REDEFINES DFHCOMMAREA.
   02 ...
```

Figure 73. Layout of data division in COBOL

```
PROCEDURE DIVISION.

A-CONTROL SECTION.

A-0000-MAIN-TASK.

    MOVE URP-INVALID TO DECODE-RESPONSE.

    IF CONVERTER-FUNCTION = URP-GETLENGTHS
    PERFORM B-0000-GETLENGTHS END-IF.

    IF CONVERTER-FUNCTION = URP-DECODE THEN
    PERFORM C-0000-DECODE END-IF.

    IF CONVERTER-FUNCTION = URP-ENCODE THEN
    PERFORM D-0000-ENCODE END-IF.

A-9999-EXIT.

    EXEC CICS RETURN END-EXEC.
    GOBACK.
```

Figure 74. Routing control to the functions in COBOL

In this program fragment:

1. The response URP-INVALID is set.
2. The IF statements examine the function code in the communication area, and pass control to the appropriate function.
3. The converter returns to the program that called it. (If the IF statements selected a function, the DECODE-RESPONSE value returned is the response from that function.)

Figure 75 is an example of a **Decode** function.

```
C-0000-DECODE.

    IF DECODE-EYECATCHER IS NOT = DECODE-EYECATCHER-INIT
      MOVE URP-INVALID TO DECODE-RESPONSE
    ELSE
      SET ADDRESS OF CLIENT-IN-DATA TO DECODE-CLIENT-DATA-PTR
      ADD 8 TO REM-PROC-COMMSIZE
      EXEC CICS GETMAIN
                       SET(DECODE-RETURNED-DATA-PTR)
                       FLENGTH(REM-PROC-COMMSIZE)
                       SHARED
                       NOSUSPEND
                       CICSDATAKEY
                       RESP(RESP)
                       RESP2(RESP2)
                       END-EXEC
    SET ADDRESS OF DECODE-RETURNED-DATA
                TO DECODE-RETURNED-DATA-PTR
    MOVE "PASSWD" TO DECODE-PASSWORD
    SET ADDRESS OF REM-PROC-DATA
                TO ADDRESS OF DECODE-SERVER-INPUT-DATA
    MOVE CLIENT-IN-U-CHAR TO REM-PROC-U-CHAR
    MOVE CLIENT-IN-CHAR TO REM-PROC-CHAR
    MOVE URP-OK TO DECODE-RESPONSE.
```

Figure 75. Example of a Decode function in COBOL

In this program fragment, the names beginning DECODE- (except DECODE-PASSWORD) are fields in the communication area for the **Decode** function. DECODE-PASSWORD is the field at the beginning of the returned data. The processing is as follows:

1. The eyecatcher is checked to see if it agrees with the function code. If it does not, the URP-INVALID response is returned.
2. If it does:
   a. The structure CLIENT-IN-DATA is overlaid on the data coming from the inbound XDR routine addressed by DECODE-CLIENT-DATA-PTR.
   b. The communication area size is increased by 8 to allow for the password field.
   c. EXEC CICS GETMAIN is used to get storage for the password and for the communication area. REM-PROC-COMMSIZE is the size of the structure REM-PROC-DATA, which defines the format of the communication area. The address of the storage is put directly into DECODE-RETURNED-DATA-PTR.
   d. The structure DECODE-RETURNED-DATA is overlaid on the newly-acquired storage addressed by DECODE-RETURNED-DATA-PTR.
   e. The password is moved into DECODE-PASSWORD.
   f. The data is moved from CLIENT-IN-DATA to REM-PROC-DATA, and the response is set to URP-OK.

Using converters

Converters run as CICS programs under the connection manager, server controller, and aliases. Converters must reside in the same CICS system as CICS ONC RPC.

Preparation

Before using a converter, you must:

1. Translate the converter using the appropriate CICS translator. If it is a COBOL program, you must use the QUOTE translator directive.
2. Compile the output from the translator.
3. Link the converter as a standard CICS application program into a CICS load library used by the CICS system on which CICS ONC RPC is installed.
4. Define the converter to CICS as a program.
5. Use the connection manager to specify the converter in one of the 4-tuple definitions, and define which of the converter functions are required for that 4-tuple.

Reference information for the converter functions

This section contains reference material for each of the three functions of a converter. Each function is documented in the same way:

- A summary table of parameters, showing which are for input only, which for input and output, and which for output only.
  – **Input** is for parameters that your function may consult, but not change.
  – **Inout** is for parameters that your function may consult, and change.
  – **Output** is for parameters that your function must not consult, but may change.
- A description of the processing that the function is expected to do.

- A list of parameters in alphabetical order, with a description of how CICS ONC RPC sets up the inputs, and what use it makes of the outputs.
- A list of the responses and reason codes that the converter can return, with a description of the action that CICS ONC RPC takes for each response and reason code.

The descriptions give the names of the program elements as they appear in C. In COBOL the names are all in uppercase, and the underscores are replaced by hyphens.

Getlengths

Summary of parameters

The names of the parameters are given in abbreviated form: each name in the table must be prefixed with **glength_** to give the name of the parameter.

To find the C type of each parameter, consult the header file DFHRPCDH provided with CICS ONC RPC. For COBOL, consult the copybook DFHRPCDO.

Input glength_	Inout glength_	Output glength_
eyecatcher **function**	**server_data_format**	**server_input_data_len** **server_output_data_len** **response** **reason**

Function

Getlengths is called when the definition of the 4-tuple is being registered, provided that the definition of the 4-tuple specified that **Getlengths** was to be called. It is not called to process client requests.

Getlengths is responsible for providing CICS ONC RPC with:

- The size of the data that is passed to and from the CICS program
- The data format (contiguous or overlaid) of the CICS program data

Parameters

glength_eyecatcher
(Input only)

A string of length 8. (The values of the eyecatchers are defined in the DFHRPUCH header file and the DFHRPUCO copybook.)

glength_function
(Input only)

A code indicating that **Getlengths** is being called. The value is URP_GETLENGTHS.

glength_reason
(Output only)

A reason code—see “Response and reason codes” on page 294.

glength_response
(Output only)

A response code—see “Response and reason codes” on page 294.

glength_server_data_format
(Input and output)

On input, that value specified for Server Data Format for the 4-tuple in the connection manager.

On output, the value is to control:

- How the input data pointer for **Encode** will be set up
- How the communication area length to be checked by the connection manager is calculated

The values you can supply are as follows:

URP_CONTIGUOUS
The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area plus the output value of **glength_server_input_data_len**, though **Decode** can modify this offset.

The connection manager calculates a communication area length by adding the output values of **glength_server_input_len** and **glength_server_output_len**. If this length is different from the actual length of the communication area passed to the CICS program, errors might occur in the processing of client requests.

URP_OVERLAID
The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area.

The connection manager calculates a communication area length by taking the larger of the output values of **glength_server_input_len** and **glength_server_output_len**. If this length is different from the actual length of the communication area passed to the CICS program, errors might occur in the processing of client requests.

glength_server_input_data_len
(Output only)

For the use of this field, see the description of **glength_server_data_format**. If you do not set a value in this field, a default value of zero is used.

glength_server_output_data_len
(Output only)

For the use of this field, see the description of **glength_server_data_format**. If you do not set a value in this field, a default value of zero is used.

Response and reason codes

You must return one of the following values in the **glength_response** field:

URP_OK
The connection manager checks that the communication area length does not exceed 32 767. If it does not, the information is saved and used to process incoming client requests, and the 4-tuple is registered. If it does, the connection manager writes an exception trace entry (trace point 9EE6), sends a message (DFHRP1991) describing the error to the terminal from which the connection manager was started, and does not register the 4-tuple.

URP_EXCEPTION
The connection manager writes an exception trace entry (trace point 9EE5), sends a message (DFHRP1988) to the terminal from which the connection manager was started, and does not register the 4-tuple.

URP_INVALID
The connection manager writes an exception trace entry (trace point 9EE5),

sends a message (DFHRP1989) to the terminal from which the connection manager was started, and does not register the 4-tuple.

URP_DISASTER
The connection manager writes an exception trace entry (trace point 9EE5), sends a message (DFHRP1990) to the terminal from which the connection manager was started, and does not register the 4-tuple.

If you return any other value in **glength_response**, it is treated as URP_DISASTER.

You can supply a 32-bit reason code in conjunction with the response value to provide further information in error cases. CICS ONC RPC does not take any action on the reason code returned by **Getlengths**. The reason code is output in any trace that results from the invocation of **Getlengths**, and you may use it as a debugging aid.

See “Numeric values of response and reason codes” on page 314 for the numeric values of the response codes in trace output.

Decode

Summary of parameters

The names of the parameters are given in abbreviated form: each name in the table must be prefixed with **decode_** to give the name of the parameter.

To find the C type of each parameter, consult the header file DFHRPCDH provided with CICS ONC RPC. For COBOL, consult the copybook DFHRPCDO.

Input decode_	Inout decode_	Output decode_
eyecatcher **function** **client_address** **client_data_ptr** **server_data_format** **program_number** **version_number** **procedure_number** **aup_time** **aup_machname_ptr** **aup_machlen** **aup_uid** **aup_gid** **aup_len** **aup_gids_ptr**	**server_program** **alias_transid** **server_input_data_len** **server_output_data_len**	**returned_data_ptr** **userid** **user_token** **response** **reason**

Function

Decode is invoked by the server controller after the inbound XDR routine. **Decode** processing must avoid making the server controller wait for resources, as this prevents the server controller from dealing efficiently with other requests. **Decode** has four main responsibilities:

- To set data lengths for the CICS program when the lengths are not the same for all requests.
- To map the input data passed from the inbound XDR routine to the input data format required by the CICS program.
- To set the user ID and password that are used to control subsequent processing.
- To set the name of the alias and CICS program for the request if those specified for the 4-tuple need to be changed.

Decode must issue EXEC CICS GETMAIN to allocate storage for the communication area to be passed to the CICS program. Note the following points about GETMAIN options:

- You must use the SHARED option, since the storage is acquired under the server controller, but is used under the alias.
- You must use the FLENGTH option.
- You must use the NOSUSPEND option to prevent the server controller from being made to wait for storage, as this would prevent the server controller from attending to incoming requests.
- To prevent overwriting by user-key programs, you should consider using the CICSDATAKEY option in the following circumstances:
  - The CICS program to be called by the alias is in another CICS system.
  - The CICS program to be called by the alias is defined as EXECKEY(CICS).

– The CICS program to be called by the alias is defined as EXECKEY(USER), but the amount of data to be copied is small.

If an overlaid data format is specified, the requested length must be the greater of the output values of **decode_server_input_data_len** and **decode_server_output_data_len**. If the data format is not overlaid, this length must be the sum of the output values of **decode_server_input_data_len** and **decode_server_output_data_len**.

Because **Decode** specifies the SHARED option, the data remains available to CICS ONC RPC modules and to CICS programs. CICS ONC RPC frees the storage when it is no longer required.

Parameters

decode_alias_transid
(Input and output)

On input, the name of the alias associated with the 4-tuple for the client request.

On output, the name of the transaction to be started by the server controller to process this client request.

See “Changing the alias and CICS program” on page 284.

decode_aup_gid
(Input only)

The client’s UNIX group id.

decode_aup_gids_ptr
(Input only)

A pointer to an array of 32-bit integers that are the UNIX group ids of which the client is a member.

decode_aup_len
(Input only)

The number of elements in the array of UNIX group identifiers pointed to by **decode_aup_gids_ptr**.

decode_aup_machlen
(Input only)

The number of characters in the machine name.

decode_aup_machname_ptr
(Input only)

A pointer to a variable length character string representing the name of the machine on which the client is executing.

decode_aup_time
(Input only)

The time at which the client created the credentials. The time is measured in seconds since 00h00m GMT on 1 January 1970.

decode_aup_uid
(Input only)

The client’s UNIX user ID.

decode_client_address
(Input only)

The 32-bit internet address of the client from which the request was received.

decode_client_data_ptr
(Input only)

A pointer to the data passed from the client. If there is no data, this pointer points to a null string.

Note: The data area pointed to by this pointer must not be changed by **Decode**, or CICS storage management errors are likely to occur.

decode_eyecatcher
(Input only)

A string of length 8. (The values of the eyecatchers are defined in the DFHRPUCH header file and the DFHRPUCO copybook.)

decode_function
(Input only)

A code indicating that **Decode** is being called. The value is URP_DECODE.

decode_procedure_number
(Input only)

The procedure number of the 4-tuple to which the client request was made.

decode_program_number
(Input only)

The program number of the 4-tuple to which the client request was made.

decode_reason
(Output only)

A reason code—see “Response and reason codes” on page 300.

decode_response
(Output only)

A response code—see “Response and reason codes” on page 300.

decode_returned_data_ptr
(Output only)

A pointer to an area of storage allocated by the converter that contains:

- **decode_password**—the password to be used for user authentication
- **decode_server_input_data**—the data that is to be passed to the CICS program as input.

It may be null if there is no password and if no data is to be passed to the CICS program.

decode_server_data_format
(Input only)

A value that controls:

- How the input data pointer for **Encode** will be set up
- How the communication area length to be checked by the connection manager is calculated

URP_CONTIGUOUS

The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area plus the output value of **decode_server_input_data_len**.

The server controller calculates a communication area length by adding the output values of **decode_server_input_data_len** and **decode_server_output_data_len**. If this length is different from the actual length of the communication area passed to the CICS program, errors might occur in the processing of client requests.

URP_OVERLAID

The value of the data pointer that will be passed to **Encode**, or to the outbound XDR routine if **Encode** is not used for this 4-tuple, is the address of the CICS program communication area.

The server controller calculates a communication area length by taking the larger of the output values specified of **decode_server_input_data_len** and **decode_server_output_data_len**. If this length is different from the actual length of the communication area passed to the CICS program, errors might occur in the processing of client requests.

decode_server_input_data_len

(Input and output)

On input, the output value of **glength_server_input_data_len**, or the value specified for Server Input Length for this 4-tuple in the connection manager.

On output, see the description of **decode_server_data_format**.

decode_server_output_data_len

(Input and output)

On input, the output value of **glength_server_output_data_len**, or the value specified for Server Output Length for this 4-tuple in the connection manager.

On output, see the description of **decode_server_data_format**.

decode_server_program

(Input and output)

On input, the name of the CICS program associated with the 4-tuple for the client request.

On output, the name of the CICS program to be linked to by the alias.

You should use this field if you want to direct the client call to a different CICS program.

decode_userid

(Output only)

An 8-character field, the user ID known to CICS that correlates to the requesting client ID. If you store no value in this field, the user ID used in subsequent processing is the default CICS user ID.

decode_user_token

(Output only)

A fullword that may be used to pass information to the **Encode** function that is subsequently invoked for the client request.

decode_version_number
(Input only)

The version number of the 4-tuple to which the client request was made.

Response and reason codes

You must return one of the following values in the **decode_response** field:

URP_OK
The server controller checks that the communication area length does not exceed 32 767. If it does not, the alias is started using the information supplied as output. If it does, the server controller writes an exception trace entry (trace point 9FC2), and issues a message (DFHRP0516) describing the error. The alias is not started, and an **svcerr_systemerr** call is used to send a reply to the client.

URP_EXCEPTION
The server controller writes an exception trace entry (trace point 9FAA), and issues a message that depends on the reason code:

- URP_CORRUPT_CLIENT_DATA—message DFHRP0626

  An **svcerr_decode** call is used to send a reply to the client.
- URP_AUTH_BADCRED—message DFHRP0628

  An **svcerr_auth** call with a why-value of AUTH_BADCRED is used to send a reply to the client.
- URP_AUTH_TOOWEAK—message DFHRP0629

  An **svcerr_auth** call with a why-value of AUTH_TOOWEAK is used to send a reply to the client.
- Any other value—message DFHRP0631

  An **svcerr_systemerr** call is used to send a reply to the client.

URP_INVALID
The server controller writes an exception trace entry (trace point 9FAA), and issues a message (DFHRP0632).

An **svcerr_systemerr** call is used to send a reply to the client.

URP_DISASTER
The server controller writes an exception trace entry (trace point 9FAA), and issues a message (DFHRP0635).

An **svcerr_systemerr** call is used to send a reply to the client.

If you return any other value in **decode_response**, the server controller writes an exception trace entry (trace point 9FAA), and issues a message (DFHRP0625). An **svcerr_systemerr** call is used to send a reply to the client.

You can supply a 32-bit reason code in conjunction with the response value to provide further information in error cases. CICS ONC RPC does not take any action on the reason code returned by **Decode**, except as indicated above under URP_EXCEPTION. The reason code is output in any trace that results from the invocation of **Decode**, and you may use it as a debugging aid.

See “Numeric values of response and reason codes” on page 314 for the numeric values of the response and CICS-defined reason codes in trace output.

Encode

Summary of parameters

The names of the parameters are given in abbreviated form: each name in the table must be prefixed with **encode_** to give the name of the parameter.

To find the C type of each parameter, consult the header file DFHRPCDH provided with CICS ONC RPC. For COBOL, consult the copybook DFHRPCDO.

Input encode_	Inout encode_	Output encode_
eyecatcher **function** **input_data_ptr** **input_data_len** **user_token**	none	**output_data_ptr** **output_data_len** **response** **reason**

Function

Encode is called by the alias after the CICS program ends. **Encode** is responsible for taking the data returned from the CICS program and changing its format so that it is suitable to be passed to the outbound XDR routine for return to the client.

If no restructuring of outbound data is required, you can specify to the connection manager that **Encode** is not to be called.

The reference to the CICS program data to be returned to the client is passed to **Encode** in the **encode_input_data_ptr** input field. This data is in CICS program format, which is a communication area structure in any CICS supported language. The CICS program data may be mapped from this format into the format required by the client, which is likely to be C, and might include pointer references, by allocating an area of storage and mapping the server data into it.

Encode must set **encode_output_data_ptr** to point to the start of the allocated storage.

Encode must issue EXEC CICS GETMAIN to allocate storage for the data that it returns. Note the following points about GETMAIN options:

- You do not need to use the SHARED option.
- You must use the FLENGTH option.
- If your CICS system is using storage protection, you can use the CICSDATAKEY option to prevent overwriting by user-key programs.

Parameters

encode_eyecatcher
(Input only)

A string of length 8. (The values of the eyecatchers are defined in the DFHRPUCH header file and the DFHRPUCO copybook.)

encode_function
(Input only)

A code indicating that **Encode** is being called. The value is URP_ENCODE.

encode_input_data_len
(Input only)

The length in bytes of the data returned from the CICS program. The value is determined as follows:

1. It is the output value of **decode_server_output_data_len**, if **Decode** set it.
2. If **Decode** did not set the value, it is the output value of **glength_server_output_data_len**, if **Getlengths** was called when the 4-tuple was registered.
3. If neither of the above is the case, it is the value specified for Server Output Length in the connection manager when the 4-tuple was defined.

encode_input_data_ptr
(Input only)

A pointer to the data returned from the CICS program. The setting of this pointer depends on the definition of the 4-tuple in the connection manager, **Getlengths** processing when the 4-tuple was registered, and **Decode** processing for the client request.

encode_output_data_len
(Output only)

The length in bytes of the data to be passed to the outbound XDR routine.

encode_output_data_ptr
(Output only)

A pointer to an area of allocated storage that contains the data that is to be passed to the outbound XDR routine.

encode_reason
(Output only)

A reason code—see “Response and reason codes”.

encode_response
(Output only)

A response code—see “Response and reason codes”.

encode_user_token
(Input only)

A fullword containing information which was output from **Decode** for this client request.

Response and reason codes

You must return one of the following values in the **encode_response** field:

URP_OK
The alias passes the output data to the outbound XDR routine.

URP_EXCEPTION
The alias writes an exception trace entry (trace point 9F17), and issues a message (DFHRP0161). An **svcerr_systemerr** call is used to send a reply to the client.

URP_INVALID
The alias writes an exception trace entry (trace point 9F17), and issues a message (DFHRP0162). An **svcerr_systemerr** call is used to send a reply to the client.

URP_DISASTER
The alias writes an exception trace entry (trace point 9F17), and issues a message (DFHRP0169). An **svcerr_systemerr** call is used to send a reply to the client.

If you return any other value in **encode_response**, the alias writes an exception trace entry (trace point 9F17), and issues a message (DFHRP0163). An **svcerr_systemerr** call is used to send a reply to the client.

You can supply a 32-bit reason code in conjunction with the response value to provide further information in error cases. CICS ONC RPC does not take any action on the reason code returned by **Encode**. The reason code is output in any trace that results from the invocation of **Encode**, and you may use it as a debugging aid.

See “Numeric values of response and reason codes” on page 314 for the numeric values of the response in trace output.

Step 5—Writing a resource checker

This step is optional. See “Writing the resource checker” on page 307 for details.

Step 6—Compile and link

This step puts the programs you have written into CICS load libraries.

Converter

The header files needed to compile the converter are discussed in “Organizing the converter” on page 285.

The program is linked into a CICS load library, since it is a normal CICS program.

XDR routines

If your XDR routines are not just XDR library functions, you must compile each XDR routine separately and link it into a CICS load library. If you used RPCL to define the data, the XDR source and header files for the compilation have been generated by RPCGEN.

Resource checker

If you need a resource checker, you must link it into a CICS load library. It must be called DFHRPRSC.

Step 7—Make CICS definitions

You must define the CICS program, converter program, resource checker, and any XDR routines that are not just library routines to CICS. See “Defining CICS ONC RPC resources to CICS” on page 249.

Step 8—Make a connection manager entry

Use the connection manager to define each 4-tuple. Completing an entry for a 4-tuple in the connection manager ensures that you provide CICS ONC RPC with all the information that it needs to service the client request.

The fields used to define each 4-tuple are described in “Defining the attributes of a 4-tuple” on page 262.

Chapter 24. Security

This chapter contains Product-sensitive Programming Interface and Associated Guidance Information.

This chapter describes how CICS ONC RPC interacts with the security facilities of ONC RPC and CICS.

Security is an important concern in the provision of ONC RPC support in the CICS environment, because CICS ONC RPC provides an Open Systems communications interface into CICS.

This chapter is organized as follows:

Security in ONC RPC

ONC RPC has its own security methods (called authentication in RPC) with dedicated fields in the ONC RPC call and reply message headers. There are three types of RPC authentication:

- **UNIX authentication**, which is used to transmit the client’s UNIX user ID, group ID, and other identification information.
- **Data Encryption Standard (DES) authentication**, which is not available at ONC RPC Version 3.9, and so cannot be used with CICS ONC RPC.
- **Null authentication**, which offers no security checking.

Security in CICS and its effect on CICS ONC RPC operations

During the operation of CICS ONC RPC, various CICS commands are used to make security checks with an external security manager (ESM). The checks will always give positive results if SEC=NO is specified as a system initialization parameter. The checks will always give negative results if SEC=YES was specified, but the ESM abended while CICS was operating. The following discussion of the use made of CICS security commands assumes that SEC=YES is specified, and that the ESM is active.

- When a transaction whose user ID is *userid1* issues EXEC CICS START USERID(*userid2*), a surrogate-user check is made with the ESM to see that *userid1* is authorized to use *userid2*. The check is made only if XUSER=YES is specified as a system initialization parameter.

  This command is issued when the connection manager starts the server controller, and each time the server controller starts an alias transaction. In the first case, the user ID used is the one supplied to the connection manager as CRPM Userid on panel DFHRP02. In the second case, the user ID used is the one output from **Decode**.
- EXEC CICS VERIFY PASSWORD is issued by the alias before it links to the CICS program that services the client request. A check is made with the ESM that the user ID and password are an acceptable combination.

© Copyright IBM Corp. 1994, 1998

- EXEC CICS QUERY SECURITY is used by the alias to check that the user ID under which it is executing is authorized to use the CICS program. The check is made only if XPPT=YES is specified as a system initialization parameter.
- During the operation of the CICS program, security checks are made each time the program tries to access a protected resource. The check is made only if RESSEC(YES) is specified in the definition of the alias transaction, and the system initialization parameter controlling security checking for the resource type is set to YES.
- During the operation of the CICS program, security checks are made each time the program tries to use a command from the CICS SPI (system programming interface). The check is made only if CMDSEC(YES) is specified in the definition of the alias transaction, and if XCMD=YES is specified as a system initialization parameter.

Figure 76 shows how CICS security interacts with the operation of CICS ONC RPC.

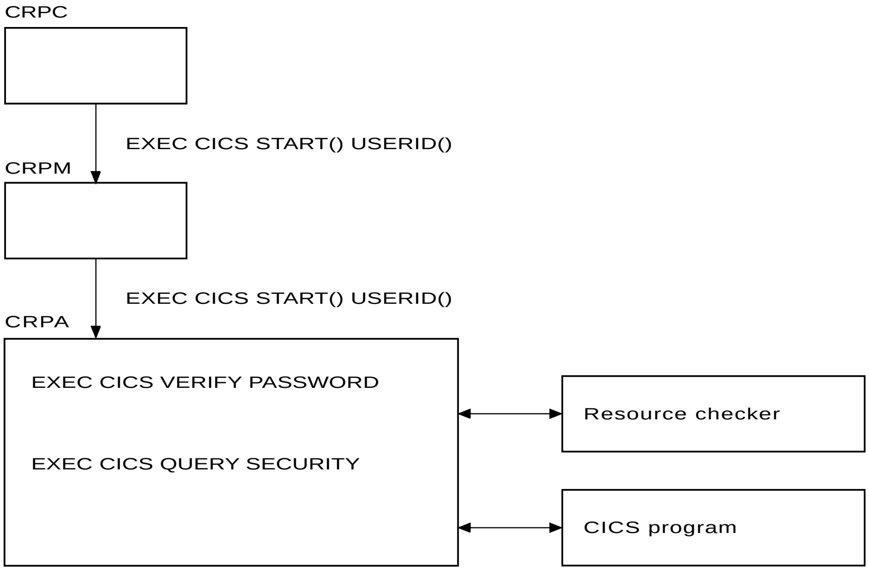

CRPC
EXEC CICS START() USERID()
CRPM
EXEC CICS START() USERID()
CRPA
EXEC CICS VERIFY PASSWORD
EXEC CICS QUERY SECURITY
Resource checker
CICS program

Figure 76. How CICS security interacts with CICS ONC RPC operations

The figure shows that the alias will link to the user-supplied resource checker program if one is configured, but the use of the resource checker program is not recommended. You should use the CICS security facilities, and make the appropriate definitions in the ESM.

Using RACF Secured Sign-on

RACF Secured Sign-on support allows clients to gain security access to CICS facilities by sending a PassTicket (that is, a one-time-only password). This avoids the security hazard of a password being transmitted across the network in clear text.

For further information, see *Resource Access Control Facility: System Programmer's Guide, Version 2 Release 2*. This includes details of the algorithm that the RPC client must use to generate the PassTicket. This algorithm includes the DES algorithm.

PassTicket generation

The algorithm that generates the PassTicket is a function of:

- The CICS user ID of the client
- The CICS application ID of the CICS region running CICS ONC RPC
- A secured sign-on application key, known to both sides
- A time and date stamp

To generate the PassTicket, the client must:

- Know its CICS user ID, the server CICS application ID, and the application key.
- Synchronize its clock to within ten minutes of the server.
- Have access to the encryption algorithm on its machine. Only the DES algorithm may be used.

Writing the resource checker

Your resource checker program must be called DFHRPRSC. There can be only one resource checker in a CICS region.

The resource checker allows you to check the credentials of inbound client requests.

The resource checker can check the client address, passed as an input parameter, against a list of known clients for the host on which the request has been received. The password passed to the resource checker is blank.

Reference information for the resource checker

The reference information for the resource checker is presented as follows:

- A summary table of parameters, showing which are for input only, and which for output only.
  - **Input** is for parameters that your resource checker may consult, but not change.
  - **Output** is for parameters that your resource checker must not consult, but may change.
- A description of the processing that the resource checker is expected to do.
- A list of parameters in alphabetical order, with a description of how CICS ONC RPC sets up the inputs, and what use it makes of the outputs.
- A list of the responses and reason codes that the resource checker can return, with a description of the action that CICS ONC RPC takes for each response and reason code.

The descriptions give the names of the program elements as they appear in C. In COBOL the names are all in uppercase, and the underscores are replaced by hyphens.

Summary of parameters

The format of the communication area containing the resource checker parameters is in the C header file DFHRPRDH, and the COBOL copybook DFHRPRDO. You will also need values defined in the C header file DFHRPUCH, or in the COBOL copybook DFHRPUCO.

Input	Output
res_check_alias_transid **res_check_cics_password_ptr** **res_check_cics_userid** **res_check_client_ip_address** **res_check_eyecatcher** **res_check_host_ip_address** **res_check_server_program_name**	**res_check_reason** **res_check_response**

Function

The resource checker is optionally invoked by the alias before it attempts to link to the CICS program that is to service the client request. It must say whether the client request is allowed to proceed.

Parameters

res_check_alias_transid
(Input only)

The 4-character name of the alias transaction that has linked to the resource checker.

res_check_cics_password_ptr
(Input only)

A pointer to the 8-character password passed from the requesting client or supplied by **Decode**. The value of this field is blank, and it is provided for compatibility with earlier versions of CICS ONC RPC.

res_check_cics_userid
(Input only)

The 8-character CICS user ID under which the alias is running.

res_check_client_ip_address
(Input only)

The fullword internet address of the client.

res_check_eyecatcher
(Input only)

A string of length 8. (Its value is defined in the header file DFHRPUCH and the copybook DFHRPUCO).

res_check_host_ip_address
(Input only)

The fullword internet address of the TCP/IP for MVS host with which the server controller is in communication.

res_check_reason
(Output only)

The reason to be returned to the alias.

res_check response
(Output only)

The response to be returned to the alias.

res_check_server_program_name
(Input only)

The 8-character name of the CICS program that is to be invoked to perform the server function requested by the client.

Response and reason codes

You must return one of the following values in the **res_check_response** field.

URP_OK
The alias will continue to process the client request.

URP_EXCEPTION
The alias writes an exception trace entry (trace point 9F0E), and issues a message that depends on the reason code:

- URP_AUTH_BADCRED—message DFHRP0130

  An **svcerr_auth** call with a why-value of AUTH_BADCRED is used to send a reply to the client.
- URP_AUTH_TOOWEAK—message DFHRP0184

  An **svcerr_auth** call with a why-value of AUTH_TOOWEAK is used to send a reply to the client.
- Any other value—message DFHRP0185

  An **svcerr_systemerr** call is used to send a reply to the client.

URP_INVALID
The alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0186).

An **svcerr_systemerr** call is used to send a reply to the client.

URP_DISASTER

The alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0187).

An **svcerr_systemerr** call is used to send a reply to the client.

If you return any other value in **res_check_response**, the alias writes an exception trace entry (trace point 9F0E), and issues a message (DFHRP0188). An **svcerr_systemerr** call is used to send a reply to the client.

You can supply a 32-bit reason code in conjunction with the response value to provide further information in error cases. CICS ONC RPC does not take any action on the reason code returned by the resource checker, except as indicated above under URP_EXCEPTION. The reason code is output in any trace or messages that result from the resource checker, and you may use it as a debugging aid.

See "Numeric values of response and reason codes" on page 314 for the numeric values of the response and CICS-defined reason codes in trace output.

Chapter 25. Problem determination

This chapter contains Diagnosis, Modification, or Tuning Information.

This chapter helps you debug problems in CICS ONC RPC user-replaceable programs, the IBM-supplied parts of CICS ONC RPC, and in the system setup of CICS ONC RPC. If you suspect that you have a problem in another part of CICS, refer to the *CICS Problem Determination Guide*.

The formats of messages and trace outputs in CICS ONC RPC are also described.

Diagnostic information is designed to provide first failure data capture, so that if an error occurs, enough information about the error is available directly without having to reproduce the error situation. The information is presented in the following forms:

Messages
CICS ONC RPC provides CICS messages. The CICS ONC RPC messages are listed in *CICS Messages and Codes*.

Trace CICS ONC RPC outputs system trace entries containing all the important information required for problem diagnosis.

Dump Dump formatting is provided for data areas relating to CICS ONC RPC.

Abend codes
Transaction abend codes are standard 4-character names. The abend codes output by CICS ONC RPC are listed in *CICS Messages and Codes*.

This chapter is organized as follows:

- “Recovery procedures”
- “MVS task control blocks (TCBs) used” on page 312
- “Troubleshooting” on page 312
- “Using messages and codes” on page 314
- “CICS ONC RPC trace information” on page 314
- “Dump and trace formatting” on page 315
- “Debugging the user-replaceable programs” on page 315
- “Operational problems” on page 316.

Recovery procedures

Software errors within the server controller may cause it to perform an immediate disable (if this is not prevented by the nature of the error). After an immediate disable of CICS ONC RPC, CICS continues to run.

CICS ONC RPC is not included in CICS recovery. Registration details are not saved on the CICS catalog. If CICS abends and is then restarted, RPC interface registrations from the previous run are not preserved. After a CICS abend, you must enable CICS ONC RPC as described in “Enabling CICS ONC RPC” on page 259. However, 4-tuple definitions can be stored in the CICS ONC RPC data set. Each time you enable CICS ONC RPC, the definitions can be retrieved from the CICS ONC RPC data set.

© Copyright IBM Corp. 1994, 1998

If TCP/IP for MVS abends, CICS ONC RPC enters immediate disable processing, but CICS continues to run.

The abending of an alias transaction might cause changes to recoverable resources to be backed out. See “Updating recoverable resources” on page 242.

CICS immediate shutdown might leave 3-tuples registered with TCP/IP for MVS. These 3-tuples can be registered again when CICS ONC RPC is enabled without loss of TCP/IP for MVS resources, since CICS ONC RPC always unregisters a 3-tuple before it registers it.

MVS task control blocks (TCBs) used

The TCB that interacts with TCP/IP for MVS goes into a wait as a result of that interaction. If the wait were on the QR (quasi-reentrant) TCB, all the transactions in the whole CICS region would be delayed. This is avoided by using an extra TCB, the RP TCB, for issuing calls to TCP/IP for MVS.

The RP TCB is used for some processing for every client request, but most of the call processing done by CICS ONC RPC takes place under the QR TCB. The split between the two TCBs is transparent to you for most of your work, but you need to be aware of it for problem determination.

Task-related user exit (TRUE)

CICS ONC RPC includes a task-related user exit; this is used to anchor shared storage and to improve CICS ONC RPC’s response to CICS shutdown. CICS ONC RPC does not use a TRUE to pass commands and data to and from TCP/IP for MVS.

Troubleshooting

This section provides some hints on troubleshooting. It follows the general outline:

1. Define the problem.
2. Obtain information (documentation) on the problem.

Defining the problem

When you have a problem, first try to define the circumstances that gave rise to it. If you need to report the problem to the IBM software support center, this information is useful to the support personnel.

1. What is the system configuration?
   - CICS Transaction Server release
   - TCP/IP for MVS release
   - LE/370 release
   - OS/390 release
2. What is the connection manager configuration?
   - Operating options
   - Registered 4-tuples
3. When did the problem first occur?

4. What were you trying to accomplish at the time the problem occurred?
5. What changes were made to the system before the occurrence of the problem?
   - To the OS/390 system
   - To CICS ONC RPC
   - To the CICS program being called by the client
   - To the converter being used in the call
   - To the XDR routines being used in the call
   - To the client
   - To CICS Transaction Server
   - To TCP/IP for MVS
6. What is the problem?
   - Incorrect output
   - Hang/Wait: If you suspect that CICS ONC RPC aliases may be in a hung state, you can use the connection manager to display a list of alias transactions and can display associated details. See “Processing the alias list” on page 277.
   - Loop: Use CEMT INQUIRE to display the details of the transaction.
   - Abend in user-replaceable program
   - Abend in a CICS program
   - Abend in the IBM-supplied part of CICS ONC RPC
   - Performance problem
   - Storage violation
   - Logic Error
7. At what point in the processing did the problem occur? (Use Figure 48 on page 241.)
8. What was the state of TCP/IP for MVS? (Try the **rpcinfo** command.)

Documentation about the problem

To investigate most problems, you must look at the dumps, traces, and logs provided with MVS and CICS.

- System Dump: This contains the CICS internal trace
- CICS auxiliary trace, if enabled
- TCP/IP for MVS trace
- GTF trace, if enabled
- Console log
- CSMT log
- CRPO log
- CICS job log

To identify which are likely to be useful for your problem, try to work out the area of CICS ONC RPC giving rise to the problem, and read the relevant section in the rest of this chapter.

Using messages and codes

CICS ONC RPC messages have identifiers of the form DFHRP*nnnn*, where *nnnn* are four numeric characters. These numbers indicate which of the CICS ONC RPC components generated the message, as shown in *CICS Messages and Codes*.

Messages are generated for end users by the connection manager. They are sent to the CICS ONC RPC message transient data queue CRPO, or the terminal user, or both, depending on the event that is being reported. If you define CRPO as an indirect destination for CSMT, the CICS ONC RPC messages appear in CSMT. Some messages are sent to the console.

When CICS ONC RPC issues a message as a result of an error, it also makes an exception trace entry. CICS ONC RPC also generates information messages, for instance during enable processing and disable processing.

CICS ONC RPC messages are supplied in English, Kanji, and Chinese. The CICS message editing utility can be used to translate them into other languages supported by CICS.

CMAC (online help facility for messages and codes)

You can use utilities supplied as part of CICS to update your base CMAC file with the CICS ONC RPC CMAC file.

The CICS ONC RPC abend codes are listed in *CICS Messages and Codes*.

CICS ONC RPC trace information

CICS ONC RPC outputs CICS system trace, which is formatted using software supplied as part of CICS ONC RPC.

Exception trace entries produced by CICS ONC RPC are written to CICS internal trace even when the Trace operating option is set to NO. See “Setting and modifying options” on page 259 for information about the Trace option.

If selected, level 2 trace gives a full trace of the data being transmitted between the client and the CICS program. CICS trace output is described in the *CICS Problem Determination Guide*, and details of the contents of each trace points are given in the *CICS User’s Handbook*.

Numeric values of response and reason codes

The response codes from the converter and resource checker appear in the trace output as numeric values as follows:

- URP_OK (0)
- URP_EXCEPTION (4)
- URP_INVALID (8)
- URP_DISASTER (12)

The CICS-defined reason codes from the converter and resource checker appear in the trace output as numeric codes as follows:

- URP_AUTH_BAD_CRED (1)
- URP_AUTH_TOO_WEAK (2)
- URP_CORRUPT_CLIENT_DATA (3)

Dump and trace formatting

To switch dump formatting on and off for CICS ONC RPC, you change the CICS VERBEXIT in the JCL for dump formatting as follows:

```
IPCS VERBEXIT CICS530 FT=0|1|2|3,TR=1|2
```

The parameters have these meanings:

FT=0 Suppress system dump for all features

FT=1 Produce system dump summary listing for all registered features

FT=2 Produce system dump for all registered features

FT=3 Produce system dump summary listing and a system dump for all registered features

TR=1 Produce abbreviated trace (includes trace for all registered features)

TR=2 Produce full trace (includes trace for all registered features)

Full details of these and other parameters are described in the *CICS Operations and Utilities Guide*.

CICS ONC RPC output in the formatted dump consists of the major control blocks of CICS ONC RPC, with interpretation of some of the fields. The CICS ONC RPC output can be found in the IPCS output by searching for `===RP`. It is under the heading `CICS ONC RPC Feature for MVS/ESA`.

Each trace entry for CICS ONC RPC has a comment `ONC RPC` to distinguish it from other trace points with the FT prefix.

Debugging the user-replaceable programs

The user-replaceable programs are:

- The user-written XDR routines
- The converters
- The resource checker
- The CICS programs that service the client requests

The debugging of the CICS programs is not dealt with in this manual.

XDR routines

The XDR routines, inbound and outbound, run under the RP TCB. The CICS application programming interface is not available under the RP TCB, so you cannot use EDF, CICS abend handling, or CICS trace to diagnose problems. The **printf** function must not be used. If an XDR routine has a program check, a C run-time message is written to the CICS job log.

Converter and resource checker

The converter and resource checker run under the QR TCB, and the CICS application programming interface is available.

Using EDF

EDF is available for debugging the resource checker and the **Encode** function. If you want to use EDF, you must:

- Ensure that the alias is terminal-attached. To do this, you must set the EDF Terminal ID field in the connection manager, as described on page 264. The chosen terminal must be a local terminal that is either logged on, or predefined.
- Define CEDF(YES) in the program definition of converter, or resource checker. The resource checker must run in the same CICS region as CICS ONC RPC if you want to use EDF to debug it.

Using trace entries

Diagnostic information can be output to the CICS trace by the use of the EXEC CICS ENTER TRACENUM command. The amount of trace information and the information contained within trace entries is at your discretion. See the *CICS Application Programming Reference* for more information about this command.

Writing messages

Diagnostic messages can be output by using EXEC CICS WRITEQ TD. Message information content, message format, frequency, and destination are at your discretion.

Abends

You are recommended to use EXEC CICS HANDLE ABEND to trap abends. You should collect the diagnostic information you need by tracing, and other forms of diagnostic output, and then return a URP_DISASTER response.

Operational problems

The server controller uses EXEC CICS START to start the aliases that run the CICS programs. CICS limits on the numbers of tasks that can be started may prevent aliases from running as soon as they are started by the server controller. This leads to delays in servicing the client requests, and this may lead to time-outs in the client.

In the XDR routines, storage allocation is done using MVS facilities, not CICS facilities. The storage is owned by the RP TCB. If an XDR routine abends, the storage is not freed by the server controller or the alias, nor is it freed by MVS, since the RP task does not end. Repeated abends in XDR routines may lead to shortage of storage that can only be corrected by stopping CICS.

Chapter 26. Performance and tuning

This chapter contains Diagnosis, Modification, or Tuning Information.

The performance of a single client request is affected by various aspects of the client, the network, CICS ONC RPC, the user-replaceable programs, and CICS.

This chapter is organised as follows:

- "The client"
- "The network"
- "TCP/IP for MVS resources"
- "CICS ONC RPC"
- "The user-replaceable programs"
- "General concepts" on page 6.

The client

The client time-out interval must take account of the possible delays in dealing with a client request in CICS ONC RPC and in CICS.

If a client request cannot be processed, an error reply is sent to the client.

The network

This manual does not deal with performance problems of TCP/IP networks.

TCP/IP for MVS resources

If, while registering 4-tuples, you cause the connection manager to register too many 3-tuples with TCP/IP for MVS, you might reduce the service that CICS ONC RPC can give to incoming client requests. See "Limits on registration" on page 267.

CICS ONC RPC

The allocation of different alias transaction names to different 4-tuples must be coordinated with the priorities given to the transactions in CICS.

The user-replaceable programs

There are some performance considerations for user-replaceable programs.

The converter

Getlengths is called only by the connection manager, and has no effect on the performance of a single client request, or on throughput.

© Copyright IBM Corp. 1994, 1998

Decode is called by the server controller. Delays here can reduce the throughput of CICS ONC RPC as well as reducing the performance of a single client request. The following recommendations are made:

- Do not use CICS trace here except to solve specific problems.
- Use NOSUSPEND on EXEC CICS GETMAIN. If GETMAIN errors occur because there is not enough storage, look for solutions that do not involve using the SUSPEND option.

Encode is called by the alias. Delays here reduce the performance of single client requests, but not the throughput of CICS ONC RPC.

The resource checker

The resource checker is called by the alias, so delays here affect the performance of a single client request, but have no effect on throughput.

Part 5. Using CICS as a DCE server

This part of the book describes CICS support for DCE remote procedure calls (RPCs). This support enables a non-CICS client program running in an open systems Distributed Computing Environment (DCE) to call a server program running in a CICS system and to pass and receive data using a communications area. The CICS program is invoked as if linked-to by another CICS program.

This part contains:

© Copyright IBM Corp. 1994, 1998

Chapter 27. Introduction to the Distributed Computing Environment?

This chapter tells you what the Distributed Computing Environment (DCE) is and why you might want to use it. For more detailed information, you should refer to the books listed in “Where to find more information” on page 327.

The chapter contains the following sections:

- “What is DCE?”
- “CICS support for DCE” on page 323.

What is DCE?

DCE was developed by the Open Software Foundation (OSF) as an Open Systems platform to address the challenges of distributed computing. It is being ported to all major IBM and many non-IBM environments. Note that all current DCE implementations use TCP/IP rather than SNA as their communication protocol.

DCE is based on three distributed computing models:

Client/server
A way of organizing a distributed application

Remote procedure call
A way of communicating between parts of a distributed application

Shared files
A way of handling data in a distributed system, based on a personal computer file access model.

Note: CICS alone (without DCE) also supports distributed computing. See “Distributed computing” on page 6.

The rest of this section gives a high level view of the services provided by DCE.

Remote procedure call (RPC)

One way of implementing communications between a client and a server of a distributed application is to use the procedure call model. In this model, the client makes what looks like a procedure call, and waits for a reply from the server. The procedure call is translated into network communications by the underlying RPC mechanism. The server receives a request and executes the procedure, returning the results to the client.

In DCE RPC, you define one or more DCE RPC interfaces, using the DCE interface definition language (IDL). Each interface comprises a set of associated RPC calls (called *operations*), each with their input and output parameters. You compile the IDL, which generates data structure definitions and executable stubs for both the client and the server. The matching parameter data structures ensure a common view of the parameters by both client and server. The matching client and server executable stubs handle the necessary data transformations to and from the network transmission format, and between different machine formats (EBCDIC and ASCII).

© Copyright IBM Corp. 1994, 1998

You use the DCE Directory Service to advertise that your server now supports the new interface you defined using the IDL. Your client code can likewise use the Directory Service to discover which servers provide the required interface.

You can also use the DCE Security Service to ensure that only authorized client end users can access your newly defined server function.

Directory Service

The DCE Directory Service is a central repository for information about resources in the distributed system. Typical resources are users, machines, and RPC-based services. The information consists of the name of the resource and its associated attributes. Typical attributes could include a user’s home directory, or the location of an RPC-based server.

The DCE Directory Service consists of several parts: the Cell Directory Service (CDS), the Global Directory Service (GDS) [4] , the Global Directory Agent (GDA), and a Directory Service programming interface. The CDS manages a database of information about the resources in a group of machines called a DCE *cell*. The Global Directory Service implements an international, standard directory service and provides a global namespace that connects the local DCE cells into one worldwide hierarchy. The GDA acts as a go-between for cell and global directory services. Both CDS and GDS are accessed using a single Directory Service application programming interface (API).

Security Service

There are three aspects to DCE security: authentication, secure communications, and authorization. They are implemented by several services and facilities that together comprise the DCE Security Service. These include the Registry Service, the Authentication Service, the Privilege Service, the Access Control List (ACL) Facility, and the Login Facility.

The identity of a DCE user or service is authenticated by the Authentication Service. Communications are protected by the integration of DCE RPC with the Security Service. Communication over the network can be checked for tampering or encrypted for privacy. Finally, access to resources is controlled by comparing the credentials conferred to a user by the Privilege Service with the rights to the resource, which are specified in the resource’s Access Control List. The Login Facility initializes a user’s security environment, and the Registry Service manages the information (such as user passwords) in the DCE Security database.

Time Service

The DCE Time Service (DTS) provides synchronized time on the computers participating in a Distributed Computing Environment. DTS synchronizes a DCE host’s time with Coordinated Universal Time (UTC), an international time standard. DTS cannot keep the time in each machine precisely the same, but can maintain it to a known accuracy. DTS also provides services which return a time range to an application (rather than a single time value), and which compare time ranges from different machines. They can be used to schedule and synchronize events across the network.

[4] The Global Directory Service is not currently supported by OS/390 Unix Systems Services DCE Base Services MVS/ESA.

File Service

The DCE File Service (DFS) allows users to access and share files stored on a File Server anywhere on the network, without having to know the physical location of the file. Files are part of a single, global namespace. A user anywhere on a network can access any file, just by knowing its name. The File Service achieves high performance, particularly through caching of file system data. Many users can access files that are located on a given File Server without a large amount of network traffic or delays.

Note: The File Service is based on a personal computer view of files, and is not relevant to the CICS Transaction Server for OS/390 environment.

Threads

DCE Threads supports the creation, management, and synchronization of multiple threads of control within a single process. This component is conceptually a part of the operating system layer, the layer below DCE. If the host operating system already supports threads, DCE can use that software and DCE Threads is not necessary. Because all operating systems do not provide a threads facility and DCE components require threads be present, this user-level threads package is included in DCE.

CICS support for DCE

CICS Transaction Server for OS/390 supports DCE remote procedure calls.

In conjunction with the OS/390 Unix Systems Services DCE Base Services MVS/ESA and OS/390 Unix Systems Services DCE Application Support MVS/ESA CICS Feature products, CICS Transaction Server for OS/390 enables a CICS program to act as a server for a DCE RPC. (Note that DCE RPC uses the DCE Security and Directory Services.) This is described in “Chapter 28. DCE remote procedure calls” on page 325.

The main advantage of a DCE remote procedure call over a CICS DPL call is that you can call CICS programs from non-CICS environments.

Chapter 28. DCE remote procedure calls

This chapter gives an overview of how CICS cooperates with the OS/390 Unix Systems Services DCE Base Services MVS/ESA and OS/390 Unix Systems Services DCE Application Support MVS/ESA CICS Feature products to enable a CICS program to act as a DCE server. For more detailed information, you should refer to the books listed in "Where to find more information" on page 327.

The chapter contains the following sections:

- "Overview"
- "What CICS server programs can do" on page 326
- "What you need for DCE RPC to a CICS server" on page 326
- "DCE terminology" on page 327
- "Where to find more information" on page 327.

Overview

The OS/390 Unix Systems Services DCE Application Support MVS/ESA CICS Feature [5] enables a DCE client application anywhere in the DCE environment to access the resources of a CICS system. The client program uses the simple DCE Remote Procedure Call (RPC), or the DCE Transactional Remote Procedure Call (TRPC), mechanism to call a CICS application program.

The client program does not need to know where the required CICS application is located. Functions of the Application Support server and the OS/390 Unix Systems Services DCE Base Services MVS/ESA product (OS/390 Unix DCE Base MVS/ESA) provide the location information. When the client and server are on different systems, the differences are transparent to the application programmer.

The Application Support server supports client programs written in C, and CICS application programs written in COBOL. The Application Support server automatically handles the conversions of the COBOL and C data types. Components of OS/390 Unix DCE Base MVS/ESA handle conversions of EBCDIC and ASCII data types, if needed.

Thus the Application Support server provides the powerful CICS application environments on the host, and the familiar (to the client workstation programmer) C language and RPC mechanism on the client.

The Application Support server:

- Coexists with all other ways of accessing CICS.
- Allows access to existing CICS applications and data.
- Allows new CICS applications to be developed as servers in the OS/390 UNIX System Services DCE Executive MVS/ESA environment.
- Allows access to all files and databases available to CICS, including DB2 databases.

[5] For clarity, in the rest of this book this product is called the "Application Support server".

© Copyright IBM Corp. 1994, 1998

- Gives the host programmer continued access to all the facilities and tools in the CICS environments. This includes requests to run other programs on the same subsystem or different subsystems using the existing CICS mechanisms.
- Allows a client program to access CICS and does not require the client machine to have CICS transaction processing function installed.

What CICS server programs can do

The Application Support server and CICS are connected by the external CICS interface (EXCI), which uses CICS interregion connection (IRC) facilities. The Application Support server maps the DCE RPC parameters into a CICS communications area, and then uses EXCI to invoke the required CICS program, as if it had been called by an EXEC CICS LINK command.

TRPCs from a client program within the scope of a single client transaction are handled by a single CICS task. A syncpoint issued by the client application commits or backsout all resources owned by the CICS server task as well as any owned by the client application.

Each RPC from a client program is handled as a CICS task, with an implied syncpoint at the end of the task. Note that this syncpoint only commits resources owned by the CICS server task. It does not commit any resources owned by the client program.

Your server program can access any file or database available in the CICS environment. It can use CICS distributed facilities to access data and programs that are managed by other CICS, IMS, or other APPC-connected systems.

You can use DCE RPCor TRPC to access CICS programs for one or more of the following reasons:

- To access CICS data from a platform which does not support CICS, but which does support DCE.
- To access CICS data from workstation programs which do not run in a CICS environment. You may want to do this even if the workstation platform supports CICS.
- To use the DCE Security Service, with its high level of protection against interception of network traffic.
- To use the DCE Directory Service, to provide client independence of the location of the required server program.

For details of how to write CICS server programs, see “Chapter 30. Application programming for DCE remote procedure calls” on page 331.

What you need for DCE RPC to a CICS server

This support requires the following products:

- Connectivity through TCP/IP protocols to the client workstation, and to the DCE directory and security servers. This normally means a TCP/IP network, though for some partner platforms it may be possible to use an SNA network with ANYNET support at both ends to transport TCP/IP protocols using SNA transmission protocols.

- IBM TCP/IP for MVS, Version 3 Release 1 or later, to present a TCP/IP interface to the DCE software, even if you are using an SNA network and ANYNET software.
- OS/390 Unix Systems Services Distributed Computing Environment Base Services MVS/ESA, Version 5 Release 1 or later.
- OS/390 Unix Systems Services Distributed Computing Environment Application Support MVS/ESA CICS Feature, Version 1 Release 1 or later.

DCE terminology

The CICS server programs are called *operations*. Each RPC requests the execution of one operation. The declarations for each operation, including the specifications for the input and output parameters, are contained in an *interface definition*. You define one or more related operations in an interface, using the Interface Definition Language (IDL).

IDL defines the server functions that a client can call. IDL is a declarative language with syntax similar to the C language. The Application Support server contains IDL extensions that enable a programmer to use COBOL syntax to define the parameters for the CICS application programs. The programmer coding the IDL declarations may be a COBOL or a C programmer.

Note: There are restrictions on the COBOL and C data structures that can be defined using the IDL. These are described in the *OS/390 DCE Application Support Programming Guide*.

Where to find more information

Refer to the following books for more information about the OS/390 Unix Systems Services DCE Base Services MVS/ESA product:

- *DCE: Understanding the Concepts*, GC09-1478
- *Introducing the OpenEdition Distributed Computing Environment*, GC09-1482
- *OpenEdition Distributed Computing Environment: Application Development Guide*, SC09-1484, for guidance information about developing the client code and using the OpenEdition DCE MVS/ESA base services.
- *OpenEdition Distributed Computing Environment: Application Development Reference*, SC09-1487, for reference information about application programming interfaces (APIs).

Refer to the following books for more information about the OS/390 Unix Systems Services DCE Application Support MVS/ESA CICS Feature:

- *OS/390 DCE Application Support Programming Guide*, SC24-5833, for information about how to install CICS remote procedure call server programs.
- *OpenEdition Distributed Computing Environment: Application Support Configuration and Administration Guide*, SC09-1659, for information about the administration tasks that complement the programming tasks.

Chapter 29. Defining CICS programs as DCE servers

This chapter gives an overview of how to define CICS programs as servers to DCE remote procedure calls (RPCs). For more detailed information, see the *OS/390 DCE Application Support Programming Guide* and the *OpenEdition Distributed Computing Environment: Application Support Configuration and Administration Guide*.

Interface definition

When you write your CICS server program and your DCE client program you must:

1. Use the GENUUID command of the DCE MVS/ESA Application Support server to obtain a skeleton interface definition. (An interface defines one or more related *operations*. Each operation relates to a server program.) The skeleton includes a Universal Unique Identifier (UUID) that uniquely identifies the interface.
2. Use the DCE Interface Definition Language (IDL) to identify each operation in the interface and define its input and output parameters.
3. Use the IDL compiler to generate data structure definitions for the RPC parameters and execution stubs for both client and server programs.

   The client stub packages (*marshalls*) the RPC parameters for transmission over the network to the server, and unpackages (*unmarshalls*) the parameters received from the server.

   The server stubs contain function that converts host COBOL data types to C data types and vice versa. They also package and unpackage RPC parameters, and convert data between EBCDIC and ASCII representations.
4. Link edit and load the server stubs into the server stub library.
5. Link edit the client stub with the client program.

You must also define your server programs to CICS using RDO, as described in *CICS Resource Definition Guide*. The definitions can be statically defined and installed, or autoinstalled when the programs are first called.

Interface installation

When you have completed your CICS server program you need to advertise its availability to potential clients. You do this by using the Application Support server administration facilities to install the interface. This exports details of the interface to the DCE distributed directory. Client programs can then use DCE facilities to locate servers which support required interfaces.

© Copyright IBM Corp. 1994, 1998

Chapter 30. Application programming for DCE remote procedure calls

Support for DCE remote procedure calls (RPCs) enables a non-CICS client program running in an Open Systems Distributed Computing Environment (DCE) to link to a server program running in a CICS system. For an introduction to DCE RPCs, see “Chapter 28. DCE remote procedure calls” on page 325.

The DCE client program

For information about coding DCE client programs, see the *OpenEdition Distributed Computing Environment: Application Development Guide* and the *OpenEdition Distributed Computing Environment: Application Development Reference* manual.

The CICS server program

Note: This is an overview only of how to write CICS programs to act as servers to DCE remote procedure calls. For further related information, see “Chapter 29. Defining CICS programs as DCE servers” on page 329. For definitive information, see the *OS/390 DCE Application Support Programming Guide*.

CICS server programs must:

- Use a communications area to pass input and output parameters.
- Pass input and output parameters by value (not by pointer).
- Contain only data-handling logic. Existing applications that have their data-handling and terminal input/output logic in separate programs can be used without modification.
- Ideally, be written in COBOL II or LE/COBOL. This is because the Application Support server compiler produces only COBOL data structure definitions for your CICS communications area, to match the RPC parameters. You can, however, write your server application in another programming language, by manually defining a communications area data structure that exactly overlays that produced in COBOL by DCE.

CICS server programs can:

- Use the same subset of EXEC CICS commands as CICS DPL server programs. (The restricted commands are listed in the programming information in the *CICS Application Programming Reference* manual.)
- Also be servers to distributed program link requests.
- Use CICS intercommunication facilities to access data and programs owned by other APPC-connected systems. For example, they can use the Front End Programming Interface (FEPI) to emulate a 3270 terminal, and thereby act as a front end for other unchanged CICS or IMS™ applications.
- Communicate with applications in remote CICS systems, using function shipping, DPL, or distributed transaction processing.

The Application Support server does not support CICS application programs that:

© Copyright IBM Corp. 1994, 1998

- Contain terminal input/output logic to the principal facility. (Note that you can use APPC terminal control commands to do distributed transaction processing to a remote back-end system.)
- Use basic mapping support (BMS).

These restrictions are the same as those for CICS distributed program link servers. Thus, you may be able to use server programs written for CICS-to-CICS DPL as servers to DCE clients.

As described in “Interface definition” on page 329, you must use the DCE MVS/ESA Application Support server compiler to generate a data structure definition for the RPC parameters passed to your server program, and an execution stub for the server. You must link edit and load the stub into the server stub library.

Part 6. Appendixes

© Copyright IBM Corp. 1994, 1998

Appendix. Routing program-link requests

Important

For detailed information about routing program-link requests, see the *CICS Intercommunication Guide*. This appendix is an overview of how program-link requests received from outside CICS can be routed to other regions.

"Traditional" CICS-to-CICS distributed program link (DPL) calls, instigated by EXEC CICS LINK PROGRAM commands, can be "daisy-chained" from region to region, simply by defining the program as remote in each region except the last (server) region, where it is to execute. The same applies to program-link requests received from *outside CICS*. For example, all of the following types of program-link request can be routed:

- Calls received from:
  - CICS Web support
  - The CICS Transaction Gateway
- Calls from external CICS interface (EXCI) client programs
- External Call Interface (ECI) calls from any of the CICS Client workstation products
- Distributed Computing Environment (DCE) remote procedure calls (RPCs)
- ONC/RPC calls.

Static routing

A program-link request received from outside CICS can be statically routed to a remote CICS region by specifying the name of the remote region on the REMOTESYSTEM option of the installed program definition.

Dynamic routing

A program-link request received from outside CICS can be dynamically routed by:

- Defining the program to CICS as DYNAMIC(YES)
- Coding your dynamic routing program to route the request.

© Copyright IBM Corp. 1994, 1998

Index

© Copyright IBM Corp. 1994, 1998

Sending your comments to IBM

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, to this address:

  Information Development Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  United Kingdom

- 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 topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

© Copyright IBM Corp. 1994, 1998

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1944-01

Spine information:

IBM CICS TS for OS/390 CICS External Interfaces Guide