IBM

Customer
Information
Control
System
CICS/MVS

Licensed Program
Version 2.1.2

Program Number
5665-403

Intercommunication Guide

SC33-0519-02

Third Edition (March 1991)

This edition applies to Version 2 Release 1 Modification 2 of the IBM licensed program Customer Information Control System/Multiple Virtual Storage (CICS/MVS), program number 5665-403, 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 *Intercommunication Guide* for CICS/MVS 2.1, SC33-0519. Changes from that edition are marked by vertical lines to the left of the changes.

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

A form for reader's comments appears at the back of this publication. If the form has been removed, address your comments to:

International Business Machines Corporation, Department 6R1H,
180 Kost Road, Mechanicsburg, PA 17055, U.S.A.

or to:

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

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

Special notices

This book is intended to help you to understand how to get CICS systems to communicate with other systems. It contains guidance about intercommunication.

Preface

What this book is about

This book is about:

- Multiregion operation: communication between CICS/MVS* systems running in different address spaces of the same processor.
- Intersystem communication: communication between a CICS/MVS system and other systems or terminals in a data communication network that support the logical unit type 6.1 or logical unit type 6.2 protocols of IBM* systems network architecture (SNA).

  Logical unit type 6.2 protocols are also known as advanced program-to-program communication (APPC).

Who should read this book

This book is for anyone who is involved in the planning and implementation of CICS intersystem communication (ISC) or multiregion operation (MRO).

What you need to know to understand this book

It is assumed throughout this book that you have experience with single CICS systems. The information it contains applies specifically to multiple-system environments, and the concepts and facilities of single CICS systems are, in general, taken for granted.

How to use this book

Initially, you should read part 1 of this book to familiarize yourself with the concepts of CICS multiregion operation and intersystem communication.

Thereafter, you can use the appropriate parts of the book as guidance and reference material for your particular task. The structure of the book is shown overleaf.

* IBM Trademark. For a list of trademarks, see page iii.

Book structure

CICS/MVS 2.1.2 library

General

CICS Library Guide GC33-0356-04
Master Index SC33-0513-01
User's Handbook SX33-6061-01
Messages and Codes SC33-0514-02

Evaluation and planning

Brochure GC33-0503-00
CICS General Information GC33-0155-01
Facilities and Planning Guide SC33-0504-01
Release Guide GC33-0505-03
Data Tables General Information SC33-0684

Administration

Installation Guide SC33-0506-01
Customization Guide SC33-0507-02
Resource Definition (Online) SC33-0508-01
Resource Definition (Macro) SC33-0509-02
Operations Guide SC33-0510-01
CICS-Supplied Transactions SC33-0511-01

Special topics

Intercommunication Guide SC33-0519-02
Recovery and Restart Guide SC33-0520-01
Performance Guide SC33-0521-01
XRF Guide SC33-0522-02
CICS Communicating with CICS OS/2 SC33-0736-1
Data Tables Guide SC33-0632-01

Service

Problem Determination Guide SC33-0516-01
Diagnosis Handbook LX33-6062-01
Diagnosis Reference LY33-6077-00
Data Areas LY33-6078-00

Programming

CICS Application Programming Primer SC33-0674-00
Application Programmer's Reference SC33-0512-01

Version 1 books

CICS/VS Application Programmer's Reference Manual (Macro Level) (SC33-0079)

CICS/OS/VS IBM 3270 Data Stream Device Guide (SC33-0232)

CICS/OS/VS IBM 4700/3600/3630 Guide (SC33-0233)

CICS/OS/VS IBM 3650/3680 Guide (SC33-0234)

CICS/OS/VS IBM 3767/3770/6670 Guide (SC33-0235)

CICS/OS/VS IBM 3790/3730/8100 Guide (SC33-0236)

Books from related libraries

Systems network architecture (SNA)

- *Concepts and Products*, GC30-3072
- *Technical Overview*, GC30-3073
- *Sessions Between Logical Units*, GC20-1868
- *Formats*, GA27-3136
- *Network Product Formats*, LY43-0081
- *Format and Protocol Reference Manual: Architecture Logic*, SC30-3112
- *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *Format and Protocol Reference Manual: Distribution Services*, SC30-3098
- *Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084.
- *LU6.2 Reference: Peer Protocols*, SC30-6808
- *Transaction Processing Concepts and Facilities*, GC33-0754

VTAM

- *VTAM Programming for LU6.2*, SC31-6410
- *VTAM Resource Definition Reference*, SC31-6412

IBM system center publications

- *CICS Advanced Program-to-Program Communication Support*, G320-0579
- *Advanced Communications Function Products Installation Guide*, GG24-1557.
- *SNA APPC in a Peer CICS/VS Environment*, GG24-1656
- *An Introduction to Advanced Program-to-Program Communication (APPC)*, GG24-1584
- *IBM System/36 APPC Implementation Guide*, GG24-1693.

IMS

- *CICS/VS to IMS/VS Intersystem Communication Primer*, SH19-6247 through SH19-6254
- *IMS/VS Version 2 Programming Guide for Remote SNA Systems*, SC26-4186.

CICS/VM

- *CICS/VM General Information*, GC33-0571
- *CICS/VM System Support and Administration*, SC33-0573.

CICS OS/2

- *CICS OS/2 System and Application Guide* SC33-0616.

Contents

Summary of changes

This edition is based on the *CICS/MVS Intercommunication Guide* (SC33-0519-1), and incorporates updates and revisions as well as enhancements introduced by CICS/MVS 2.1.1 and CICS/MVS 2.1.2. These enhancements are described in the *CICS/MVS Release Guide*.

The opportunity has also been taken to correct errors and incorporate readers' comments.

All changes that are new in this edition, other than editorial changes, are marked by revision bars in the left margin, like this paragraph.

The "Intercommunication and XRF" chapter has been added to Part 5, *Recovery and Restart*.

A new Part 6, called "Link services", has been created. This incorporates the *Security in the intercommunication environment* chapter and the expanded material headed *Master terminal operations for LUTYPE6.2 connections*, which originally formed part of chapter 3.4.

Questionnaire

CICS/MVS Version 2 Release 1 Modification 2 Intercommunication Guide **Publication No. SC33-0519-02**

To help us produce books that meet your needs, please fill in this questionnaire. A reader's comment form is also included at the back of this book should you want to make more detailed comments. Whichever form you use, your comments will be sent to the author's department for review and appropriate action.

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

1. Please rate the book on the points shown below

The book is:

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

And the book as a whole?

excellent	1	2	3	4	5	poor

2. Which topics does the book handle well? ____________________

3. And which does it handle badly? ____________________

4. How could the book be improved? ____________________

5. How often do you use this book? Less than once a month? ☐ Monthly? ☐ Weekly? ☐ Daily? ☐

6. What sort of work do you use CICS for? ____________________

7. How long have you been using CICS? ________ years/months

8. Have you any other comments to make? ____________________

Thank you for your time and effort. No postage stamp necessary if mailed in USA. (If you are outside the USA, please mail this form to your local IBM office or representative who will be happy to forward your comments or you may mail directly to either address in the Edition Notice on the back of the title page.) Be sure to print your name and address below if you would like a reply.

Name...Job Title ....

Company..Address..

....Zip..

Readers' Comments
SC33-0519-02

Cut o
Alon

Fold and Tape | **Please do not staple** | Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 6R1H
180 KOST ROAD
MECHANICSBURG PA 17055-0786

Fold and Tape | **Please do not staple** | Fold and Tape

Cut or
Along

Part 1. Concepts and facilities

This part of the manual describes the basic concepts of CICS intercommunication and the various facilities that are provided.

"Chapter 1.1. Introduction to CICS intercommunication" on page 3 defines CICS **intercommunication**, and introduces the two types of intercommunication: **multiregion operation** and **intersystem communication**. It then describes the intercommunication facilities that CICS provides. These are:

- CICS Function Request Shipping
- Asynchronous Processing
- CICS Transaction Routing
- Distributed Transaction Processing (DTP).

"Chapter 1.2. Multiregion operation" on page 9 through "Chapter 1.7. Distributed transaction processing" on page 55 then describe each of these topics in more detail, as follows:

✓

Chapter 1.1. Introduction to CICS intercommunication

In this book, it is assumed that you are familiar with the use of CICS as a stand-alone system, with associated data resources and a network of terminals. Instead, we are concerned with the role of CICS in a multiple-system environment, in which CICS can communicate with other systems that have similar communication facilities. We have called this sort of communication **CICS intercommunication**.

CICS intercommunication is, then, communication between a **local** CICS system and a **remote** system, which may or may not be another CICS system.

Intercommunication methods

There are two ways in which CICS can communicate with other systems: **intersystem communication** and **multiregion operation**.

Intersystem communication

For communication between systems that are in different hosts, you require an SNA access method, such as ACF/VTAM*, to provide the necessary communication protocols. Communication between systems via SNA is called **intersystem communication** (ISC).

Note: This form of communication can also be used between systems in the same host processor, via the application-to-application facilities of ACF/VTAM.

The SNA protocols that CICS uses for intersystem communication are Logical Unit Type 6.1 (otherwise known as LUTYPE 6.1) and Logical Unit Type 6.2 (otherwise known as LUTYPE 6.2 or Advanced Program-to-Program Communication (APPC)). Additional information on this topic is given in "Chapter 1.3. Intersystem communication" on page 13.

CICS/MVS Version 2 Release 1 Modification 2 can use ISC to communicate with the following systems:

- Any CICS/MVS Version 2 system
- CICS/ESA*
- CICS/OS/VS Version 1 Release 7
- CICS/DOS/VS Version 1 Release 7
- CICS/VSE* Version 2 Release 1
- CICS/VM*
- CICS OS/2*
- IMS/VS Version 2 Release 1
- IMS/VS Version 2 Release 2
- IMS/ESA* Version 3 Release 1

* IBM Trademark. For a list of trademarks, see page iii.

- Any system that supports LUTYPE6.2 (APPC) protocols. This includes LUTYPE6.2 single session terminals, such as:
  - Displaywriter*
  - Scanmaster
  - System/36
  - System/38.

Multiregion operation

For CICS-to-CICS communication in the same MVS image, CICS provides an **interregion communication** facility that is independent of the SNA access method. This form of communication is called **multiregion operation** (MRO).

| CICS/MVS Version 2 Release 1 Modification 2 can communicate via MRO with other CICS/MVS Version 2 systems, with CICS/ESA systems, and with CICS/OS/VS release 1.7.

Intercommunication facilities

In the multiple-system environment, each participating system can have its own, local, terminals and databases, and can run its local application programs independently of other systems in the network. In addition, it can establish links to other systems, and thereby gain access to remote resources. This mechanism enables resources to be distributed among and shared by the participating systems.

CICS intercommunication provides four basic types of facility:

- CICS Function Request Shipping
- Asynchronous Processing
- CICS Transaction Routing
- Distributed Transaction Processing (DTP).

These facilities are not universally available for all forms of intercommunication. The circumstances under which they can be used are shown in Table 1.

Table 1. Availability of intercommunication facilities

Facility	**LUTYPE6.2 (CICS)**	**LUTYPE6.2 (non-CICS)**	**LUTYPE6.1 (CICS)**	**LUTYPE6.1 (IMS)**	**MRO**
Function Request Shipping	Yes	No	Yes	No	Yes
Asynchronous Processing	Yes	No	Yes	Yes	Yes
Transaction Routing	Yes	No	No	No	Yes
Distributed Transaction Processing	Yes	Yes	Yes	Yes	Yes

CICS function request shipping

This facility enables an application program to access a resource owned by another CICS system. Both read and write access are permitted, and facilities for exclusive control and recovery/restart are provided.

The remote resource can be:

- A file or a DL/I database
- A transient data queue
- A temporary storage queue.

Remote transactions are also considered to be resources, and can be initiated by requests from a local application program. This form of request shipping is called **asynchronous processing** in this book.

Application programs that access remote resources can be designed and coded as if the resources were owned by the system in which the transaction is to run. During execution, CICS ships the request to the appropriate system.

Asynchronous processing

This facility enables a CICS transaction to initiate a transaction in a remote system and to pass data to it. The data can include the name of a local transaction that is to be initiated by the remote system to receive the reply. The reply is not necessarily returned to the **task** that initiated the remote transaction, and no direct correlation between requests and replies (other than that provided by user-defined fields in the data) is possible; therefore the process is **asynchronous**.

From the CICS point of view, asynchronous processing is a form of function shipping, in which interval control START requests are shipped to and received from remote systems. Functionally, it is similar to distributed transaction processing (see below) in that it allows processing to be distributed between systems.

CICS transaction routing

This facility enables a terminal that is owned by one CICS system to run a transaction that is owned by another CICS system. Similarly, a transaction that is started by automatic transaction initiation (ATI) can acquire a terminal that is owned by another CICS system.

Transaction routing is available between CICS systems connected either by interregion links (MRO) or by LUTYPE6.2 (APPC) links.

Distributed transaction processing (DTP)

This facility enables a CICS transaction to communicate with a transaction running in another system. The transactions are designed and coded explicitly to communicate with each other, and thereby to utilize the intersystem link with maximum efficiency.

The communication in distributed transaction processing is, from the CICS point of view, **synchronous**, which means that it occurs during a single invocation of the CICS transaction and that requests and replies between two transactions can be directly correlated. This contrasts with the asynchronous processing described previously.

Applications of CICS intercommunication

The CICS intercommunication facilities enable you to implement many different types of distributed data processing. This section describes a few typical applications. The list is by no means exhaustive, and further examples are presented in the other chapters of this part.

Multiregion operation enables two CICS regions to share selected system resources, and to present a "single-system" view to terminal operators. At the same time, each region can run independently of the other, and can be protected against other-region errors. Various possible applications of MRO are described in "Chapter 1.2. Multiregion operation" on page 9.

| CICS intersystem communication, together with an SNA access method
| (ACF/VTAM) and a network control program (ACF/NCP/VS), enables resources to
| be distributed among and shared by different systems, which can be in the same or different physical locations.

Figure 1 on page 7 shows some typical possibilities.

Connecting regional centers

Many users have computer operations set up in each of the major geographical areas in which they operate. Each system has a database organized towards the activities of that area, and its own network of terminals able to inquire or update the regional database. When requests from one region require data from another, without intersystem communication, manual procedures have to be used to handle such requests. The intersystem communication facilities allow these "out-of-town" requests to be automatically handled by providing file access to the database of the appropriate region.

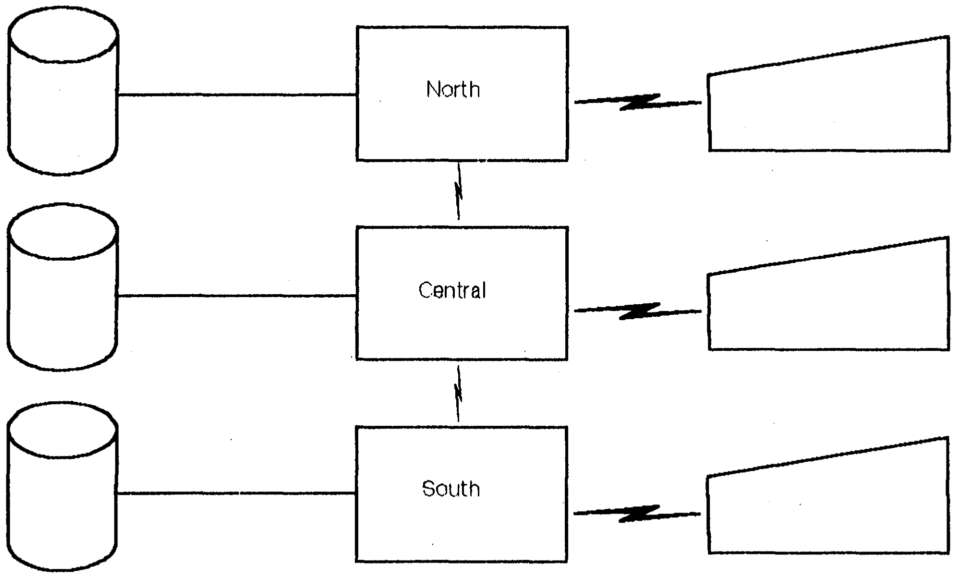
Using CICS function shipping, application programs can be written to be independent of the actual location of the data, and able to run in any of the regional centers. An example of this type of application is the validation of credit against customer accounts.

Connecting divisions within an organization

Some users are organized divisionally, with separate systems, terminals, and databases for each division: for example, Engineering, Production, and Warehouse divisions. Connecting these divisions to each other and to the headquarters location improves access to programs and data, and thus can improve the coordination of the enterprise.

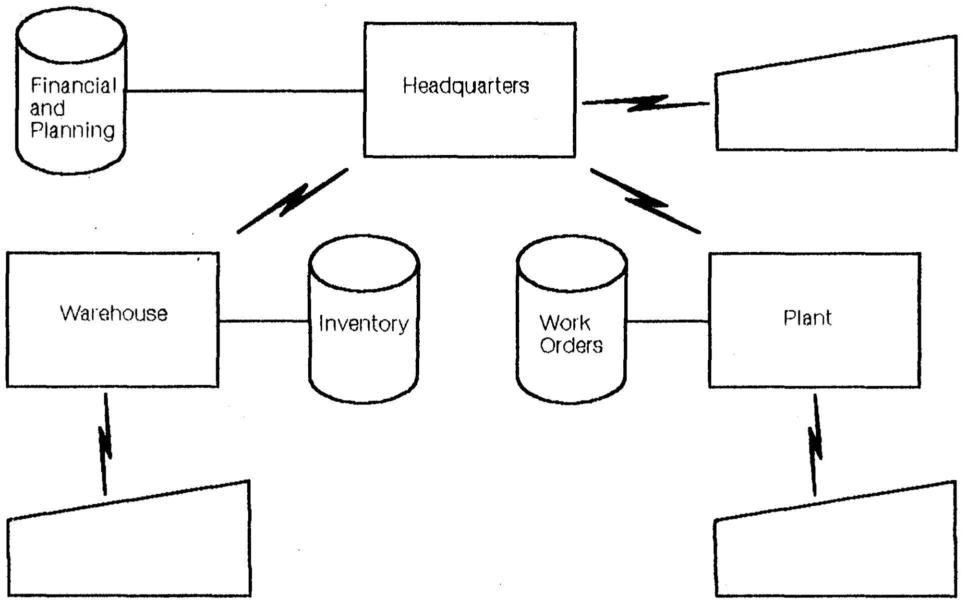
The applications and data can be hierarchically organized, with summary and central data at the headquarters site and detail data at plant sites. Alternatively, the applications and data can be distributed across the divisional locations, with planning and financial data and applications at the headquarters site, manufacturing data and applications at the plant site, and inventory data and applications at the distribution site. In either case applications at any site can access data from any other site, as necessary, or request applications to be run at a remote site (containing the appropriate data) with the replies routed back to the requesting site when ready.

Connecting regional centers



- Data base partitioned by area
- Same applications run in each center
- All terminal users can access applications or data in all systems
- Terminal operator and applications unaware of location of data
- Out-of-town requests routed to the appropriate system

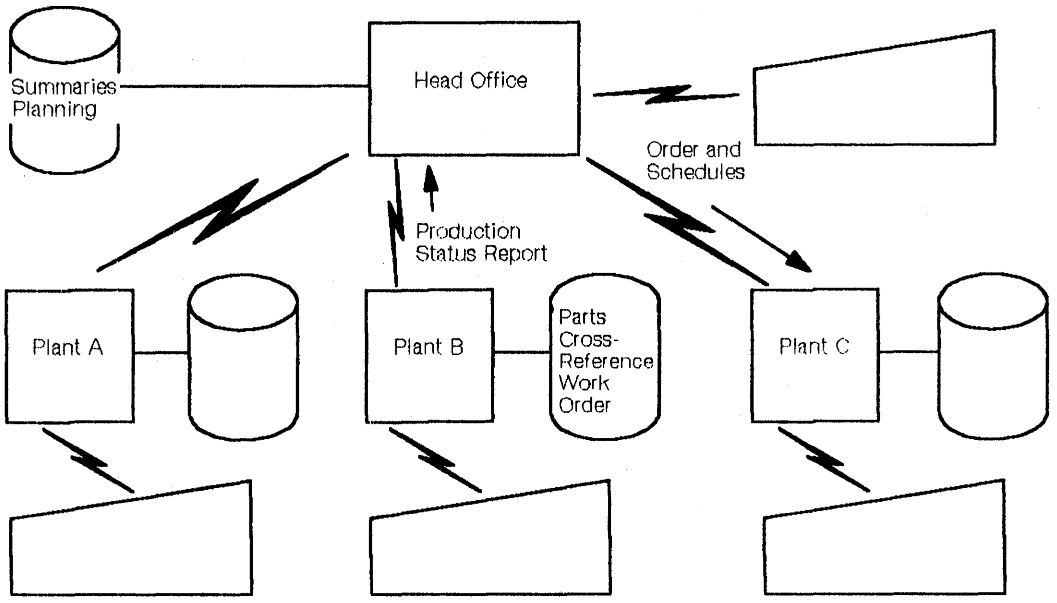
Connecting divisions: distributed applications and data



- Data base partitioned by function
- Applications partitioned by function
- All terminal users and applications can access data in all systems
- Requests for nonlocal data routed to the appropriate system

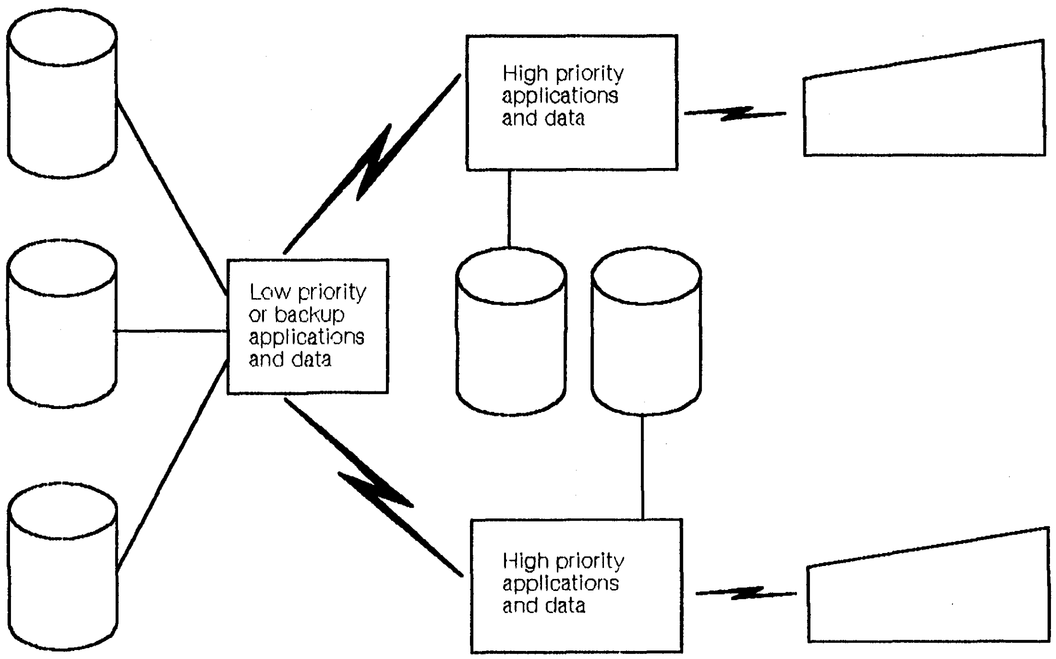
Figure 1 (Part 1 of 2). Examples of distributed resources

Hierarchical division of data base



- Summaries and central data at HQ, detail data at plant location
- Order processing at HQ: orders and schedules transmitted to plants of production status
- Plants and summaries of production status to HQ (for example, overnight)
- Access to data from HQ or Plant possible if required

Connecting division: hierarchical distribution of data and application


High priority applications and data
Low priority or backup applications and data
High priority applications and data

- Improved response through distributed processing

Figure 1 (Part 2 of 2). Examples of distributed resources

Chapter 1.2. Multiregion operation

CICS Multiregion Operation (MRO) enables CICS systems that are running in different address spaces of the same MVS image to communicate with each other. MRO does not support communication between a CICS system and a non-CICS system such as IMS.

ACF/VTAM and SNA networking facilities are not required for MRO. The data transfers between address spaces are made either by a CICS-supplied interregion program running in supervisor state, or by means of MVS cross-memory services.

The support within CICS that accomplishes region to region communication is called **Interregion Communication (IRC)**. IRC is implemented through support in CICS terminal control management modules and by use of a CICS-supplied interregion program (DFHIRP) loaded in the link pack area (LPA) of MVS. DFHIRP is invoked by a type 2 supervisory call (SVC).

MVS cross-memory services can be selected as an alternative to the CICS type 2 SVC mechanism. In this case, DFHIRP is used only to open and close the interregion links.

The CICS IRC support utilizes a Service Request Block (SRB) routine.

Note: IRC is also used by CICS/MVS for support of IMS DB batch regions accessing DL/I databases controlled by CICS through CICS's Shared Database facility and for support of the Data Dictionary program product.

The intercommunication facilities available via MRO are:

- Function request shipping
- Transaction routing
- Asynchronous processing
- Distributed transaction processing.

There are some restrictions for distributed transaction processing under MRO that do not apply under ISC.

Installation of CICS multiregion operation is described in "Chapter 2.1. Installation considerations for multiregion operation" on page 79.

Applications of multiregion operation

This section describes a number of typical applications of multiregion operation.

Extended recovery facility (XRF)

The *CICS/MVS XRF Guide* provides examples of the various types of XRF configuration that are possible using MRO.

Program development

The testing of newly-written programs can be isolated from production work by running a separate CICS region for testing. This enables the reliability and availability of the production system to be maintained during the development of new applications, because the production system remains up even if the test system terminates abnormally.

By using function request shipping, the test transactions can access resources of the production system, such as files or transient data queues. By using transaction routing, terminals connected to the production system can be used to run test transactions.

The test system can be brought up and taken down as required, without interrupting production work. During the cutover into production of the new programs, terminal operators can run transactions in the test system from their regular production terminals, and the new programs can access the full resources of the production system.

Time-sharing

If one CICS system is used for compute-bound work, such as APL or ICCF, as well as regular DB/DC work, the response time for the DB/DC user can be unduly long. It can be improved by running the compute-bound applications in a lower priority address space and the DB/DC applications in another. Transaction routing allows any terminal to access either CICS system without the operator being aware that there are two different systems.

Reliable database access

In your installation, it may be possible to divide your applications into two sets; one containing applications that are known to be completely reliable, and one containing applications that can possibly bring the CICS system down.

With MRO, you can define two CICS regions, one of which owns the unreliable applications, and the other the reliable applications and also the database. The fewer applications that run in the database-owning region, the more reliable this region will be. On the other hand, the cross-region traffic will be greater, so performance can be degraded. You must balance performance against reliability.

You can take this application of MRO to its limit by having no user applications at all in the database-owning region. The online performance degradation may be a worthwhile trade-off against the elapsed time necessary to restart a CICS region that owns a very large database.

Departmental separation

MRO allows various departments of an organization to have their own CICS systems. Each can bring up and take down its own system as it requires. At the same time, each can have access to other departments' data, with access being controlled by the system programmer. A department can run a transaction on another department's system, again subject to the control of the system programmer. Terminals need not be allocated to departments, since, with transaction routing, any terminal could run a transaction on any system.

Multiprocessor performance

With MRO, using several CICS systems, the user can take advantage of a multiprocessor, and allow any terminal to access the transactions and data resources of any of the systems. Transaction routing presents the terminal operator with a single system image; the operator need not be aware that there is more than one CICS system.

The system programmer can assign transactions and data resources to any of the connected systems so as to balance the load and achieve optimum performance.

Virtual storage constraint relief

In some large CICS systems, the amount of virtual storage available can become a limiting factor. In these cases, it is often possible to relieve the virtual storage problem by splitting the system into two or more separate systems with shared resources. All the facilities of MRO can be used to help maintain a single-system image for end users.

Note: If you are using DL/I databases, and wish to split your system to obtain virtual storage constraint relief, you should consider using IMS data sharing, rather than CICS function shipping, to share the databases between your CICS address spaces.

Conversion from single region system

Existing single-region CICS systems can generally be converted to multiregion CICS systems with little or no reprogramming.

CICS function request shipping will allow an existing command-level application to continue accessing existing data resources after either the application or the resource has been transferred to another CICS region. Applications that use function request shipping must conform to the rules given in "Chapter 4.2. Application programming for CICS function shipping" on page 163, which may necessitate program modification in some cases.

CICS transaction routing will allow an existing command-level or macro-level application to be run from an existing terminal after either the application or the terminal has been transferred to another CICS region. The restrictions that apply in this case are given in "Chapter 4.4. Application programming for CICS transaction routing" on page 169.

In all cases it will be necessary to define an MRO link between the two regions and to provide local and remote definitions of the shared resources. These operations are described in "Part 3. Resource definition and master terminal operation" on page 89.

Batching of work in an MRO region

In some MRO configurations, a CICS region may frequently enter the wait state because it has no incoming MRO requests to handle and no other work to do. The next incoming MRO request will then carry the overhead involved in getting CICS out of the wait state, and possibly of returning it to the wait state when the request has been dealt with.

This overhead can often be reduced by delaying the posting of a region until several incoming MRO requests are outstanding. The region can then be posted and can handle all the outstanding requests before it returns to the wait state.

You can specify the number of MRO requests that are to be batched in this manner by means of the MROBTCH operand of the system initialization table (SIT). The number can have a value in the range from 1 to 255. The default is 1, meaning that no batching is to occur.

The maximum time that a region is allowed to remain in the wait state is specified in the ICV operand of the SIT. If you use MRO batching, you should choose an ICV value to ensure that MRO requests are not unduly delayed in a lightly-loaded system.

Chapter 1.3. Intersystem communication

The data formats and communication protocols required for communication between systems in a multiple-system environment are embodied in IBM Systems Network Architecture (SNA). The CICS implementation of intersystem communication is effectively an implementation of this architecture.

It is assumed that you are familiar with the general concepts and terminology of SNA. Some books on this subject are listed under "Books from related libraries" on page viii.

Connections between systems

This section presents a brief overview of the ways in which systems can be connected together for the purposes of intersystem communication. There are three basic forms to be considered:

1. ISC within a single processor
2. ISC between physically adjacent processors
3. ISC between physically remote processors.

A possible configuration is shown in Figure 2.

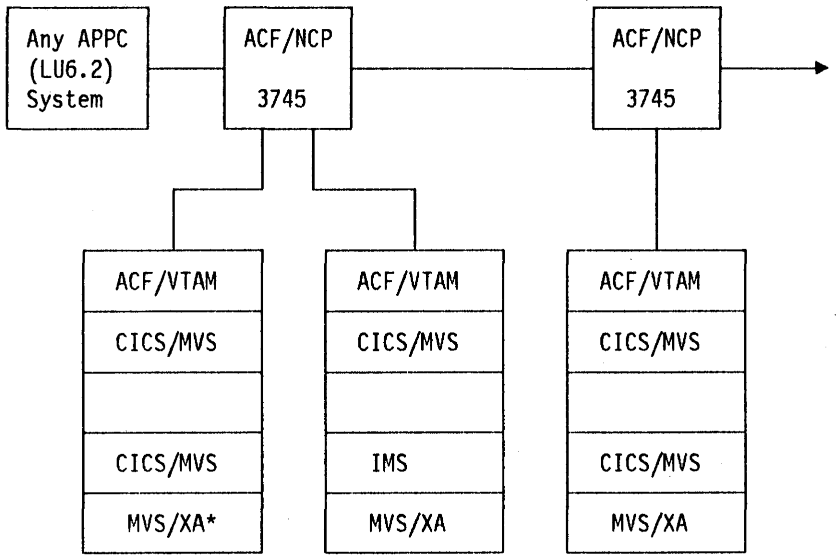


Figure 2. A possible configuration

★ IBM Trademark. For a list of trademarks, see page iii.

Single processors

ISC within a single processor (intrahost ISC) is possible through the application-to-application facilities of ACF/VTAM or ACF/TCAM.

In an MVS system, you can use intrahost ISC for communication between two or more CICS/MVS systems or between, for example, a CICS/MVS system and an IMS system.

From the CICS point of view, intrahost ISC is indistinguishable from ISC between systems in different VTAM* domains.

Physically adjacent processors

| An IBM 3725 or 3745 communications controller can be configured with a multiple-channel adapter which enables you to connect two VTAM or TCAM domains via a single ACF/NCP/VS. This configuration may be useful for communication between:

1. A production machine and a local but separate test machine
2. Two production machines with differing characteristics or requirements.

Direct channel-to-channel communication is available between systems that have ACF/VTAM Version 2 Release 1 (or a later release) installed.

Remote processors

This is the most typical configuration for intersystem communication. Each participating system can be appropriately configured for its particular location, using MVS or Virtual Storage Extended/Advanced Functions (VSE/AF), CICS or IMS, and one of the ACF access methods such as ACF/VTAM.

Intersystem sessions

CICS uses ACF/VTAM to establish, or **bind**, logical-unit-to-logical-unit (LU-LU) sessions with remote systems. Being a logical connection, an LU-LU session is independent of the actual physical route between the two systems, and a single intersystem link can carry multiple independent sessions. Such sessions are called **parallel** sessions.

The formats and protocols used for intersystem sessions are those of Logical Unit (LU) Type 6, and the sessions are known as LUTYPE6 sessions. Each session partner is also called an LU Type 6. LUTYPE6 links support parallel sessions and negotiable binds, and use specific types of function management headers (FMH).

CICS supports two types of LUTYPE6 session, both of which are defined by IBM Systems Network Architecture:

- LUTYPE6.1 sessions
- LUTYPE6.2 sessions.

* IBM Trademark. For a list of trademarks, see page iii.

The general term chosen for the LUTYPE6.2 protocol is Advanced Program-to-Program Communication (APPC).

LUTYPE6.1

LUTYPE6.1 is the term used to refer to the logical unit that was formerly called LUTYPE6. The ".1" is used to distinguish it from LUTYPE6.2.

The keyword LUTYPE6 has been retained in some CICS system generation and resource definition macros to preserve compatibility with earlier releases. In contexts where it is necessary to distinguish between LUTYPE6.1 and LUTYPE6.2, the keyword LUTYPE6 refers to LUTYPE6.1. In other contexts, it refers generically to both LU types.

The characteristics of LUTYPE6 sessions, and the formats of the associated function management headers, are described in Systems Network Architecture publication *Sessions Between Logical Units*. Details of the CICS implementation of LUTYPE6 are given in the *CICS/MVS Diagnosis Reference* manual.

Currently, LUTYPE6.1 sessions are supported by CICS and by IMS, and can be used for CICS-to-CICS and CICS-to-IMS communication.

LUTYPE6.2 (APPC)

Like LUTYPE6.1 sessions, LUTYPE6.2 sessions can be used for data communication between transaction processing systems. However, LUTYPE6.2 provides an architecture within which not only host- or system-level products, but also device-level products, can communicate.

At the interhost communication level, it offers facilities over and above those provided by the LUTYPE6.1 architecture. At the same time, it provides defined subsets which enable device-level products (LUTYPE6.2 terminals) to communicate with host level products (and also with each other). LUTYPE6.2 therefore represents both an upwards and downwards extension of the LUTYPE6.1 facilities.

LUTYPE6.2 sessions can be used for CICS-to-CICS communication, and for communication between CICS and other LUTYPE6.2 systems or terminals.

The following paragraphs provide an overview of some of the principal characteristics of the LUTYPE6.2 architecture.

Data stream

The data stream employed for LUTYPE6.2 communication is the SNA **generalized data stream** (GDS). In GDS, data is preceded by a header field (LLID) that specifies the overall length of the data (LL) and an identification of the data type (ID). The data type for user application data is X'12FF'.

Application programming interface

LUTYPE6.2 is the first SNA LU type to have a defined application programming language in which conversations can be coded. Details of this SNA-defined language are given in the Systems Network Architecture publication *Transaction Programmer's Reference Manual for LU Type 6.2.*

As a CICS user, you do not need to use this language directly; CICS provides a command-level language to enable you to write application programs that hold LUTYPE6.2 conversations.

Two types of LUTYPE6.2 conversation are defined:

1. **Mapped conversations**

   In mapped conversations, the data passed to and received from the LUTYPE6.2 API is simply user data. The user has no knowledge of the GDS headers, and is not responsible for building or interpreting them.

2. **Unmapped conversations**

   In unmapped conversations (also known as **basic** conversations) the data passed to and received from the LUTYPE6.2 API contains GDS headers. The user is responsible for building and interpreting the LL and ID fields. Unmapped conversations are used principally for communication with device-level products that do not support mapped conversations, and which possibly do not have an application programming interface open to the user.

In CICS, unmapped conversations are written using EXEC CICS GDS commands. Details of these commands and the way in which they are used are given in "Chapter 4.6. CICS applications for logical unit type 6.2 unmapped conversations" on page 221.

Mapped conversations are written using normal EXEC CICS commands. Details of these commands and the way in which they are used are given in "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171.

The CICS commands provided for mapped and unmapped conversations are basically implementations of the LUTYPE6.2 verbs described in the architecture. The mappings between CICS commands and LUTYPE6.2 verbs are given in Appendix C, "CICS mapping to the LUTYPE6.2 architecture" on page 413.

Synchronization levels

LUTYPE6.2 provides for three different levels of synchronization:

Level 0 (none)
: This level is for use when communicating with systems or devices that do not support synchronization points, or when no synchronization is required.

Level 1 (confirm)
: This level allows conversing transactions to exchange private synchronization requests.

  The commands involved are SEND CONFIRM, which is used by one transaction to inform the other that it requires a response, and ISSUE

CONFIRMATION, which is sent in reply to SEND CONFIRM. (Either ISSUE ERROR or ISSUE ABEND can be used as a negative response to the SEND CONFIRM command.)

Apart from the transmission of these commands, no architected or CICS-provided function is involved.

Level 2 (syncpoint)
This level is the equivalent of full CICS syncpointing, including rollback.

The commands involved are usual CICS syncpointing commands SYNCPOINT and SYNCPOINT ROLLBACK. Level 1 synchronization requests can also be used.

For complex syncpointing situations, where many intersystem sessions are involved, the ISSUE PREPARE command can be used to prepare all the session partners for the syncpoint before syncpointing starts. This technique is described in "The ISSUE PREPARE command" on page 196.

The maximum synchronization level that can be used on an LUTYPE6.2 session is governed by the level supported by the more restrictive of the two logical units. With this constraint, the actual synchronization level that will be used can be specified by the transaction that initiates the conversation. If the second transaction does not expect to operate at the same level, the conversation cannot be held.

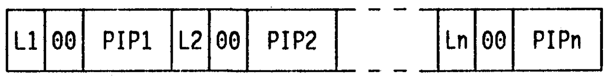
LU services manager

Multi-session LUTYPE6.2 connections use the **LU services manager**. This is the software component responsible for negotiating session binds, session activation and deactivation, resynchronization, and error handling. It requires two special sessions with the remote LU called the **SNASVCMG sessions**. When these are bound, the two sides of the LU-LU connection can communicate with each other, even if the connection is *out-of-service* for users.

A single-session LUTYPE6.2 connection has no SNASVCMG sessions. For this reason, its function is limited. It cannot, for example, support synchronization level 2.

Process initialization parameter data

When a transaction initiates a remote transaction on an LUTYPE6.2 session, it can pass data in the form of process initialization parameter (PIP) subfields. PIP subfields are formatted as follows:



where Ln is a halfword binary integer specifying the length of the subfield.

The length includes the length field and the two reserved bytes; that is, Ln = length of PIPn + 4.

PIP data is of concern only to the two transactions involved. It is not used for CICS-to-CICS communication, but it may be needed for communication with some other APPC systems. The meaning assigned to PIP data is defined by the APPC system concerned.

CICS provides facilities to enable a transaction to send PIP data to or receive PIP data from a remote transaction. These are described in Part 4.

Class of service

The CICS implementation of LUTYPE6.2 includes support for "class of service" selection.

Class of service (COS) is an ACF/VTAM facility that allows sessions between a pair of logical units to have different characteristics. This provides a user with the following function:

1. Alternate Routing — Virtual Routes for a given COS can be assigned to different physical paths (Explicit Routes).
2. Mixed Traffic — different kinds of traffic can be assigned to the same Virtual Route and, by selecting appropriate transmission priorities, undue session interference can be prevented.
3. Trunking — Explicit Routes can use parallel links between certain nodes.

In particular, sessions can take different Virtual Routes, and thus use different physical links; or the sessions can be of high or low priority to suit the traffic carried on them.

In CICS, LUTYPE6.2 sessions are specified in groups called modesets, each of which is assigned a modename. The modename must be the name of a VTAM LOGMODE entry (also called a "modegroup"), which can specify the class of service required for the session group. (See "ACF/VTAM LOGMODE table entries for CICS" on page 83.)

Limited resources

For efficient use of some network resources (for example, switched lines), SNA allows for such resources to be defined in the network as **limited resources**. Whenever a session is bound, VTAM indicates to CICS whether the bind is over a limited resource. When a task using a session across a limited resource frees the session, CICS unbinds that session if no other task wants to use it.

Both single and parallel sessions may use limited resources. For a parallel-session connection, CICS does not unbind LU service-manager sessions until all modegroups in the connection have performed initial CNOS exchange.

The use of limited resources is effective only if the systems at both ends of the connection support this function. This is because a CICS system without support for limited resource does not recognize the **available** connection state. That is the connection state in which there are no bound sessions and all are unbound because they were over limited resources.

Establishing intersystem sessions

Before traffic can flow on an intersystem session, the session must be established, or **bound**. CICS can be either the primary (BIND sender) or secondary (BIND receiver) in an intersystem session, and can be either the contention winner or the contention loser. The contention winner in an LU-LU session is the LU that is permitted to begin a bracket at any time. The contention loser is the LU that must use an SNA BID command (LUTYPE6.1) or LUSTATUS command (LUTYPE6.2) to request permission to begin a bracket.

The number of contention-winning and contention-losing sessions required on a link to a particular remote system can be specified by the system programmer.

For LUTYPE6.1 sessions, CICS always binds as a contention loser.

For LUTYPE6.2 links, the number of contention-winning sessions is specified when the link is defined (see "Defining logical unit type 6.2 links" on page 116). The contention-winning sessions are normally bound by CICS, but CICS will also accept bind requests from the remote system for these sessions.

Normally, the contention-losing sessions are bound by the remote system. However, CICS can also bind contention-losing sessions if the remote system is incapable of sending bind requests.

Single sessions to LUTYPE6.2 terminals are normally defined as contention winners, and are bound by CICS. In this case, CICS will accept a negotiated bind in which it is changed to the contention loser.

Session initiation can be performed in one of the following ways:

1. By CICS during CICS initialization for sessions for which AUTOCONNECT(YES) or AUTOCONNECT(ALL) has been specified (see "Chapter 3.1. Defining links to remote systems" on page 91).
2. By a request from the CICS master terminal operator.
3. By the remote system with which CICS will communicate.
4. By CICS when an application explicitly or implicitly requests the use of an intersystem session and the request can be satisfied only by binding a previously unbound session.

Chapter 1.4. CICS function shipping

CICS function shipping allows CICS command-level application programs to:

- Access files and DL/I databases managed by other CICS systems by shipping requests for file control or DL/I functions.
- Transfer data to or from transient data and temporary storage queues in other CICS systems by shipping requests for transient data and temporary storage functions.
- Initiate transactions in other CICS systems, or other non-CICS systems that implement SNA LU Type 6 protocols, such as IMS, by shipping interval control START requests. This form of communication is described in "Chapter 1.5. Asynchronous processing" on page 33.

Applications can be written without regard for the location of the requested resources; they simply use file control commands, temporary storage commands, and so on, in the normal way. Entries in the CICS resource definition tables allow the system programmer to specify that the named resource is not on the local (or requesting) system but on a remote (or owning) system.

An illustration of a shipped file control request is given in Figure 3 on page 22. In this figure, a transaction running in CICSA issues a file control READ command against a file called NAMES. From the file control table, CICS discovers that this file is owned by a remote CICS system called CICSB. CICS turns the READ request into a suitable transmission format, and then ships it to CICSB for execution.

In CICSB, the request is passed to a special transaction known as the **mirror transaction**. The mirror transaction recreates the original request, issues it on CICSB, and passes the acquired data back to CICSA.

The CICS recovery and restart facilities allow resources in remote systems to be updated and attempt to ensure that when the requesting application program reaches a synchronization point, any mirror transactions that are updating protected resources also take a synchronization point, so that changes to protected resources in remote and local systems are consistent. The CICS master terminal operator is notified of any failures in this process, so that suitable corrective action can be taken. This can be a manual process, or be effected by user-written code.

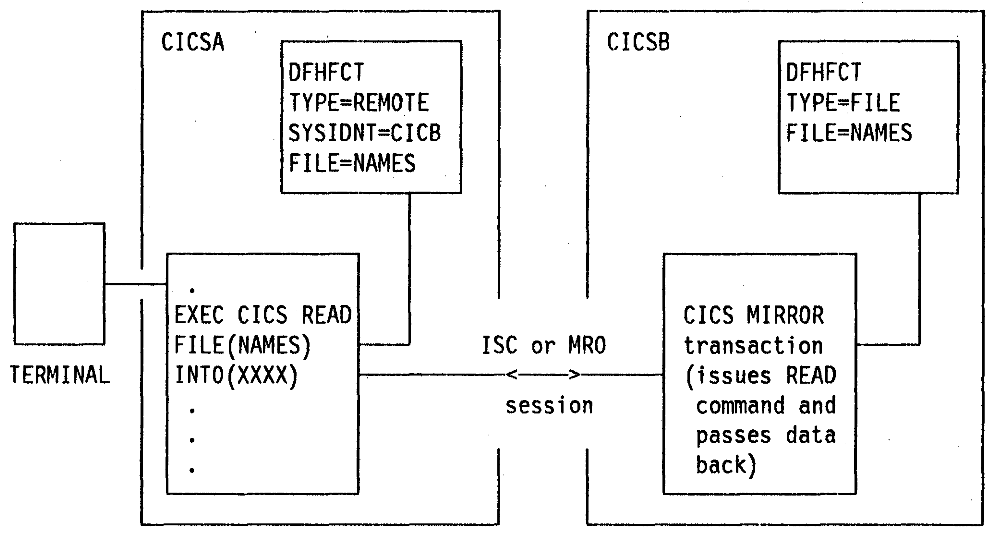


Figure 3. Function shipping

Design considerations

User application programs can run in a CICS intercommunication environment and make use of the intercommunication facilities without being aware of the location of the file or other resource being accessed. The location of the resource is defined by the system programmer in the appropriate CICS table (details are given in "Chapter 3.2. Defining remote resources" on page 133).

The resource definition can also specify the name of the resource as it is known on the remote system, if it is different from the name by which it is known locally. When the resource is requested by its local name, CICS substitutes the remote name before sending the request. This facility is useful when a particular resource exists with the same name on more than one system but contains data peculiar to the system on which it is located.

Application programs can also name remote systems explicitly on commands that can be function-shipped, by using the SYSID option. If this option is specified, the request is routed directly to the named system, and the resource definition tables on the local system are not used. The local system can be specified in the SYSID option, so that the decision whether to access a local resource or a remote one can be taken at execution time.

File control

Intercommunication allows access to BDAM or VSAM files located on a remote CICS system. OPEN and CLOSE are not supported. Both inquiry and update requests are allowed, and the files can be defined as protected in the system on which they reside. Updates to remote protected files will not be committed until the application program issues a syncpoint request or terminates successfully. Linked updates of local and remote files can be performed within the same logical unit of work, even if the remote files are located on more than one connected CICS system.

Caution is needed when designing systems where remote file requests using physical record identifier values are employed, such as BDAM, VSAM RBA, or files with keys not embedded in the record, because of the need to ensure that all application programs in remote systems have access to the correct values following addition of records or reorganization of these types of file.

You can improve data access time by using the optional CICS data tables feature. CICS supports both user-maintained and CICS-maintained remote data tables under MRO. However, CICS does not support creation of a local data table from a remote source data set. To simulate this, you will have to load local user-maintained data tables from a remote file by having an empty dummy VSAM data set as the source data set. You can then, for example, load the data table with its data by using a transaction that browses the remote file and writes the records to the local table.

For further information on data tables, see the *CICS/MVS Data Tables Guide*.

DL/I

Function shipping allows a CICS transaction to access IMS DB databases associated with a remote CICS/ESA, CICS/MVS, or CICS/OS/VS system, or DL/I DOS/VS databases associated with a remote CICS/VSE or CICS/DOS/VS system. (See "Chapter 1.1. Introduction to CICS intercommunication" on page 3 for a list of systems with which CICS/MVS 2.1.2 can communicate.)

As with File Control, updates to remote DL/I databases are not committed until the application reaches a syncpoint. With IMS DB, it is not possible to schedule more than one PSB per logical unit of work, even when both PSBs are defined to be on remote systems. Hence linked DL/I updates on different systems cannot be made in a single logical unit of work.

The PSB directory list (PDIR or DLZACT) is used to define a PSB as being on a remote system. The remote system owns the database and the associated PCB definitions. When DL/I access requests are made to another processor system by a CICS/MVS system but no local requests are made, it is not necessary to install IMS DB on the requesting system.

Temporary storage

Intercommunication enables application programs to send data to, or retrieve data from, temporary storage queues located on remote systems. A temporary storage queue is specified as being remote by means of an entry in the local TST. If the queue is to be protected, its queue name (or remote name) must also be defined as recoverable in the TST of the remote system.

Transient data

An application program can access intrapartition or extrapartition transient data queues on remote systems. The Destination Control Table (DCT) in the requesting system defines the named queue as being on the remote system. The DCT entry for the queue in the remote system specifies whether the queue is protected, and whether it has a trigger level and associated terminal.

Extrapartition queues can be defined (in the owning system) as having fixed, variable, or undefined length records.

Many of the uses currently made of transient data and temporary storage queues in a stand-alone CICS system can be extended to an interconnected CICS system environment. For example, a queue of records can be created in a system for processing overnight. Queues also provide another means of handling requests from other systems while freeing the terminal for other requests. The reply can be returned to the terminal as soon as it is ready, and delivered to the operator when there is a lull in entering transactions.

If a transient data destination has an associated transaction, the named transaction must be defined to execute in the system owning the queue; it can not be defined as remote. If there is a terminal associated with the transaction, it can be connected to another CICS system and used via the transaction routing facility of CICS.

The remote naming capability enables a program to send data to the CICS service destinations, such as CSMT, in both local and remote systems.

The mirror transaction and transformer program

CICS supplies a number of mirror transactions, each of which corresponds to a particular "process" name. Their transaction identifiers are CSM1, CSM2, CSM3, CSM5, and CSMI[1]. All these transaction identifiers map to a single mirror program DFHMIR.

Details of the individual mirror transactions are given in "Chapter 3.3. Defining local resources" on page 151. In the rest of this book, they are referred to generally as the mirror transaction, and given the transaction identifier CSM*.

The following description of the mirror transaction and the transformer program is generally applicable to both ISC and MRO function shipping. There are, however, a number of differences in the way that the mirror transaction works under MRO, and a different transformer program is used. These differences are described in "MRO function shipping" on page 26.

The mirror transaction executes as a normal CICS transaction and uses the CICS terminal control program facilities to communicate with the requesting system.

1 If you are using MRO, LU6.1 or LU6.2 Synclevel(2) parallel sessions, CICS runs transaction CSMI with program DFHMIR. If you are using an LU6.2 single-session connection, or an LU6.2 parallel-session connection that supports only synchronization level 1, CICS runs transaction CVMI with program DFHMIRVM. The difference is in the way syncpointing is handled. Unless otherwise stated, all references to CSMI/DFHMIR also refer to CVMI/DFHMIRVM.

In the requesting system, the command level EXEC interface program (for all except DL/I requests) determines that the requested resource is on another system, calls the function-shipping transformer program DFHXFP to transform the request into a form suitable for transmission, and calls on the intercommunication component to send the request to the appropriate connected system. For DL/I requests, part of this function is handled by CICS DL/I interface modules.

The intercommunication component uses CICS terminal control program facilities to send the request to the mirror transaction. The first request to a particular remote system on behalf of a transaction will cause the communication component in the local system to precede the formatted request with the appropriate mirror transaction identifier, in order to attach this transaction in the remote system. Thereafter it keeps track of whether or not the mirror transaction terminates, and reinvokes it as required.

The mirror transaction uses the function-shipping transformer program DFHXFP to decode the formatted request and executes the corresponding command. At completion of the command the mirror transaction uses the transformer program to construct a formatted reply, and returns this to the requesting system. On that system the reply is decoded, again using the transformer program, and used to complete the original command level request made by the application program.

If the mirror transaction is not required to update any protected resources, and no previous request updated a protected resource in its system, the mirror transaction will terminate after sending its reply. However, if the request causes the mirror transaction to change or update a protected resource, or the request is for any DL/I PSB, it will not terminate until the requesting application program issues a synchronization point request or terminates successfully. When the application program issues a synchronization point request, or terminates successfully, the intercommunication component sends a message to the mirror transaction which causes it also to issue a synchronization point request and terminate. The successful synchronization point by the mirror transaction is indicated in a response sent back to the requesting system, which then completes its synchronization point processing, so committing changes to any protected resources. If DL/I requests have been received from another system, CICS issues a DL/I TERM call as a part of the processing resulting from a synchronization point request made by the application program and executed by the mirror transaction.

The application program is not constrained in the order in which it accesses protected or unprotected resources, nor is it affected by the location of protected resources (they could all be in remote systems, for example). When the application program accesses resources in more than one remote system, the intercommunication component invokes a mirror transaction in each system to execute requests on behalf of the application program. Each mirror transaction follows the above rules for termination, and when the application program reaches a synchronization point, the intercommunication component exchanges synchronization point messages with those mirror transactions that have not yet terminated (if any). This is referred to as the multiple-mirror situation.

The mirror transaction uses the CICS command level interface to execute CICS requests and the DL/I CALL interface to execute DL/I requests. The request is thus processed as for any other transaction and the requested resource is located in the appropriate resource table. If its entry defines the resource as being remote, the mirror transaction's request is formatted for transmission and sent to yet another mirror transaction in the specified system. This situation is referred to as "chained-mirror." It is strongly recommended that the system designer avoids defining a connected system in which chained mirror requests will occur, except when the requests involved do not access protected resources, or are inquiry-only requests.

MRO function shipping

For MRO function shipping, the operation of the mirror transaction is slightly different from that described in the previous section.

Reusable mirror tasks

For ISC function shipping, mirror tasks are terminated when they have no further work to do, and each new invocation of the mirror transaction requires a new mirror task to be attached. For MRO, however, the mirror task is detached from the interregion link when it has no further work to do, but is **suspended** rather than terminated.

The "suspend" command issued by the mirror transaction carries a fixed timeout value of 2 seconds, so that the task is detached if it has not been reused when this time expires. This mechanism allows the number of reusable mirror tasks to vary dynamically according to the current interregion traffic. The trade-off in overheads associated with attaching and detaching mirror tasks and maintaining suspended tasks is thus optimized.

The timeout value specified on the suspend command is fixed by the implementation, and cannot be altered.

Suspended mirror tasks are eligible to service any function shipping request on an interregion link. A mirror task attach is thus necessary only when a suspended mirror task is not available.

Long-running mirror tasks

Mirror tasks are normally terminated (or suspended) as soon as possible, to keep the number of active tasks to a minimum and to avoid holding on to the session for long periods.

However, for some applications, it is more efficient to retain both the mirror task and the session until the next synchronization point, even though this is not required for data integrity. For example, a transaction that issues a large number of READ FILE requests to a remote system may be better served by a single mirror task, rather than by a separate mirror task for each request. In this way, the overheads of allocating sessions on the sending side and attaching mirror tasks on the receiving side can be reduced.

Mirror tasks that wait for the next syncpoint, even though they logically do not need to do so, are called **long-running** mirrors. They are applicable to MRO links only, and are specified, *on the system on which the mirror will run*, by coding MROLRM=YES in the system initialization table. A long-running mirror is terminated by the next syncpoint (or RETURN) on the sending side.

Figures 5 and 6 show the action of the mirror for MROLRM=NO and MROLRM=YES respectively.

Suspension and resumption of mirror tasks

A mirror task is required to service any incoming request that specifies the CICS mirror transaction CSMI or one of the architected processes CSM1, CSM2, CSM3, or CSM5. A mirror task can be obtained in one of two ways:

1. By resuming a suspended mirror task, if one is available.
2. By attaching a new mirror task in the normal way.

When a mirror task has no further work to do, it issues a syncpoint if necessary, and calls the monitoring program just as if it were terminating. It is then disconnected from the interregion link and, normally, suspended with a 2-second timeout value.

The maximum number of mirror tasks that can ever be created in a CICS region is equal to the number of receive sessions that are defined to other CICS regions, although this number is unlikely to be approached in practice.

The number of suspended mirror tasks peaks whenever a large number of mirror tasks finish within the 2-second timeout period. For example, to reach 100 suspended mirror tasks would require sufficient function shipping requests to attach 100 mirror tasks, and for all these tasks to finish and be suspended within a two-second period, with no additional requests to cause resumption of a suspended task.

You can limit the number of suspended mirror tasks in your system by means of the MAXSMIR operand of the system initialization table. If the suspension of a mirror task would cause the MAXSMIR value (default 999) to be exceeded, the mirror task is terminated instead.

Other conditions that can cause a mirror task to be terminated rather than suspended are:

- CICS is "short-on-storage".
- The maximum task count has been reached.
- Interregion communication is being closed (CEMT SET IRC CLOSED).

  In this case, all suspended mirror tasks are resumed and allowed to terminate.

Suspended mirror tasks contribute towards the maximum task count.

The short-path transformer

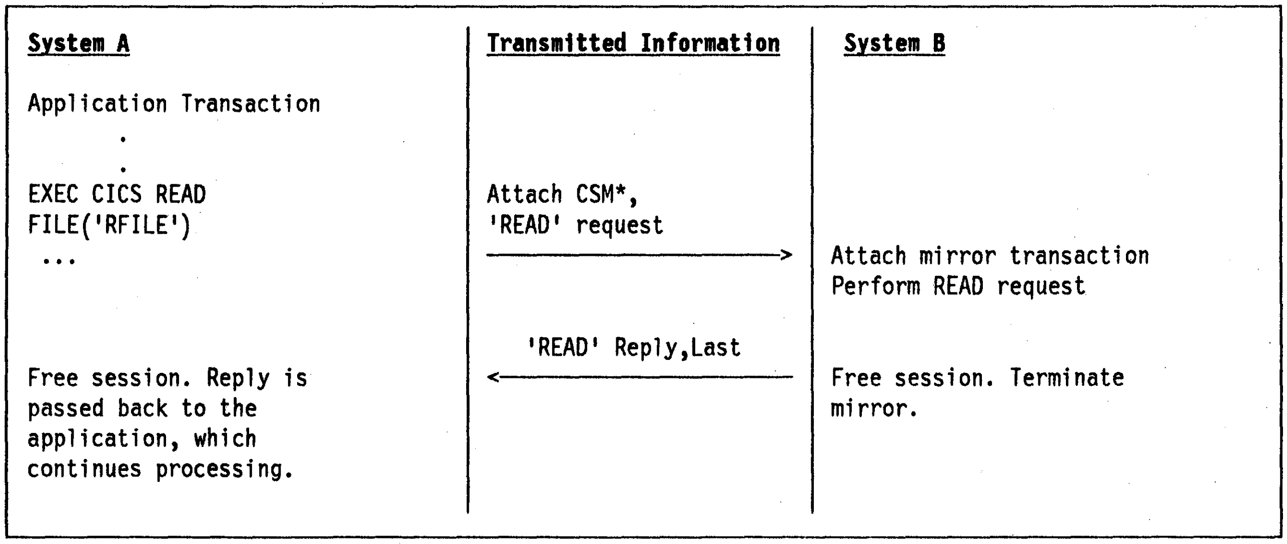
CICS use a special transformer program for function shipping over MRO links. This transformer program is designed to optimize the path length involved in the construction of the terminal input/output areas (TIOA) that are sent on an MRO session for function shipping. It does this by using a private CICS format for the transformed request, rather than the architected format defined by SNA.

CICS uses the short-path transformer to ship file control, transient data, temporary storage, and interval control (asynchronous processing) requests. The short-path transformer is not used for DL/I requests. The shipped request always specifies the CICS mirror transaction CSMI; architected process names are not used.

Function shipping — examples

This section gives some examples to illustrate the life time of the mirror transaction and the information flowing between the application and its mirror (CSM*). The examples contrast the action of the mirror transaction when accessing protected and non-protected resources on behalf of the application program, over MRO or ISC links, with and without MRO long-running mirror tasks. Further details can be found in the *CICS/MVS Diagnosis Reference* manual.

Note: In the following examples, references to the suspension and resumption of the mirror apply only to MRO function shipping (see "Reusable mirror tasks" on page 26).



```
System A                   Transmitted Information   System B

Application Transaction
       .
       .
EXEC CICS READ             Attach CSM*,
FILE('RFILE')              'READ' request
 ...                       -------------------->     Attach mirror transaction
                                                     Perform READ request

                             'READ' Reply,Last
Free session. Reply is     <--------------------     Free session. Terminate
passed back to the                                   mirror.
application, which
continues processing.
```

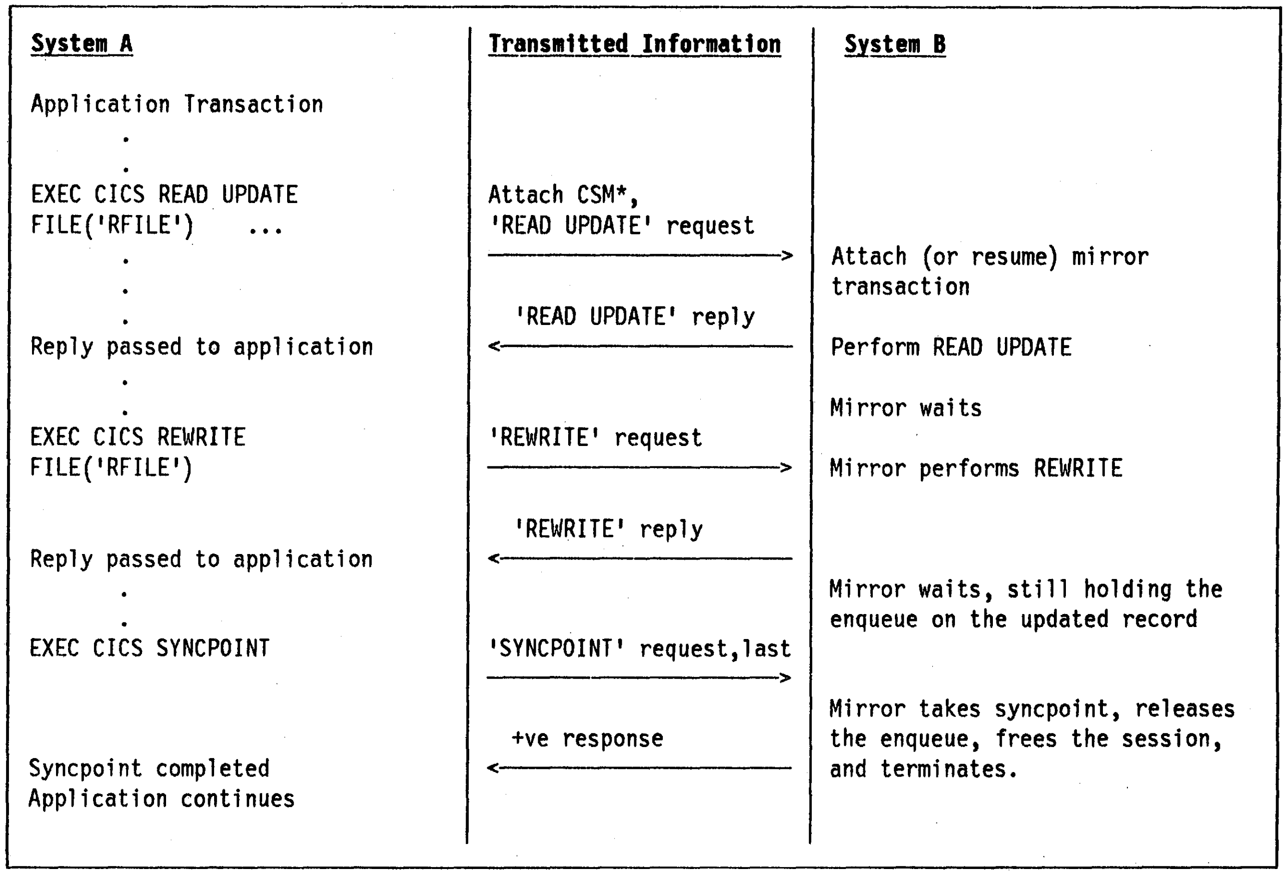
Figure 4. ISC function shipping — simple enquiry. Here no resource is being changed; the session is freed and the mirror task is terminated immediately.

System A	Transmitted Information	System B
		{DFHSIT MROLRM=NO}
Application Transaction . .		
EXEC CICS READ FILE('RFILE') ...	Attach CSM*, 'READ' request ——————————>	Attach (or resume) mirror transaction. Perform READ request
Free session. Reply is passed back to the application, which continues processing.	'READ' Reply,Last <——————————	Free session. Terminate mirror.

Figure 5. MRO function shipping — simple enquiry. Here no resource is being changed. Because long-running mirror tasks are not specified, the session is freed by system B and the mirror task is terminated immediately.

System A	Transmitted Information	System B
		{DFHSIT MROLRM=YES}
Application Transaction . .		
EXEC CICS READ FILE('RFILE') ...	Attach CSM*, 'READ' request ——————————>	Attach (or resume) mirror transaction. Perform READ request
Hold session. Reply is passed back to the application, which continues processing.	'READ' Reply <——————————	Hold session. Mirror task waits for next request.

Figure 6. MRO function shipping — simple enquiry. Here no resource is being changed. However, because long-running mirror tasks are specified, the session is held by system B and the mirror task waits for the next request.



```
System A                        Transmitted Information     System B

Application Transaction
      .
      .
EXEC CICS READ UPDATE           Attach CSM*,
FILE('RFILE')    ...            'READ UPDATE' request
      .                         ------------------------>   Attach (or resume) mirror
      .                                                     transaction
      .                           'READ UPDATE' reply
Reply passed to application     <------------------------   Perform READ UPDATE
      .
      .                                                     Mirror waits
EXEC CICS REWRITE               'REWRITE' request
FILE('RFILE')                   ------------------------>   Mirror performs REWRITE

                                  'REWRITE' reply
Reply passed to application     <------------------------
      .                                                     Mirror waits, still holding the
      .                                                     enqueue on the updated record
EXEC CICS SYNCPOINT             'SYNCPOINT' request,last
                                ------------------------>
                                                            Mirror takes syncpoint, releases
                                  +ve response              the enqueue, frees the session,
Syncpoint completed             <------------------------   and terminates.
Application continues
```

Figure 7. Function shipping — update. Because the mirror must wait for the REWRITE, it becomes long-running and does not terminate until SYNCPOINT is received. Note that the enqueue on the updated record would not be held beyond the REWRITE command if the file was not recoverable.

System A	**Transmitted Information**	**System B**
Application Transaction . .		
EXEC CICS READ UPDATE FILE('RFILE')	Attach CSM*, 'READ UPDATE' request ————————>	Attach (or resume) mirror transaction.
Reply passed to application .	'READ UPDATE' reply <————————	Perform READ UPDATE Mirror waits
EXEC CICS REWRITE FILE('RFILE') . .	'REWRITE' request ————————>	Mirror performs REWRITE
Reply passed to application . .	'REWRITE' reply <————————	Mirror waits
EXEC CICS SYNCPOINT	'SYNCPOINT' request,last ————————>	Mirror attempts syncpoint but abends (logging error, for example). Mirror backs out and terminates.
Application is abended and backs out.	−ve response <————————	
Message routed to CSMT	Abend message <————————	Session freed

Figure 8. Function shipping — update with ABEND. This is similar to the previous example, except that an abend occurs during syncpoint processing.

Chapter 1.5. Asynchronous processing

Asynchronous processing provides a means of distributing the processing that is required by an application between systems in an intercommunication environment. Unlike distributed transaction processing, however, the processing is **asynchronous**.

In distributed transaction processing, a session is held by two transactions for the period of a "conversation" between them, and requests and replies can be directly correlated.

In asynchronous processing, the processing is independent of the sessions on which requests are sent and replies are received. No direct correlation can be made between a request and a reply, and no assumptions can be made about the timing of the reply. These differences are illustrated in Figure 9.

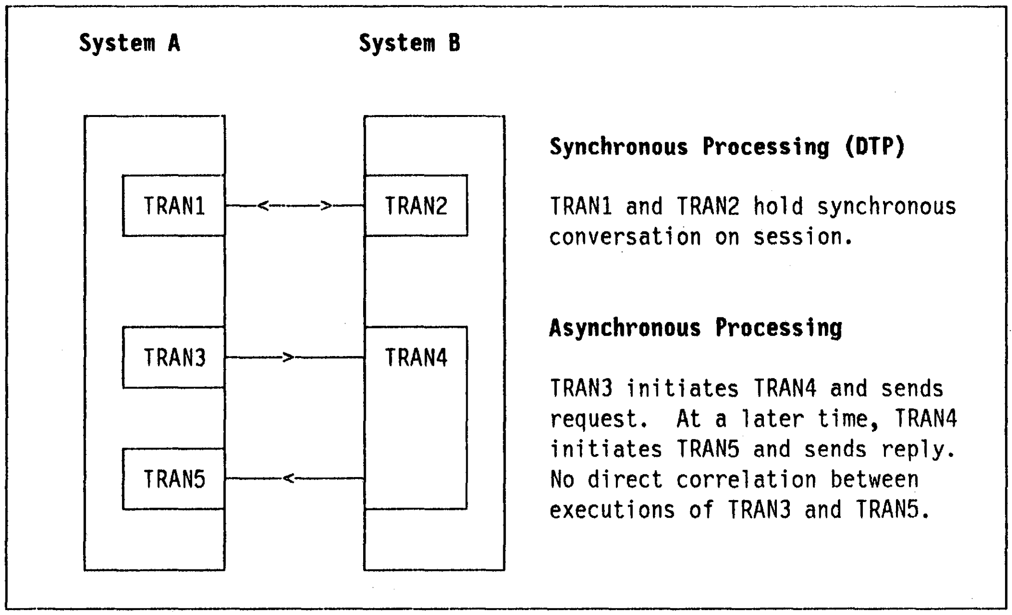

System A
System B
TRAN1
TRAN2
TRAN3
TRAN4
TRAN5
Synchronous Processing (DTP)
TRAN1 and TRAN2 hold synchronous conversation on session.
Asynchronous Processing
TRAN3 initiates TRAN4 and sends request. At a later time, TRAN4 initiates TRAN5 and sends reply. No direct correlation between executions of TRAN3 and TRAN5.

Figure 9. Synchronous and asynchronous processing compared

A typical application area for asynchronous processing is online inquiry on remote databases; for example, a credit rating check application. A terminal operator can use a local transaction to enter a succession of inquiries without waiting for a reply to each individual inquiry. For each inquiry, the local transaction initiates a remote transaction to process the request, so that many copies of the remote transaction can be executing concurrently. The remote transactions send their replies by initiating a local transaction (possibly the same transaction) to deliver the output to the operator terminal. The replies may not arrive in the same order as the inquiries were issued; correlation between the inquiries and the replies must be made by means of fields in the user data.

In general, asynchronous processing is applicable to any situation in which it is not necessary or desirable to tie-up local resources while a remote request is being processed.

Asynchronous processing is not suitable for applications that involve synchronized changes to local and remote resources; for example, it cannot be used to carry out simultaneous linked updates to data split between two systems.

Asynchronous processing methods

In CICS, asynchronous processing can be done in either of two ways:

1. By using the interval control commands START and RETRIEVE.

   You can use the START command to schedule a transaction in a remote system in much the same way as you would in a single CICS system. This type of asynchronous processing is essentially a form of CICS function shipping, and as such, it is transparent to the application. The systems programmer determines whether the attached transaction is local or remote.

   If you use the START command for asynchronous processing, you can communicate only with systems that support the special protocol needed for function shipping; that is, CICS itself and IMS.

   A CICS transaction that is initiated by a remotely-issued start request can use the RETRIEVE command to retrieve any data associated with the request. A task that is not associated with a terminal can access only the single data record associated with the original start request. A task associated with a terminal can retrieve many data records. Each data record comes from a START command, specifying the same transaction and terminal.

2. By using distributed transaction processing (DTP).

   This is essentially a cross-system method and has no single-system equivalent. You can use it to initiate a transaction in a remote system that supports one of the DTP protocols.

   When you use DTP to attach a remote transaction, you also allocate a session and start a conversation. This permits you to send data directly and, if you want, to receive data from the remote transaction. Your transaction design determines the format and volume of the data you exchange. For example, you can use repeated SEND commands to pass multi-record files.

   When you have exchanged data, you can terminate the conversation and quit the local transaction, leaving the remote transaction to run on independently.

   The procedure to be followed by the two transactions during the time that they are working together is determined by the application programming interface (API) for the protocol you are using. LUTYPE6.2 is the preferred one, although you must use LUTYPE6.1 if you want to communicate with IMS. You may want to take advantage of the flexible data exchange facilities by employing this method across MRO links too.

   Whatever protocol you decide to use, you must observe the rules it imposes. However short the conversation, during the time it is in progress, the processing is synchronous. In terms of command sequencing, error recovery and syncpointing, the normal DTP rules apply.

Note: If the remote transaction has been defined with RSLC(YES) or RSLC(EXTERNAL), it cannot access any resources after it has freed the session that is its principal facility.

In both forms of asynchronous processing (and also in synchronous processing), a CICS transaction can use the CICS ASSIGN command, plus possibly an examination of the EIB, to determine how it was initiated.

CICS-to-IMS communication includes a special case of the DTP method described above. Because it restricts data communication to one SEND LAST command answered by a single RECEIVE, this book refers to it elsewhere as the SEND/RECEIVE interface. The circumstances under which it is used are described in "Chapter 4.8. CICS-to-IMS applications" on page 263.

The remainder of this chapter is devoted to asynchronous processing using START and RETRIEVE commands. Distributed transaction processing is described in "Chapter 1.7. Distributed transaction processing" on page 55.

Asynchronous processing using START/RETRIEVE commands

CICS interval control is described in the *CICS/MVS Application Programmer's Reference* manual. The interval control commands that can be used for asynchronous processing are:

- START
- CANCEL
- RETRIEVE.

Starting and canceling remote transactions

The interval control START command is used to schedule transactions asynchronously in remote CICS and IMS systems. The command causes an "attach" FMH and a concatenated "scheduler" FMH to be sent with the data to the remote system; that is, the command is effectively "function shipped". If the remote system is CICS, the mirror transaction is invoked in the remote system to issue the START command on that system. The FMH DSECTs are defined in the *CICS/MVS Data Areas* manual.

For CICS-to-CICS communication, you can include time-control information on the shipped START command in the normal way, by means of the INTERVAL or TIME option. A TIME specification is converted by CICS to a time interval, relative to the local clock, before the command is shipped. Because each end of an intersystem link may be in a different time zone, it is usually better to think in terms of time intervals, rather than absolute times, for intersystem communication.

Note particularly that the time interval specified on a START command specifies the time at which the remote transaction is to be initiated, not the time at which the request is to be shipped to the remote system.

A START command shipped to a remote CICS system can be canceled at any time up to its expiration time by shipping a CANCEL command to the same system. The particular START command is uniquely identified by an identifier (REQID) which you can specify on the START command and on the associated CANCEL command. The CANCEL command can be issued by any task that "knows" the identifier.

Time control cannot be specified for START commands sent to IMS systems; INTERVAL(0) must be specified or allowed to default. Consequently, start requests for IMS transactions cannot be canceled after they have been issued.

Passing information with the START command

The START command has a number of options that enable information to be made available to the remote transaction when it is started. If the remote transaction is in a CICS system, the information is acquired by means of a RETRIEVE command. The information that can be specified is summarized in the following list:

- User data — specified in the FROM option.

  This is the principal way in which data can be passed to the remote transaction.

  For CICS-to-CICS communication, additional data can be made available in a transient data or temporary storage queue named in the QUEUE option. The queue can be on any CICS system that is accessible to the system on which the remote transaction is executed.

  The QUEUE option cannot be used for CICS-to-IMS communication.

- A "terminal" name — specified in the TERMID option.

  For CICS-to-CICS communication, this is the name of a terminal that is to be associated with the remote transaction when it is initiated. The terminal might be defined on the region that owns the remote transaction and not be owned by that region. If so, it is acquired by the automatic transaction initiation (ATI) facility of transaction routing (see "Automatic transaction initiation" on page 46).

  The global user exits XICTENF and XALTENF can be coded to cover the case where the terminal is *shippable* but not yet defined in the region that owns the remote transaction. See "Shipping terminals for automatic transaction initiation" on page 48.

  For CICS-to-IMS communication, it is a transaction code or an LTERM name.

- A transaction name and an associated terminal name — specified in the RTRANSID and RTERMID options.

  These options provide the means for the remote transaction to pass a reply to the local system, by specifying a transaction that is to be invoked and a terminal that is to be associated with it.

The use of any of these options is optional.

Passing an APPLID with the START command

If you have a transaction that can be started from several different systems, and is required to issue a start command to the system that initiated it, you can arrange for all of the invoking transactions to send their local system APPLID as part of the user data in the START command. A transaction can obtain its local APPLID by means of an ASSIGN APPLID command. (Note that this command returns the **generic** name of the applid.)

The transaction that is started can then find its own, local, SYSID for the passed APPLID by means of an EXTRACT TCT command, and name that SYSID in the START command that it issues in reply. This approach cannot be used for MRO connections, because the EXTRACT TCT command is not supported.

Improving performance of intersystem START requests

In many enquiry-only applications, sophisticated error-checking and recovery procedures are not justified. Where the transactions make enquiries only, the terminal operator can retry an operation if no reply is received within a certain time. In such a situation, the number of messages to and from the remote system can be substantially reduced by means of the NOCHECK option of the START command. Where the connection between the two systems is via VTAM, this can result in considerably improved performance. The price paid for better performance is the inability of CICS to detect certain types of error in the START command.

A typical use for the START NOCHECK command is in the remote enquiry application described at the beginning of this chapter on page 33.

The transaction attached as a result of the terminal operator's enquiry issues an appropriate START command with the NOCHECK option, which causes a single message to be sent to the appropriate remote system to start, asynchronously, a transaction that makes the enquiry. The command should specify the operator's terminal identifier. The transaction attached to the operator's terminal can now terminate, leaving the terminal available for either receiving the answer or initiating another request.

The remote system performs the requested enquiry on its local database, then issues a start request for the originating system. This command passes back the requested data, together with the operator's terminal identifier. Again, only one message passes between the two systems. The transaction that is then started in the originating system must format the data and display it at the operator's terminal.

If a system or session fails, the terminal operator must reenter his enquiry, and be prepared to receive duplicate replies. To aid him, either a correlation field must be shipped with each request, or all replies must be self-describing.

An example of intercommunication using the NOCHECK option is given at the end of this chapter in Figure 11 on page 42.

The NOCHECK option is always required when shipping of the START command is queued pending the establishment of links with the remote system (see "Local queuing" on page 38), or if the request is being shipped to IMS.

Including start request delivery in a logical unit of work

The delivery of a start request to a remote system can be made part of a logical unit of work by specifying the PROTECT option on the START command. The PROTECT option indicates that the remote transaction must not be scheduled until the local one has successfully completed a synchronization point. (It can take the synchronization point either by issuing a SYNCPOINT command or by terminating.)

Successful completion of the syncpoint guarantees that the start request has been delivered to the remote system. It does not guarantee that the remote transaction has completed, or even that it will be initiated.

If the remote system is IMS, no message must cross the link between the START command and the synchronization point. Both PROTECT and NOCHECK must be specified for all IMS recoverable transactions.

Deferred sending of START requests

For START commands with the NOCHECK option, whether or not PROTECT is specified, CICS defers transmission of the request to the remote system until one of the following events occurs:

- The transaction issues a further START command (or any function shipping request) for the same system
- The transaction issues a SYNCPOINT command
- The transaction terminates (implicit syncpoint).

The first, or only, start request transmitted from a transaction to a remote system carries the begin-bracket indicator; the last, or only, request carries the end-bracket indicator. Also, if any of the start requests issued by the transaction specifies PROTECT, the last request carries the syncpoint-request indicator. Deferred sending allows the indicators to be added to the deferred data, and thus reduces the number of transmissions required.

The sequence of requests is transmitted within a single SNA bracket and, if the remote system is CICS, all the requests are handled by the same mirror task.

For IMS, as stated in the previous section, no message must cross the link between a START request and the following syncpoint. Therefore, you cannot send multiple START NOCHECK PROTECT requests to IMS. Each request must be followed by a SYNCPOINT command, or by termination of the transaction.

Local queuing

When a local transaction is ready to ship a START request, the intersystem facilities may be unavailable, either because the remote system is not active or because a connection cannot be established. The normal CICS action in these circumstances is to raise the SYSIDERR condition. This can be avoided by arranging for the request to be queued locally, and forwarded when the required link is in service. The storing and forwarding can be carried out by user-written transactions, or by the CICS local queuing facility.

CICS can provide local queuing for START commands intended to initiate transactions on remote systems. The commands must include the NOCHECK option, and local queuing must be specified by means of either a user exit invoked from the CICS routine DFHISP, or the LOCALQ operand in the local definition of the remote transaction. The user exit can specify local queuing for all requests from the local system; the LOCALQ operand can specify local queuing for all requests from the local system for a particular remote transaction.

Local queuing is ineffective for START requests that specify the SYSID option.

Data retrieval by a started transaction

A CICS transaction that is started by a start request can acquire the user data and other information associated with the request by means of the RETRIEVE command.

In accordance with the normal rules for CICS interval control, a start request for a particular transaction that carries both user data and a terminal identifier will be queued if the transaction is already active and associated with the same terminal. During the waiting period, the data associated with the queued request can be accessed by the active transaction by means of a further RETRIEVE command. This has the effect of canceling the queued start request.

It is thus possible to design transactions that can handle the data associated with multiple start requests. Typically, a long-running transaction could be designed to accept multiple enquiries from a terminal and ship start requests to a remote system. From time to time, the transaction would issue RETRIEVE commands to receive the replies, the absence of further replies being indicated by the ENDDATA condition.

The WAIT option of the RETRIEVE command can be used to put the transaction into a WAIT state pending the arrival of the next start request from the remote system. Suitable precautions must be made to ensure that the transaction does not get into a permanent wait state in the absence of further start requests.

Terminal acquisition by a remotely-initiated CICS transaction

When a CICS transaction is started by a start request that names a terminal (TERMID), CICS makes the terminal available to the transaction as its principal facility. Its makes no difference whether the start request was issued by a user-transaction in the local CICS system or was received from a remote system and issued by the mirror transaction.

Starting transactions with ISC or MRO sessions

You can name a system, rather than a terminal, in the TERMID option of the START command.

If CICS finds that the "terminal" named in a locally- or remotely-issued start request is a system, it selects an available session to that system and makes it the principal facility (see "Terminology" on page 161) of the started transaction. If no session is available, the request is queued until there is one.

If the link to the system is an LUTYPE6.2 link, CICS uses the modename associated with the transaction definition to select a class-of-service for the session.

System programming considerations

This section discusses the CICS resources that must be defined for asynchronous processing. Information on how to define the resources is given in "Part 3. Resource definition and master terminal operation" on page 89.

- A link to remote system must be defined.
- Remote transactions that are to be initiated by start requests must be defined as remote resources to the local CICS system. This is not necessary, however, for transactions that are initiated only by START commands that name the remote system explicitly in the SYSID option.
- If the QUEUE option is used, the named queue must be defined on the system to which the start request is shipped. The queue can be either a local or a remote resource on that system.
- If a START request names a "reply" transaction, that transaction must be defined on the system to which the start request is shipped.

Asynchronous processing — examples

Note: In the following examples, references to the suspension and resumption of the mirror apply only to CICS systems using MRO (see "Reusable mirror tasks" on page 26).

System A	**Transmitted Information**	**System B**

{DFHSIT MROLRM=YES}
 	
Transaction TRX
initiated by terminal T1
 	
EXEC CICS START
TRANSID('TRY')
RTRANSID('TRZ')
RTERMID('T1')
FROM(area)
LENGTH(length)
	Attach CSM*
	'SCHEDULE' request for
	transaction
	——————————>
		Attach (or resume) mirror
		transaction. Perform START
		request for transaction TRY.

Free session. Pass return code to application program. Continue processing.	'SCHEDULE' Reply,last <—————————— Session available for remote requests from other transactions in system A or B.	Free session. Terminate (or suspend) mirror. Transaction TRY is dispatched and starts processing. EXEC CICS RETRIEVE INTO (area) LENGTH(length) RTRANSID(TR) RTERMID(T) (TR has value 'TRZ', T has value 'T1') Processing based on data acquired. Results put into TS queue named RQUE. EXEC CICS START TRANSID(TR) TERMID(T) QUEUE('RQUE') (TR has value 'TRZ', T has value 'T1')
Attach (or resume) mirror transaction. (continued)	Attach CSM* 'SCHEDULE' request for transaction <——————————	

Figure 10 (Part 1 of 2). Asynchronous processing — remote transaction initiation

System A	**Transmitted Information**	**System B**
Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'.		
	'SCHEDULE' Reply ————————————>	
mirror waits for SYNCPOINT.		
		RETURN (implicit syncpoint)
	'SYNCPOINT' request,last <————————————	
	+ve response ————————————>	
Free session. Terminate (or suspend) mirror.		
Transaction TRZ is dispatched on terminal T1 and starts processing.		

EXEC CICS RETRIEVE
     INTO(area)
     LENGTH(length)
     QUEUE(Q)
Q has value 'RQUE'		

Transaction now uses function shipping to read and then to delete the remote queue.		

Figure 10 (Part 2 of 2). Asynchronous processing — remote transaction initiation. This example shows an MRO connection with long-running mirrors (MROLRM) specified for system A but not for system B. Note the different action of the mirror transaction on the two systems.

System A	**Transmitted Information**	**System B**
Transaction TRX initiated by terminal T1		

EXEC CICS START
         TRANSID('TRY')
         RTRANSID('TRZ')
         RTERMID('T1')
         FROM(area)
         LENGTH(length)
         NOCHECK		

(continued)		

Figure 11 (Part 1 of 2). Asynchronous processing — remote transaction initiation using NOCHECK

System A	**Transmitted Information**	**System B**
Terminate, and free terminal T1. T1 could now initiate another transaction, but TRZ could not start until T1 became free again.	Attach CSM* 'SCHEDULE' request for trans, last (no reply) ————————>	Attach (or resume) mirror. Perform START request for transaction TRY. Free session. Terminate (or suspend) mirror.
	session available	
		Transaction TRY is dispatched and starts processing. EXEC CICS RETRIEVE INTO (area) LENGTH(length) RTRANSID(TR) RTERMID(T) (TR has value 'TRZ', T has value 'T1')
		Processing based on data acquired. Reply put in data area REP.

		EXEC CICS START
		TRANSID(TR)
		FROM(REP)
		LENGTH(length)
		TERMID(T)
		NOCHECK
		(TR has value 'TRZ',
		T has value 'T1')

	Attach CSM* 'SCHEDULE' request for trans, last (no reply) <————————	TRY terminates
Attach (or resume) mirror transaction		
Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'. Free session.	session available	
Terminate (or suspend) mirror.		
Transaction TRZ is dispatched on terminal T1 and starts processing.		

Figure 11 (Part 2 of 2). Asynchronous processing — remote transaction initiation using NOCHECK. This example shows an ISC connection, or an MRO connection without long-running mirrors.

Chapter 1.6. CICS transaction routing

CICS transaction routing allows terminals connected to one CICS system to run with transactions in another, connected, CICS system. The two systems can be connected either by MRO links or by LUTYPE6.2 (APPC) ISC links. Transaction routing across LUTYPE6.1 links is not supported.

A terminal operator at a terminal connected to one CICS system (the terminal-owning region or **TOR**) can enter a transaction code for a transaction without being aware of the location of the transaction. If the transaction is "remote", CICS will route the request to the application-owning region or **AOR**, and the transaction will run exactly as if the terminal were attached to the application-owning region.

Similarly, a transaction that is started by automatic transaction initiation (ATI) can acquire a terminal that is owned by a different, connected, system.

CICS handles all routing of requests and replies between the two systems, and transactions can usually be designed and coded without regard to the fact that the terminal is connected to another CICS system.

To communicate with the terminal, the application program can use the terminal control, BMS, or batch data interchange facilities of CICS. Mapping and data interchange functions are performed in the application-owning region. BMS paging operations are performed on the terminal-owning region. See "Basic mapping support" on page 52 for more information on BMS operations.

Both conversational and pseudoconversational transactions are supported. The various transactions that make up a pseudoconversational transaction can run on different systems.

Applications written for single CICS systems can use transaction routing without, in most cases, any reprogramming. System errors associated with cross-system traffic will cause the application to abend.

The relay program

When a terminal operator enters a transaction code for a transaction which is in a remote system, the transaction that is attached executes a CICS-supplied program (DFHCRP) known as the **relay program**. This program, which always executes in the terminal-owning region, provides the communication mechanism between the terminal and the remote transaction.

The transaction that is attached in the terminal-owning region is a user-defined transaction with user-defined attributes; usually those of the "real" transaction in the remote region. However, because it executes the relay program, it is often called the **relay transaction**.

When the relay transaction is attached, it acquires an interregion or intersystem session and sends a request to the remote system to cause the "real" user transaction to be started. In the application-owning region, the terminal is represented by a control block known as the **surrogate** TCTTE. This TCTTE becomes the transaction's principal facility, and is indistinguishable by the transaction from a "real" terminal entry. However, if the transaction issues a request to its principal facility, the request is intercepted by the CICS terminal control program and shipped back to the relay transaction over the interregion or intersystem session. The relay transaction then issues the request or output to the terminal. In a similar way, terminal status and input are shipped by the relay transaction to the user transaction.

Automatic transaction initiation is handled in a similar way. If a transaction that is initiated by ATI requires a terminal that is connected to another system, a request to start the relay transaction is sent to the terminal-owning region. When the terminal is free, the relay transaction is connected to it.

The relay transaction remains in existence for the life of the user transaction and has exclusive use of the session to the remote system during this period. When the user's transaction terminates, an indication is sent to the relay transaction, which then also terminates and frees the terminal.

If the user application takes a synchronization point, and if it was defined as a protected task requiring committed output messages, an indication is sent to the relay transaction, which then takes its own synchronization point. Each of the two CICS systems maintains its own system log. Committed output messages are logged on the terminal-owning region.

Combined transaction routing and function shipping

A user's transaction can be in session with only one relay transaction at a time (because a transaction can converse with only one principal facility). But it can be in session with several mirror transactions and a relay transaction (it may have several function shipping requests outstanding). A mirror transaction can be in the same CICS system as the relay or a different one; in the former case, the user's transaction will be using two simultaneous sessions between the two systems.

Automatic transaction initiation

Automatic transaction initiation (ATI) is the process whereby a transaction request made internally within a CICS system or systems network leads to the scheduling of the transaction.

CICS transaction routing allows an ATI request for a transaction owned by a particular CICS system to name a terminal that is owned by another, connected system.

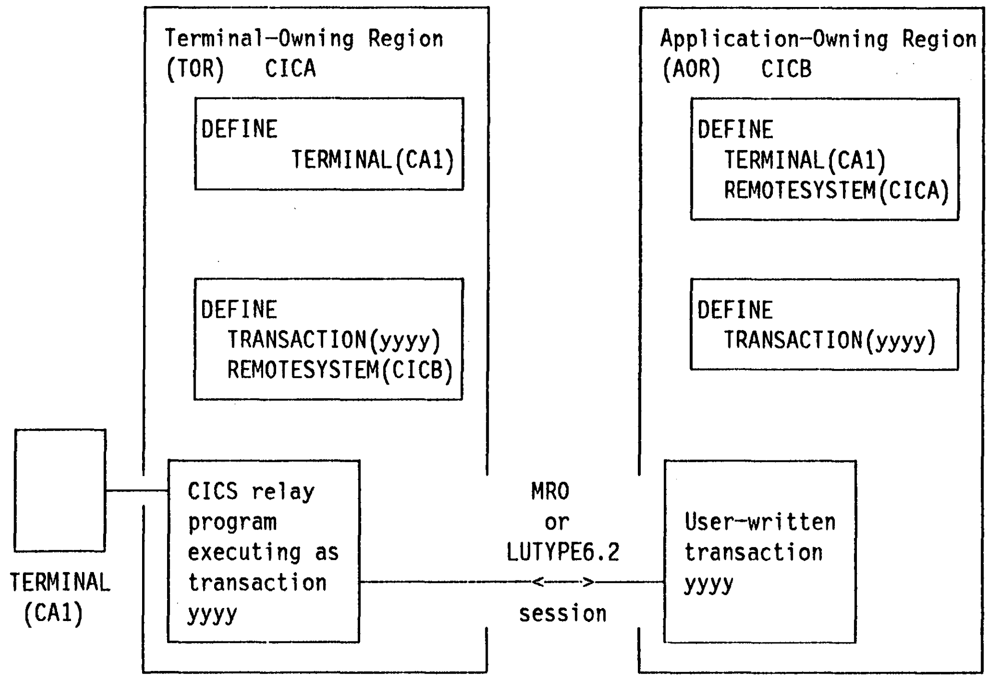
Although the original ATI request occurs in the application-owning region, it is sent by CICS to the terminal-owning region for execution. Here it causes the relay program to be initiated, with the specified terminal. The "real" transaction is then accessed in the manner described for terminal-originated transaction routing.

ATI requests are queued in the application-owning region if the link to the terminal-owning region is not available, and subsequently in the terminal-owning region if the terminal is already in use.

The overall effect is to create a "single-system" view of ATI as far as the application-owning region is concerned; the fact that the terminal is remote does not affect the way in which ATI appears to operate.

In the application-owning region, the normal rules for ATI apply. The transaction can be initiated from a transient data queue, when the trigger level is reached, or on expiry of an interval control "start" request. Note particularly that, for transient data initiation, the transient data queue must be in the same system as the transaction. Transaction routing does not enable transient data queue entries to initiate remote transactions.

In Figure 12, transaction yyyy in system CICB can be initiated by an ATI request that names the remote terminal CA1. The ATI request is sent to system CICA, which owns terminal CA1. Here the terminal is acquired, and the initiation of transaction yyyy, which is defined as remote in system CICA, causes the relay


Terminal-Owning Region
(TOR) CICA
DEFINE TERMINAL(CA1)
DEFINE TRANSACTION(yyyy) REMOTESYSTEM(CICB)
CICS relay program executing as transaction yyyy
TERMINAL (CA1)
MRO or LUTYPE6.2 <——> session
Application-Owning Region
(AOR) CICB
DEFINE TERMINAL(CA1) REMOTESYSTEM(CICA)
DEFINE TRANSACTION(yyyy)
User-written transaction yyyy

Notes:

1. You can use macro-level definition or resource definition online (RDO) to define the terminal, the transaction, and the intersystem or interregion session. See "Part 3. Resource definition and master terminal operation" on page 89.
2. The terminal need not be defined in the application-owning region if RDO is used to define the terminal as "shippable" in the terminal-owning region. See "Shipping terminal definitions" on page 141.

Figure 12. Transaction routing

program to be initiated. The relay program initiates transaction yyyy in system CICB by means of the normal transaction routing facility.

Figure 12 shows how the transaction and the terminal are defined in both systems for straightforward ATI operation. However, if you are using *shippable* terminals, CICS allows you to determine the terminal location at ATI scheduling time.

Shipping terminals for automatic transaction initiation

System CICA can cause an ATI request to be executed in System CICB in three ways:

1. CICA function-ships a START request to CICB.
2. CICA function-ships WRITEQ requests for a transient data queue owned by CICB, which eventually triggers.
3. CICA instigates routing to a transaction in CICB, which then issues a START or writes to a transient data queue.

If the ATI request has a terminal associated with it, CICB searches its resources for a definition for that terminal. If it finds that the terminal is remote, it sends the ATI request to the system that is specified on the REMOTESYTEM option of the terminal definition. Remember that an ATI request is executed ultimately in the TOR.

Terminal-not-known condition

To ensure correct functioning of cross-region ATI, you could define your terminals to all the systems on the network that need to use them. However, you cannot do this, if you are using *autoinstall* (see the *CICS/MVS Resource Definition (Online)* manual for information on the autoinstallation of terminals). Autoinstalled terminals are unknown to the system until they log on, and you rely on CICS to ship terminal definitions to all the systems where they are needed (see "Shipping terminal definitions" on page 141). This works fine when routing from a terminal to a remote system, but there are cases where a system is unable to process an ATI request, because it has not been told the location of the associated terminal.

The example shown in Figure 13 on page 49 should make this clear:

- The operator at terminal T1 selects the menu transaction M1 on CICA.
- The menu transaction M1 runs and the operator selects a function that is implemented by transaction X1 in CICB.
- Transaction M1 issues the command:

  ```
  EXEC CICS START
       TRANSID(X1)
       TERMID(T1)
  ```

  and exits.
- CICA function ships the START command to CICB.
- CICB now processes the START command and, in doing so, tries to discover which region owns T1, because this is the region that has to execute the ATI request resulting from the START command.

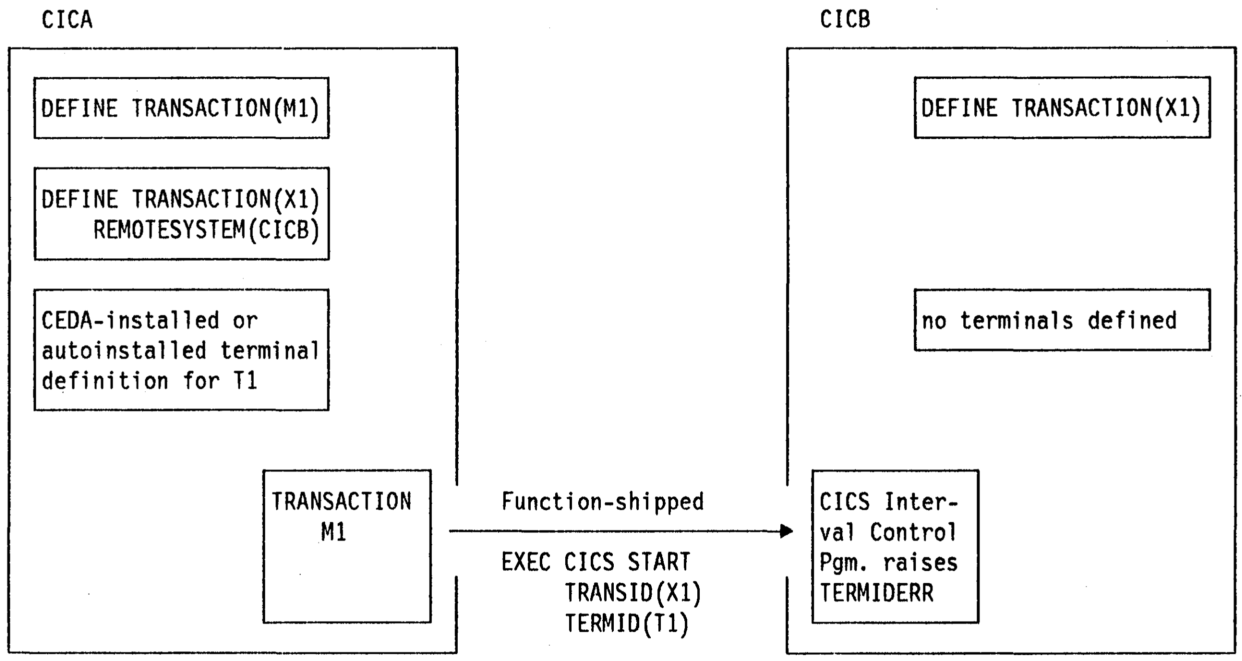

CICA
CICB
DEFINE TRANSACTION(M1)
DEFINE TRANSACTION(X1) REMOTESYSTEM(CICB)
CEDA-installed or autoinstalled terminal definition for T1
TRANSACTION M1
Function-shipped
EXEC CICS START TRANSID(X1) TERMID(T1)
DEFINE TRANSACTION(X1)
no terminals defined
CICS Interval Control Pgm. raises TERMIDERR

Figure 13. Failure of an ATI request in a system where the termid is unknown

- Only if a definition of T1, resulting from an earlier routed transaction, is present will CICB know where to send the ATI request. Assuming no such definition exists, the Interval Control Program rejects the START request with *termiderr.*

The global user exits XICTENF and XALTENF

You, as user of the system, know how this routing problem could be solved, and CICS gives you a way of communicating your solution to the system. The two global user exits XICTENF and XALTENF have been provided. XICTENF is driven when the Interval Control Program (DFHICP) processes a START command and discovers the associated *termid* is not known to the system. XALTENF is driven from the Terminal Allocation Program also when the *termid* is not known.

The Terminal Allocation Program schedules requests resulting both from the eventual execution of a START command and from the transient data queue trigger mechanism. This means that a START command could result in an invocation of both exits.

The program you provide to service one or both of these global user exits has access to a parameter list containing the following information:

- Whether the ATI request resulted from: a START command with data, a START command without data, or a transient data queue trigger.
- Whether or not the START command was issued by a transaction which had been the subject of transaction routing.

- Whether or not the START command was function shipped from another region.
- The identifier of the transaction to be run.
- The identifier of the terminal the transaction should run with.
- The identifier of the terminal associated with the transaction that issued the START command, if this was a routed transaction, or the identifier of the session, if the command was function shipped. Otherwise blanks are returned.
- The *netname* of the last system the START request was shipped from or, if the START was issued locally, the netname of the system last transaction-routed from. Blanks are returned if no remote system was involved.
- The *sysid* corresponding to the previous parameter.

When it returns from the global user exit, your program tells CICS whether the terminal exists and, if it does, you supply either the *netname* or the *sysid* of the TOR. CICS sends the ATI request to the region you specify. As a result, the terminal definition is shipped from the TOR to the AOR, and transaction routing proceeds normally.

There is now a solution to the problem of Figure 13 on page 49. It is only necessary to write a small exit program that returns the CICS-supplied parameters unchanged and sets the return code for *netname returned*.

The events that follow are shown graphically in Figure 14 on page 51:

1. The interval control program accepts the START command and signals acceptance to the issuing system if this is required. It then processes the start request and finds no terminal defined, and so takes the XICTENF exit, which supplies the required *netname*.
2. After the specified interval has expired, or immediately if no interval was specified, the terminal allocation program tries to schedule the ATI request. It finds no terminal defined and takes the exit XALTENF, which again supplies the required *netname*.
3. The ATI request is shipped to CICA. CICA allocates a relay transaction, establishes a transaction routing link to transaction X1 in CICB, and ships a copy of the terminal definition for T1 to CICB.

The *terminal not known* condition can arise in DFHALP during restart; that is, before the user has had a chance to enable any global user exits. If you want to intervene here too, you specify the name of the exit program on the SIT parameter ALEXIT. This facility applies to both warm start and emergency start.

Of course, the example above shows only one of many possible configurations. More complex situations can arise in multi-region networks, but if you have understood the basic solution to the problem, it should be possible for you to apply it generally.

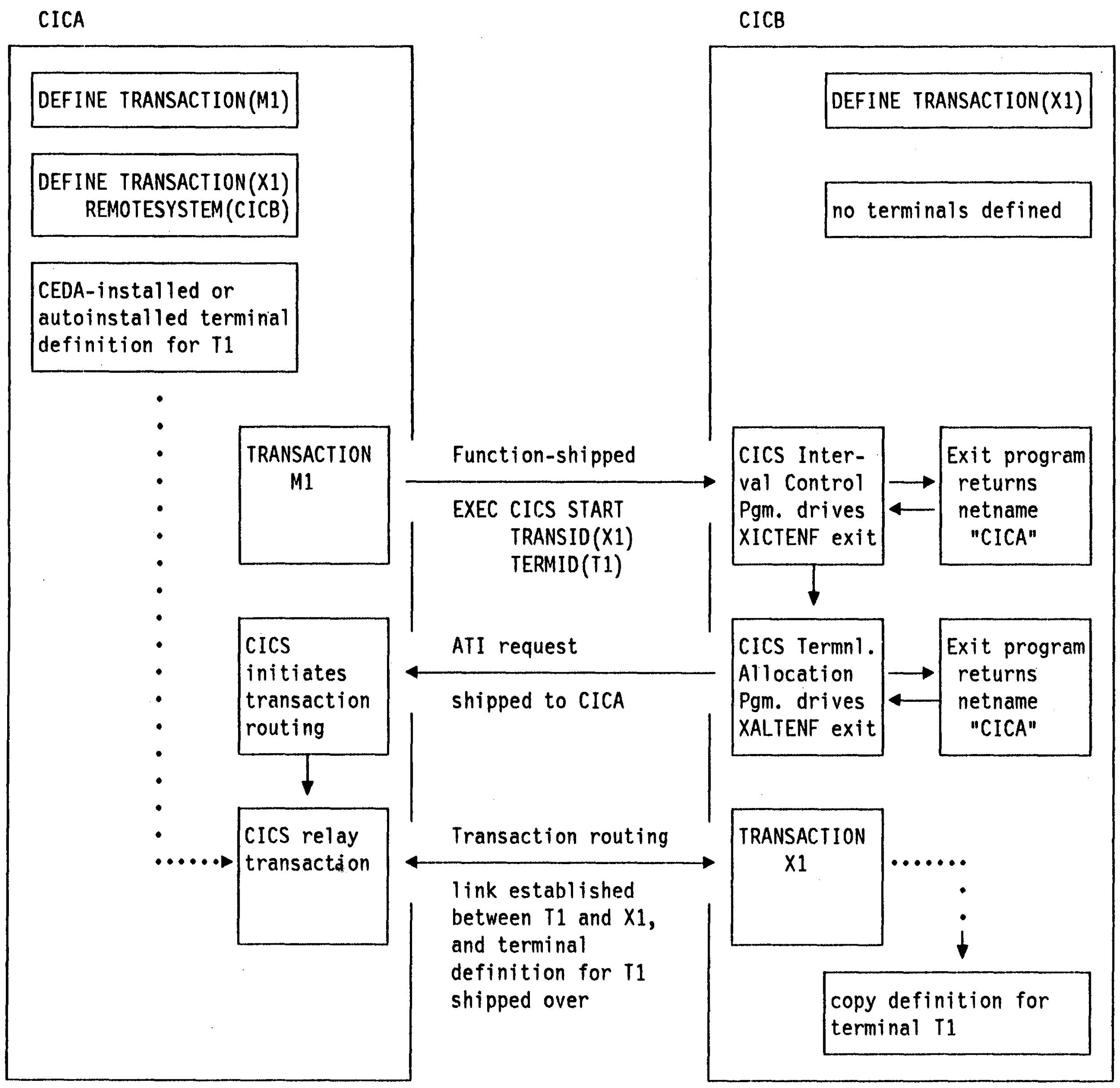

CICA
CICB
DEFINE TRANSACTION(M1)
DEFINE TRANSACTION(X1) REMOTESYSTEM(CICB)
CEDA-installed or autoinstalled terminal definition for T1
DEFINE TRANSACTION(X1)
no terminals defined
TRANSACTION M1
Function-shipped
EXEC CICS START TRANSID(X1) TERMID(T1)
CICS Interval Control Pgm. drives XICTENF exit
Exit program returns netname "CICA"
CICS initiates transaction routing
ATI request
shipped to CICA
CICS Termnl. Allocation Pgm. drives XALTENF exit
Exit program returns netname "CICA"
CICS relay transaction
Transaction routing
link established between T1 and X1, and terminal definition for T1 shipped over
TRANSACTION X1
copy definition for terminal T1

Figure 14. Resolving a terminal-not-known condition on a START request

Resource definition

You do not have to be using autoinstalled terminals to make use of the exits XICTENF and XALTENF. The technique also works with CEDA-installed terminals, if they are defined with SHIPPABLE(YES) specified.

It is important to remember that, although there is no need to have all terminal definitions in place before you operate your network, other definitions are essential. All links between systems must be fully defined, and remote transactions must be known to the systems that want to use them.

The AOR must have a direct link to the TOR. In other words, the *sysid* or *netname* that you pass back to CICS from the exit program must **not** be for an indirectly connected system.

The exit program for the XICTENF and XALTENF exits

How your exit program identifies the TOR from the parameters supplied by CICS can only be decided by reference to your system design. In the simplest case, you would hand back to CICS the *netname* of the system that originated the START request. In a more complex situation, you may decide to give each terminal a name that reflects the system on which it resides.

The *CICS/MVS Customization Guide* has details on how to code the exit program. A sample program is also available.

Basic mapping support

The mapping operations of BMS are performed in the system on which the user's transaction is running. The mapped information is routed between the terminal and this transaction via the relay transaction, as for terminal control operations.

For BMS page building and routing requests, the pages are built and stored in the user transaction's system. When the logical message is complete the pages are shipped to the terminal-owning region (or possibly regions if they were generated by a routing request) and deleted from the user transaction region. Page retrieval requests are processed by a BMS program running in the system to which the terminal is connected.

BMS message routing to remote terminals and operators

You can use the BMS ROUTE command (see the *CICS/MVS Application Programmer's Reference* manual for details of the BMS ROUTE command) to route messages to remote terminals. You cannot, however, route a message to a selected remote operator or operator class unless you also specify the terminal at which the message is to be delivered.

Table 2. BMS message routing to remote terminals and operators

LIST Entry	OPCLASS	Result
None specified	Not specified	The message is routed to all the remote terminals defined in the originating system
Entries specifying a terminal but not an operator	Not specified	The message is routed to the specified remote terminal
Entries specifying a terminal but not an operator	Specified	The message is delivered to the specified remote terminal when an operator with the specified OPCLASS is signed on
None specified	Specified	The message is not delivered to any remote operator
Entries specifying an operator but not a terminal	(Ignored)	The message is not delivered to the remote operator
Entries specifying both a terminal and an operator	(Ignored)	The message is delivered to the specified remote terminal when the specified operator is signed on.

Table 2 shows how the possible combinations of route list entries and OPCLASS operands govern the delivery of routed messages to remote terminals. In all cases, the remote terminal must be defined in the system that issues the ROUTE command (or a shipped terminal definition must already be available; see "Shipping terminal definitions" on page 141). Note that the facility described in "Shipping terminals for automatic transaction initiation" on page 48 does not apply to terminals addressed by the ROUTE command.

The routing transaction (CRTE)

The routing transaction (CRTE) is a CICS-provided transaction that enables a terminal operator to invoke transactions that are owned by a connected CICS system. It differs from normal transaction routing in that the remote transactions do not have to be defined in the local system. However, the terminal through which CRTE is invoked must be defined on the remote system (or defined as "shippable" in the local system), and an entry for the terminal operator is usually required in the remote systems signon table. CRTE can be used from any 3270 display device.

To use CRTE, the terminal operator enters:

```
CRTE SYSID=xxxx[,TRPROF={DFHCICSS|profile-name}]
```

where xxxx is the name of the remote system, as specified in the CONNECTION operand of the DEFINE CONNECTION command (or the SYSIDNT operand of the

DFHTCT TYPE=SYSTEM macro), and profile-name is the name of the profile to be used for the session with the remote system (see "Defining communication profiles" on page 151). The transaction then indicates that a routing session has been established, and the user enters input of the form:

yyyyzzzzzz...

where yyyy is the name by which the required remote transaction is known on the remote system, and zzzzzz... is the initial input to that transaction. Subsequently, the remote transaction can be used as if it had been defined locally and invoked in the ordinary way. All further input is directed to the remote system until the operator terminates the routing session by entering CANCEL.

In secure systems, operators are normally required to sign on before they can invoke transactions. The first transaction that is invoked in a routing session is therefore usually the signon transaction CSSN; that is, the operator signs on to the remote system.

Although the routing transaction is implemented as a pseudoconversational transaction, the terminal from which it is invoked is held by CICS until the routing session is terminated. Any ATI requests that name the terminal are therefore queued until the CANCEL command is issued.

The CRTE facility is particularly useful for invoking a master terminal transaction (CSMT or CEMT) on a particular remote system. It avoids the necessity of defining the remote CSMT or CEMT in the local PCT with a different name. CRTE is also useful for testing remote transactions before final installation.

System programming considerations

You will have to perform the following operations to implement transaction routing in your installation:

1. Install MRO or ISC support, or both, as described in "Part 2. Installation" on page 77.
2. Define MRO or ISC links between the systems that are to be connected, as described in "Chapter 3.1. Defining links to remote systems" on page 91.
3. Define the terminals and transactions that will participate in transaction routing, as described in "Chapter 3.2. Defining remote resources" on page 133.
4. If you want to route to *shippable* terminals from regions where those terminals might be *not known*, code and enable the global user exits XICTENF and XALTENF as described in the *CICS/MVS Customization Guide*.
5. Ensure that the required communication profiles, transactions, and programs are included in the program control table and the processing program table, as described in "Chapter 3.3. Defining local resources" on page 151.

Chapter 1.7. Distributed transaction processing

Like asynchronous processing, which is described in "Chapter 1.5. Asynchronous processing" on page 33, distributed transaction processing (DTP) provides a means of distributing the processing required by an application between two or more systems in an intercommunication environment.

In contrast with asynchronous processing, however, DTP provides **synchronous** communication. In CICS intercommunication, this means that a session is acquired and held by two transactions for the period of a "conversation" between them. Because the transactions have exclusive use of the session, messages that pass between them as part of the conversation can be directly correlated, and each transaction can carry out processing that depends directly on the results of a previous stage of processing performed by the other.

Synchronous communication also enables actions taken by the conversing transactions to be made part of the same logical unit of work. Synchronization points taken by one transaction can force corresponding synchronization points in the other, so that changes made to local and remote resources can be coordinated.

DTP is the most powerful, flexible, and complex of the CICS intercommunication facilities. This chapter introduces the concepts of DTP, "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171 through "Chapter 4.8. CICS-to-IMS applications" on page 263 give detailed descriptions of DTP programming.

Why function shipping and transaction routing are not enough

Function shipping gives you access to remote resources and transaction routing lets a terminal communicate with remote transactions. At first sight, these two facilities may appear sufficient for all your intercommunication needs. Certainly, from a functional point of view, they are probably all you do need. In the real world, however, there are always design criteria that go beyond pure function. Machine loading, response time, continuity of service, and economic use of resources are just some of the factors that affect transaction design.

Consider the following example:

A hypermarket chain has many branches, which are served by several distribution centers, each stocking a different range of goods. Local stock records at the branches are updated online from point-of-sale terminals. Sales information has also to be sorted for the separate distribution centers, and transmitted to them to enable reordering and distribution.

Without thinking too much about it, an analyst might decide to use function shipping to write reorder records to a remote file, at the appropriate distribution center, at the same time that the local stock records are updated by the point-of-sale terminals. This method has the virtue of simplicity, but must be rejected for several reasons:

1. Data is transmitted to the remote systems irregularly in small packets. This means inefficient use of the links.
2. The transactions associated with the point-of-sale devices are competing for sessions to the remote distribution centers. This could mean unacceptable delays at point-of-sale.
3. Failure of a link results in a catastrophic suspension of operations at a branch.
4. Intensive intercommunications activity, for example at peak periods, causes reduction in performance at the terminals.

Now consider the solution where each sales transaction writes its reorder records to a transient data queue. Here the data is quickly disposed of, leaving the transaction to carry on its conversation with the terminal.

It is seldom that restocking requests have any urgency, so that it may be possible to delay the sorting and sending of the data until an off-peak period. Alternatively, the transient data queue could be set up to trigger off the sender transaction when a predefined data level is reached. Either way, the sender transaction has the same job to do.

Again, one might be tempted to use function shipping to transmit the reorder records. After the sort process, each record could be written to a remote file in the relevant remote system. However, this method is not ideal here either. The sender transaction would have to wait after writing each record to make sure that it got the right response. Apart from using the link inefficiently, waiting between records would make the whole process impossibly slow. This chapter tells you how to solve this problem, and others, using distributed transaction processing.

The flexibility of DTP can, in some circumstances, be used to achieve improved performance over function shipping. Consider an example in which you are browsing a remote file to select a record that satisfies some criteria. If you use function shipping, CICS ships the GETNEXT request across the link, and lets the mirror perform the operation and ship the record back to the requestor.

This involves two flows, and a significant amount of data, on what may be a very busy network. If the browse is on a large file, the overhead can be unacceptably high. One alternative is to write a DTP conversation that ships the selection criteria, and returns only the keys and relevant fields from the selected records. This reduces both the number of flows and the amount of data sent over the link, thus reducing the overhead as compared to the function-shipping case.

Why distributed transaction processing?

In a multisystem environment, data transfers between systems are necessary because end users need access to remote resources. In managing these resources, network resources are used. But performance suffers if the network is used excessively. There is therefore a performance gain if application design is oriented toward doing the processing associated with a resource in the resource-owning region.

DTP lets you process data at the point where it arises, instead of overworking network resources by assembling it at a central processing point.

There are, of course, other reasons for using DTP. Here are some of them:

- It allows some measure of parallel processing to shorten response times.
- It provides a common interface to a transaction that is to be attached by several different transactions.
- It enables communication with applications running on other systems, particularly on non-CICS systems.
- It provides a buffer between a security-sensitive file or database and an application, so that no application need know the format of the file records.
- It enables batching of nonurgent data destined for a remote system.

What is a conversation and what makes it necessary?

In DTP, transactions pass data to each other directly. While one sends, the other receives. The exchange of data between two transactions is called a **conversation**. Although several transactions can be involved in a single distributed process, communication between them breaks down into a number of self-contained conversations between pairs. Each such conversation uses a CICS resource known as a **session**.

Conversation initiation and transaction hierarchy

A transaction starts a conversation by requesting the use of a session to a remote system. Having obtained the session, it causes an *attach* request to be sent to the other system to activate the transaction that is to be the conversation partner.

A transaction can initiate any number of other transactions, and hence, conversations. In a complex process, a distinct hierarchy emerges, with the transaction that was started by the terminal at the tip of the pyramid. Figure 15 on page 58 shows a possible configuration. Transaction AAAA is attached over the terminal session. Transaction AAAA attaches transaction BBBB, which, in turn, attaches transactions CCCC and DDDD. Both these transactions attach the same transaction, SUBR, in system CICSE. This gives rise to two different tasks of SUBR.

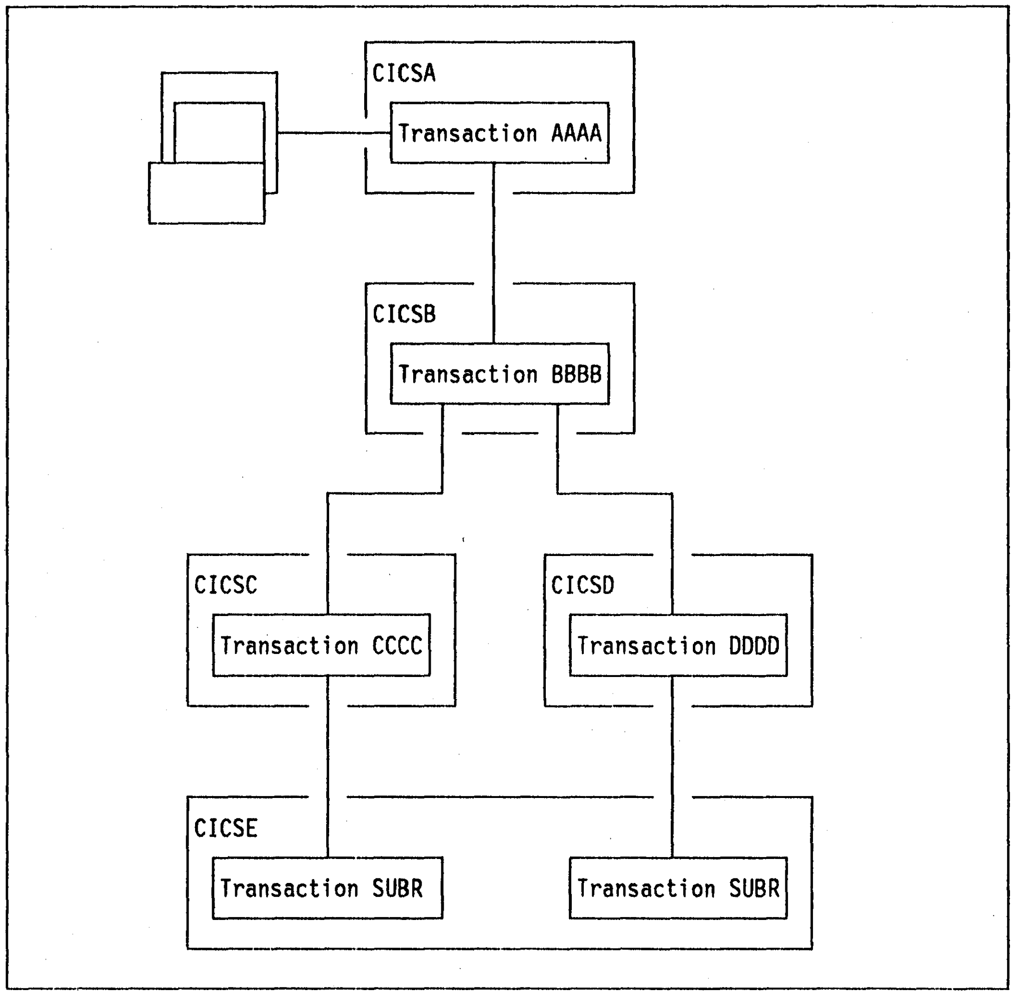


Figure 15. DTP in a multisystem configuration

The structure of a distributed process is determined dynamically by program; it cannot be predefined. Notice that, for every transaction, there is only one inbound attach request, but that there can be any number of outbound attach requests. The session that activates a transaction is called its **principal facility**. A session that is allocated by a transaction to activate another transaction is called its **alternate facility**. Therefore, a transaction can have only one principal facility, but any number of alternate facilities.

When a transaction initiates a conversation, it is the **front end** on that conversation. Its conversation partner is the **back end** on the same conversation. It is normally the front end that dominates, and determines the way the conversation goes. You can arrange for the back end to take over if you want, but in a complex process, this can cause unnecessary complications. This is further explained in the discussion on synchronization later in this chapter.

Dialog

A conversation transfers data from one transaction to another. For this to function properly, each transaction must "know" what the other intends. It is therefore necessary to design, code, and test front end and back end as one software unit. The same applies when there are several conversations and several transaction programs. Each new conversation adds to the complexity of the overall design.

In the example on page 55, the DTP solution is to transmit the contents of the transient data queue from the front end to the back end. The front end issues a SEND command for each record that it takes off the queue. The back end issues RECEIVE commands until it receives an indication that the transmission has ended.

In practice, most conversations simply transfer a file of data from one transaction to another. The next stage of complexity is to cause the back end to return data to the front end, perhaps the result of some processing. Here the front end is programmed to request conversation turnaround at the appropriate point.

Control flows and brackets

During a conversation, data passes over the link in both directions. A single transmission is called a **flow**. Issuing a SEND command does not always cause a flow. This is because the transmission of user data can be deferred; that is, held in a buffer until some event takes place. The LUTYPE6.2 architecture defines data formats and packaging. CICS handles these things for you, and they concern you only if you need to trace flows for debugging purposes.

The LUTYPE6.2 architecture defines a data header for each transmission, which holds information about the purpose and structure of the data following. The header also contains bit indicators to convey control information to the other side. For example, if one side wants to tell the other that it can start sending, CICS sets a bit in the header that signals a change of direction in the conversation.

To keep flows to a minimum, nonurgent control indicators are accumulated until it is necessary to send user data, when they are added to the header.

The LUTYPE6.2 architecture defines acronyms for all control indicators. Here are the ones you will meet in this book:

Table 3. LUTYPE6.2 control indicators — sample

Acronym	Name	Meaning
BB	BEGIN_BRACKET	Start a conversation.
BIND	BIND_SESSION	Request session bind negotiation.
CD	CHANGE_DIRECTION	Receiver can now send.
CEB	CONDITIONAL_END_BRACKET	End conversation.
CNOS	CHANGE_NUMBER_OF_SESSIONS	Renegotiate change to number and character of available sessions.
CTD	COMMITTED	Recoverable resources have been committed.
FGT	FORGET	Syncpointing activity on this transaction is complete.
PTC	PREPARE_TO_COMMIT	Start of syncpointing activity.
RB	ROLL_BACK	A transaction within the distributed process wants to return recoverable resources to the state they were in at the last actual or implied syncpoint.
RC	REQUEST_COMMIT	This transaction is ready to commit its recoverable resources.

In complex procedures, such as establishing syncpoints, it is often necessary to send control indicators when there is no user data available to send. This is called a **control flow**.

BEGIN_BRACKET marks the start of a conversation; that is, when a transaction is attached. CONDITIONAL_END_BRACKET ends a conversation. End bracket is conditional because the conversation can be reopened under some circumstances. A conversation is **in bracket** when it is still active.

For DTP, MRO is not unlike LUTYPE6.2 in its internal organization. In fact, it is based on LUTYPE6.1, which is also a SNA-defined architecture.

Conversation state and error detection

As a conversation progresses, it moves from one state to another within both conversing transactions. The conversation state determines the commands that may be issued. For example, it is no use trying to send or receive data if there is no session linking the front end to the back end. Similarly, if the back end signals end of conversation and the front end has seen the end bracket, the front end cannot be in a state to receive more data.

Either end of the conversation can cause a change of state, usually by issuing a particular command from a particular state. CICS tracks these changes, and stops transactions from issuing the wrong command in the wrong state.

Synchronization

There are many things that can go wrong during the running of a transaction. The conversation protocol helps you to recover from errors and ensures that the two sides remain in step with each other. This use of the protocol is called **synchronization**.

Synchronization allows you to protect resources such as transient data queues and files. Whatever goes wrong during the running of a transaction should not leave the associated resources in an incorrect state.

Examples of use

Suppose, for example, that a transaction is transmitting a queue of data to another system to be written to a DASD file. Suppose also that for some reason, not necessarily connected with the intercommunications activity, the receiving transaction is abended. Even if a further abend can be prevented, there is the problem of how to continue the process without loss of data. It is uncertain how many queue items have been received and how many have been correctly written to the DASD file. The only safe way of continuing is to go back to a point where you know that the contents of the queue are consistent with the contents of the file. However, you then have the problem of restoring the queue entries on the one side, and of deleting the corresponding entries in the DASD file on the other side.

The cancelation by an application program of all changes to recoverable resources since the last known consistent state is called **rollback**. The physical process of recovering resources is called **backout**. The condition that exists as long as there is no loss of consistency between distributed resources is called **data integrity**.

There are cases where you may want to recover resources, even though there are no error conditions. Consider an order entry system. While entering an order for a customer, an operator is told by the system that the customer's credit limit would be exceeded if the order went through. Because there is no use continuing until the customer is consulted, the operator presses a PF key to abandon the order. The transaction is programmed to respond by returning the data resources to the state they were in at the start of the order.

Taking syncpoints

If you were to log your own data movements, it would be possible for you to arrange backout of your files and queues. However, it would involve some very complex programming, which you would have to repeat for every similar application. To save you the trouble, CICS arranges resource recovery for you. Logical unit management works together with resource management in ensuring that resources can be restored.

The points in the process where resources are declared to be in a known consistent state are called **synchronization points**, often shortened to **syncpoints**. Synchronization points are implied at the beginning and end of a transaction. A transaction can define other syncpoints by program command. All processing between two syncpoints belongs to a **logical unit of work** (LUW).

Taking a syncpoint **commits** all recoverable resources. This means that all systems involved in a distributed process erase all the information they have been keeping about data movements on recoverable resources. Now backout is no longer possible, and all changes to the resources since the last synchronization point are made irreversible.

Although CICS commits and backs out resources for you, the service must be paid for in performance. You might have transactions that do not need such sophistication, and it would be wasteful to employ it. If the recovery of resources is not a problem, you can use simpler methods of synchronization.

The three synchronization levels

The LUTYPE6.2 architecture defines three levels of synchronization:

- Level 0 — NONE
- Level 1 — CONFIRM
- Level 2 — SYNCPOINT

At sync level 0, there is no system support for synchronization. It is nevertheless possible to achieve some degree of synchronization through the interchange of data, using the SEND and RECEIVE commands.

If you select sync level 1, you can use special commands for communication between the two conversation partners. One transaction can *confirm* the continued presence and readiness of the other. The user is responsible for preserving the data integrity of recoverable resources.

The level of synchronization described earlier in this section corresponds to sync level 2. Here system support is available for maintaining the data integrity of recoverable resources.

CICS implies a syncpoint when it starts a transaction; that is, it initiates logging of changes to recoverable resources, but no control flows take place. CICS takes a full syncpoint when a transaction is normally terminated. The abending of a transaction causes rollback. The transactions themselves can initiate syncpoint or rollback requests. However, a syncpoint or rollback request is propagated to another transaction only when the originating transaction is in conversation with the other transaction, and sync level 2 has been selected for the session between them.

Remember that syncpoint and rollback are not peculiar to any one conversation within a transaction. They are propagated on every conversation that is currently *in bracket*.

MRO or LUTYPE6.2?

You can program DTP applications for both MRO and LUTYPE6.2 links. The two conversation protocols are **not** identical. Although you seldom have the choice for a particular application, an awareness of the differences and similarities will help you to make decisions about compatibility and migration.

You must use LUTYPE6.2 for communication between systems in different MVS images. If you want to set up a link between two CICS systems in the same MVS image, you can use MRO, LUTYPE6.2, or both. However, MRO always has a performance advantage over LUTYPE6.2.

Table 4 points out the main differences between the two protocols.

Table 4. MRO compared with LUTYPE6.2

MRO	LUTYPE6.2
Function realized within CICS.	Depends on VTAM or similar.
Non-standard architecture	SNA architecture
CICS to CICS links only	Links to non-CICS systems possible
Runs within single MVS image.	Communicates across multiple MVS images.
Sync level 2 forced for the conversation.	Sync level 0, 1, or 2 can be selected.
PIP data not supported.	PIP data supported.
Data transmission not deferred.	Deferred data transmission.
Partner transaction identified in data.	Partner transaction defined by CONNECT PROCESS command.
RECEIVE can only be issued in receive state.	RECEIVE causes conversation turnaround when issued in send state on mapped conversations.
No expedited flow possible.	ISSUE SIGNAL command enables expedited flow.
WAIT command has no function.	WAIT command causes transmission of deferred data.

LUTYPE6.2 mapped or basic?

APPC conversations can either be **mapped** or **basic**. If you are interested in CICS-to-CICS applications, you need only use mapped conversations. Basic conversations (also referred to as "unmapped") are useful only when communicating with systems that do not support mapped conversations. These include some APPC devices.

The two protocols are similar. The main difference lies in the way user data is formatted for transmission to the other side. In mapped conversations, you merely send the data you want your partner to receive. In basic conversations, you have to add a few control bytes to convert the data into an SNA-defined format called a **generalized data stream** (GDS). You also have to include the keyword GDS on EXEC CICS commands for basic conversations.

Table 5 summarizes the differences between mapped and basic conversations.

Table 5. LUTYPE6.2 conversations – mapped compared with basic

Mapped	**Basic**
The conversation partners only exchange data that is relevant to the application.	Both partners must package the user data before sending and unpackage it on receipt.
All conversations for a transaction share the same EXEC Interface Block for status reporting.	Each conversation has its own area for state information.
The transaction can *handle* exceptional conditions or let them default.	The transaction must test for exceptional conditions in a data area set aside for the purpose.
A RECEIVE command issued in send state causes conversation turnaround.	A RECEIVE command is illegal in send state.
Transactions can be written in any of the supported languages.	Transactions can be written only in Assembler language.

Availability of DTP facilities

CICS DTP facilities are provided through the command-level interface of the CICS Terminal Control program. No special DTP facilities are provided at the macro level. Application programs can be written in COBOL, PL/I, or assembler language.

For CICS-CICS communication, DTP can be used between any CICS systems coupled by MRO, LUTYPE6.1, or LUTYPE6.2 links.

For CICS-IMS communication, DTP can be used between CICS and some types of IMS transaction (for example, RESPONSE MODE transactions), but only when CICS is the front end. When IMS is the front end, it always uses asynchronous processing to initiate CICS transactions. Communication between CICS and IMS is possible only on LUTYPE6.1 links.

Design concepts

Overview of the application programming interface

This section provides an overview of the application programming facilities provided for distributed transaction processing, and of some of the basic protocols associated with DTP conversations. Fuller information is given in "Part 4. Application programming" on page 159.

Acquiring a session to the remote system

A front-end transaction acquires a session to a remote system by executing an ALLOCATE command. Normally, only the name of the remote system is specified on the command, and CICS selects an available session for the transaction. The name of the session, or conversation, is then made available to the transaction.

Specific sessions can also be named for communication on LUTYPE6.1 links, but this is not normally necessary. System names, rather than session names, **must** be used for session allocation on MRO or LUTYPE6.2 links.

Facilities exist for the transaction to specify what action is required if a session is not available; either to wait until a session is available or to continue processing.

The PROFILE option can be used on the ALLOCATE command to specify a set of terminal control processing options. Profiles are generated by the system programmer in the Program Control Table. They determine such factors as whether an FMH received from the other system is to be included in the application program's input data area, and whether automatic journaling is to be used. For LUTYPE6.2 links, they can also specify the modeset name of a group of LUTYPE6.2 sessions, thereby enabling a particular class of service to be selected for the session.

Depending upon the circumstances, CICS sometimes ignores profile specifications. For example, INBFMH is always used for MRO sessions. Also, LUTYPE6.2 FMHs are never passed to application programs, regardless of the profile specification. Further information on profiles is given in "Defining communication profiles" on page 151.

Initiating the back-end transaction

After the session has been allocated, the front-end transaction can initiate the back-end transaction.

For CICS-CICS communication on MRO or LUTYPE6.1 links, this is done by the first SEND or CONVERSE command issued by the front-end transaction. The name of the back-end transaction is sent either as the first four bytes of the first message to the remote system, or as part of an SNA-defined field, called an "attach" function management header (FMH), that is transmitted with the first message. A BUILD ATTACH command exists to enable the transaction to build an LUTYPE6.1 attach FMH.

Because IMS does not use the convention of a transaction name in the first four bytes of data, an attach FMH must always be built to initiate CICS-IMS DTP.

For LUTYPE6.2 sessions, CICS provides a special command, CONNECT PROCESS, which is used before the first SEND or CONVERSE command, and which builds an LUTYPE6.2 FMH to initiate the back-end transaction. All LUTYPE6.2 FMHs are handled by CICS; a CICS transaction never has to build them, nor can it receive them.

To assist in migrating DTP applications from LUTYPE6.1 to LUTYPE6.2 links, CICS will accept a transaction name in the first four bytes of data and build a suitable LUTYPE6.2 attach header. CONNECT PROCESS should, however, always be used for new applications.

The conversation

The front-end and back-end transactions communicate by executing SEND, RECEIVE, or CONVERSE commands. The front-end transaction must always specify the name of the session on these commands. This is not necessary for the back-end transaction, because it is communicating with its principal facility.

The MRO or ISC session between the transactions follows the SNA half-duplex flip-flop protocol. This means that at no time can both transactions attempt to send messages together. One transaction must be in send state while the other is in receive state.

The front-end transaction is always in send state when it acquires the session, and the back-end transaction is always in receive state when it is initiated.

A change of state is usually initiated by the transaction currently in send state. It can do so by executing a CONVERSE or a SEND INVITE command. Both of these commands invite the transaction that is in receive state to send. (In an LUTYPE6.2 conversation, a partner in receive state can initiate a change of state with an ISSUE ERROR command, as explained below).

On ISC sessions, a transaction in receive state can request the one in send state to reverse the states by executing an ISSUE SIGNAL command. This causes CICS to transmit an SNA SIGNAL command, which carries an SNA code that means "request change-direction", to the other transaction. CICS indicates receipt of the command by raising the SIGNAL condition in the send-state transaction. A transaction can change to receive state (by issuing a SEND INVITE command) following receipt of the signal command, though it is not an error not to do so. ISSUE SIGNAL is not supported on MRO sessions.

For LUTYPE6.2 sessions, there are two additional commands that enable either transaction to inform the other that an error has occurred, irrespective of whether they are in send or receive state. They are ISSUE ERROR and ISSUE ABEND.

ISSUE ERROR causes EIBERR to be set in the transaction that receives it. If this command is issued by a transaction that is in receive state, the send/receive states of the two transaction are reversed. This enables the issuing transaction to send additional information about the error.

On mapped conversations, ISSUE ABEND causes the conversation to abend. A CICS transaction that issues abend can free the session and continue normally. A CICS transaction that receives a conversation abend can also continue normally provided that it is designed to handle the TERMERR condition. The default action for TERMERR is to abend the transaction.

Synchronization points

On MRO or LUTYPE6.1 sessions, either transaction can initiate a synchronization point when it is in send state, by issuing a SYNCPOINT command. This causes an indication to be sent to the other transaction to specify that it too must take a syncpoint.

On LUTYPE6.2 sessions, the way in which a synchronization point can be taken is governed by the synchronization level that is established for the conversation.

For level 0 conversations, no synchronization is possible.

For level 1 conversations, the commands SEND CONFIRM and ISSUE CONFIRMATION (together with ISSUE ERROR or ISSUE ABEND) are available to enable the transactions to exchange private synchronization protocols. CICS syncpointing mechanisms are not involved in these exchanges. This is the maximum synchronization level permitted for LUTYPE6.2 single-session links.

For level 2 conversations, full CICS syncpointing is available, including SYNCPOINT ROLLBACK. The level 1 commands SEND CONFIRM and ISSUE CONFIRMATION can also be used on level 2 conversations.

The maximum synchronization level available on an LUTYPE6.2 session is dependent on the capabilities of the communication system, which is determined when the session is bound. **All** sessions to any particular system are therefore bound at the same maximum synchronization level. The synchronization level for a specific conversation (which cannot exceed the maximum allowed for the session) must be specified on the CONNECT PROCESS command.

CICS always uses synchronization level 2 for transactions that have been migrated from LUTYPE6.1 but have not been modified to include a CONNECT PROCESS command. For this reason, it is not possible to run these applications over an LUTYPE6.2 single-session link.

Freeing the session

A session is explicitly freed by a FREE command that specifies the session name. The transaction that is in send state must issue the command; an indication is then sent to the other transaction to specify that it should now free the session.

A session is implicitly freed when one of the transactions ends, provided that it is valid for the transaction to free the session at that time.

SNA considerations

The information given in "Part 4. Application programming" on page 159 will enable you to construct valid command sequences for DTP applications. However, an understanding of the SNA protocols and corresponding data flow control indicators used by CICS for DTP, and their relationship with CICS commands and command options, will enable you to understand why the rules are necessary, and can help you to design efficient and error-free applications.

Although MRO sessions do not use the services of an SNA access method, they do, at the user level, employ SNA formats and protocols. With some exceptions, which are pointed out in the text, the following sections apply equally to MRO sessions.

With the exception of certain commands that can cause messages to be transmitted "against the flow" (such as ISSUE SIGNAL) the session and transaction states in DTP are dictated by indicators sent by whichever transaction is currently in send state. In the receiving transaction, the arrival of a particular indicator causes an appropriate field to be set in the EIB.

The SNA indicators of direct concern are:

end-bracket
: The beginning and the end of a conversation between two transactions are indicated by begin-bracket and end-bracket indicators. A conversation is, in other words, an SNA bracket.

  Only end-bracket need be considered. The bracket is begun automatically when the back-end transaction is initiated by one of the methods described previously.

change-direction
: In SNA half-duplex flip-flop protocol, the change-direction indicator is sent by the send-state transaction to reverse the direction of flow on the session. It causes the send-state transaction to switch to receive state and the receive-state transaction to switch to send state.

syncpoint-request
: A syncpoint-request indicator is sent on the session to indicate that the send-state transaction is taking a syncpoint and that the receive-state transaction must also take a syncpoint.

To understand the flows of these indicators on the session, you must consider two aspects of the CICS implementation of DTP:

1. Under what circumstances the indicators are generated ready for sending
2. Under what circumstances the indicators are actually transmitted.

How SNA Indicators are generated

The change-direction, syncpoint-request, and end-bracket indicators can be generated:

- Explicitly as a result of a CICS command or command option
- Automatically by CICS because it detects that one is needed.

The change direction indicator changes the issuing transaction from send state to receive state, and the other transaction from receive state to send state. It is generated explicitly by one of the following:

A SEND command with the INVITE option
A CONVERSE command.

On ISC sessions, CICS will supply the missing change direction indicator if you use a SEND command (without INVITE) followed by a RECEIVE command. On MRO sessions, however, you must use either CONVERSE or SEND INVITE.

A syncpoint-request indicator is generated explicitly by a CICS SYNCPOINT command, or automatically at task termination, provided that the session is still in bracket-state. The session is in bracket state if the end-bracket indicator has not been transmitted, even if it has already been generated.

An end-bracket indicator is generated by one of the following:

A SEND command with the LAST option

A SEND command followed by a FREE command

A SEND command followed by termination of the task

A RECEIVE command which causes EIBRECV to be turned off (X'00') followed by a FREE command or by termination of the task.

When SNA indicators are transmitted

To optimize the use of ISC sessions, CICS implements deferred output processing for SEND commands. This means that a command is not sent across the link until either an internal buffer becomes full, or CICS knows that the conversation is turning around. A consequence of this is that application programs ***must not make any assumptions*** about the physical data flows between the partners in a conversation. If such assumptions are made, the transaction does not work correctly when environmental conditions alter the pattern of physical data flows.

Deferred output often enables CICS to add SNA indicators to waiting data before it is transmitted, and the number of transmissions required on the session is thereby reduced. The addition of indicators to deferred output is sometimes called **piggy-backing**.

For LUTYPE6.2 sessions, further optimization is achieved by accumulating as much data as possible in an internal CICS buffer before actually transmitting it across the link. Thus the data from a series of SEND commands is transmitted only when the buffer becomes full or when the transmission must be forced (for example, if SEND WAIT is encountered). This additional optimization does not affect the number of flows that are "seen" at the application programming interface; LUTYPE6.1 and LUTYPE6.2 are equivalent in this respect.

Deferred output is not implemented for MRO sessions. This can lead to a difference between the number of transmissions required on an ISC link and the number required on an MRO link when the same command sequence is executed. This in turn leads to a difference in the number of RECEIVE commands that the receiving transaction must issue to receive both the data and the indicators.

Important

You must not make any assumptions as to the state of a DTP conversation. You must ***always*** test the EIB flags after each command to determine the current state, and act accordingly.

A further difference between ISC and MRO links caused by deferred output is that some command sequences that are valid for ISC sessions are invalid for MRO sessions.

As an example, consider the following command sequence:

```
EXEC CICS SEND FROM(data-area) INVITE
EXEC CICS SYNCPOINT
EXEC CICS RECEIVE INTO(data-area)
```

On ISC links, the INVITE option generates the change-direction indicator, but the sending of the message is deferred. The transaction is therefore still in send state, and the following SYNCPOINT command is valid. CICS adds the syncpoint-indicator to the deferred output. The output data, the change-direction indicator, and the syncpoint-request are sent in a single transmission.

On MRO links, because there is no deferred output, the output data and the change-direction indicator are sent immediately. The transaction changes to receive state, and the following SYNCPOINT command is invalid.

Output is not deferred if the SEND command has the WAIT option; the message is transmitted immediately. Use of the WAIT option therefore removes one of the implementation differences between ISC and MRO distributed transaction processing. However, on ISC links, it can lower efficiency by increasing the number of flows that are required.

The WAIT option can sometimes be used to allow input and output on the session to be overlapped with processing in the transaction. The following sequence shows how this can be done:

```
EXEC CICS SEND FROM(data-area) INVITE WAIT
.
(Non-CICS programming statements)
.
EXEC CICS RECEIVE INTO(data-area)
```

Because of the WAIT option, execution of the non-CICS programming statements will not begin until execution of the SEND is complete. However, transmission of the input data from the remote system can overlap the processing of these statements. Without the WAIT option, execution of the SEND would not start until processing reached the RECEIVE statement, so no overlapping would be possible.

The WAIT option can also be used to force end-bracket to flow on the session, and so prevent the session from being involved in syncpointing activity. In the command sequence:

```
EXEC CICS SEND FROM(data-area) LAST WAIT
EXEC CICS RETURN
```

the LAST option causes end-bracket to be generated and the WAIT option causes it to be transmitted. The session is therefore not involved in the implicit syncpoint caused by the RETURN statement.

Design hints

The two transactions involved in a DTP conversation should, if possible, be designed as a *requester* or main routine and a *server* or subroutine. In general, the front-end transaction will be the requester, and the back-end transaction the server.

The logic should be contained as far as possible in the requester. This transaction should pass requests to the server only as necessary. The server should return its response to the requester, and then wait for another request. Attempts to distribute the logic between the two transactions, thus making them into peers, are likely to lead to complex design problems. If the roles of the two systems need to be reversed, it is generally preferable for an independent second pair of transactions to be invoked, rather than for the requester and server roles of a single pair of transactions to be reversed.

Programming example

As an example of distributed program design, consider the LUTYPE6.2 mapped conversation shown in Figure 16.

```
Front-End Transaction   |              | Back-End Transaction
                        |              |
EXEC CICS ALLOCATE      |              |
                        |              |
EXEC CICS CONNECT       |              |
 PROCESS SYNCLEVEL(2)   |              |
                        |              |
EXEC CICS CONVERSE      |  ----------> | EXEC CICS RECEIVE
                        |              |
                        |  <---------- | EXEC CICS SEND LAST
                        |              |
EXEC CICS FREE          |              | EXEC CICS RETURN
                        |              |
(transaction abends     |              |
 because syncpoint      |              |
 request is ignored)    |              |
```

Figure 16. An incorrect LUTYPE6.2 mapped conversation

In this figure, the front-end transaction:

1. Allocates a session.
2. Uses a CONNECT PROCESS command to initiate the back-end transaction at synchronization-level 2.
3. Uses a CONVERSE command to:
   a. Send a message to the back-end transaction.
   b. Receive a reply.
4. Frees the session.

This apparently simple conversation fails because the back-end transaction's SEND LAST command does not flow until the RETURN statement is executed.

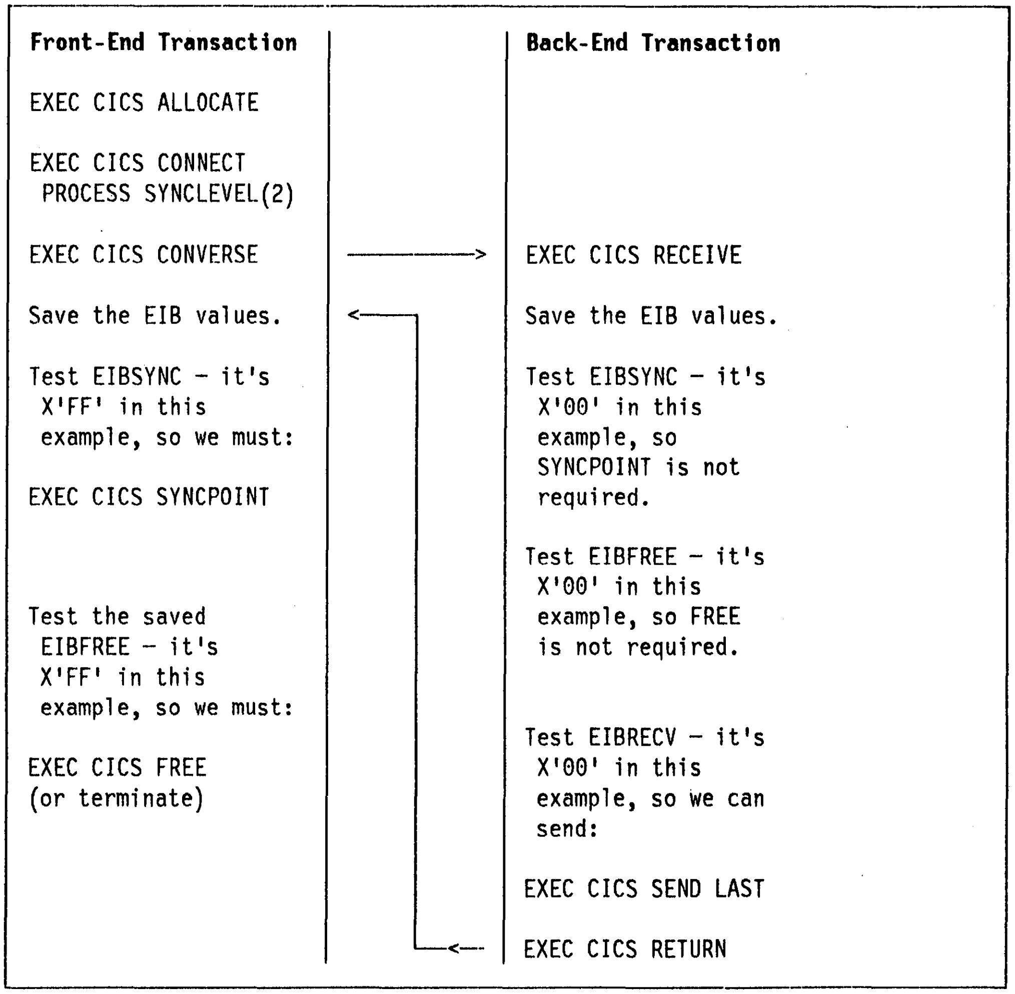
The RETURN statement causes an implicit syncpoint, which goes with the end-bracket indicator. The front-end transaction ignores the syncpoint request, and abends.

You can correct this conversation in a number of ways:

1. By coding SEND LAST WAIT instead of SEND LAST in the back-end transaction. This forces LAST to flow, so that no syncpointing takes place on the session.
2. By understanding that syncpoint-request will be received, and coding EXEC CICS SYNCPOINT before the FREE command in the front-end transaction. This approach is not recommended.
3. By examining the EIB values and taking the appropriate action. This method allows all the possibilities to be catered for, and does not rely on a detailed knowledge of the ways in which the flows are generated.

The previous example can be modified to include EIB testing as shown in Figure 17 on page 73.

More, or fewer, EIB tests than those shown may be needed, depending on the type of session that is being used and possibly on what the other transaction is designed to send. To determine what tests are required, refer to the appropriate chapter in "Part 4. Application programming" on page 159.



```
Front-End Transaction          |               | Back-End Transaction
                               |               |
EXEC CICS ALLOCATE             |               |
                               |               |
EXEC CICS CONNECT              |               |
 PROCESS SYNCLEVEL(2)          |               |
                               |               |
EXEC CICS CONVERSE             |  ---------->  | EXEC CICS RECEIVE
                               |               |
Save the EIB values.           |  <------+     | Save the EIB values.
                               |         |     |
Test EIBSYNC - it's            |         |     | Test EIBSYNC - it's
 X'FF' in this                 |         |     |  X'00' in this
 example, so we must:          |         |     |  example, so
                               |         |     |  SYNCPOINT is not
EXEC CICS SYNCPOINT            |         |     |  required.
                               |         |     |
                               |         |     | Test EIBFREE - it's
                               |         |     |  X'00' in this
Test the saved                 |         |     |  example, so FREE
 EIBFREE - it's                |         |     |  is not required.
 X'FF' in this                 |         |     |
 example, so we must:          |         |     |
                               |         |     | Test EIBRECV - it's
EXEC CICS FREE                 |         |     |  X'00' in this
(or terminate)                 |         |     |  example, so we can
                               |         |     |  send:
                               |         |     |
                               |         |     | EXEC CICS SEND LAST
                               |         |     |
                               |         +<--- | EXEC CICS RETURN
```

Figure 17. The corrected LUTYPE6.2 mapped conversation

Multiple LU type 6 sessions

A transaction may initiate several transactions in other systems. The design of applications using such transactions is likely to be very complex unless it is highly structured. The least complicated design will probably be a requester/server tree, in which each transaction acts as the requester for all transactions for which it is the originating node, and the first transaction to be initiated is the requester for the whole tree.

The requester/server concept is particularly important in relation to synchronization points in a tree of transactions. Unless these are originated by the transaction that is the requester for the whole tree, they are unlikely to be successful. They are originated at the top of the tree, then propagated down the whole tree.

Any transaction issuing a synchronization request must be in send state with respect to all its LU Type 6 sessions, except those for which either a synchronization point has been requested (EIBSYNC set) or which have been freed by the transaction at the other end of the session (EIBFREE set). If an attempt is made to take a synchronization point when these conditions do not hold, the transaction making the attempt will be abended, which will lead to all transactions in the tree being abended.

Advanced program-to-program communication (APPC) supports both the requester/server and peer/peer application models. Each model requires its own design to handle data flows and syncpointing. Individual transaction design should take account of the overall network design. If each program tests the EIB flags, and always reacts correctly to the current state of the conversation, results should be accurate.

Queue transfer

The following special considerations apply to transactions intended to transfer queues of data between systems.

- The sending transaction should be designed as the requester and the receiving transaction as the server.
- For simplicity of design, a separate pair of transactions should be used for each queue to be transferred.
- The requester transaction should not send large numbers of records without first obtaining confirmation from the server transaction that it is attached and is processing the records successfully (perhaps by using the CONFIRM option for LUTYPE6.2 conversations).
- For large queues, take synchronization points at regular intervals, unless performance is a crucial consideration.
- For large queues, use the pacing facilities of VTAM to avoid flooding the network. Alternatively, use frequent synchronization points, which have a similar effect to pacing.
- It will probably be advisable for the receiving transaction to store the incoming data on either a transient data or a temporary storage queue, rather than to update permanent storage as the records are received. The design will be simpler, because the updating will be a separate operation and because error recovery will be easier.
- Consider how transmission should be restarted if an error occurs when the queue has been partly transmitted. There is a basic choice to be made between continuing from the point at which the error occurred and retransmitting the whole queue.

CICS to non-CICS systems

CICS can communicate with transactions running in other types of system, provided they implement a suitable subset of the SNA LU Type 6 protocols. This includes IMS. It is necessary, however, for the designer of such applications to understand in detail the SNA data flow control commands and protocols generated by the other system.

In some cases CICS transactions converse with the remote system, rather than with user written transactions running in that system. CICS transactions converse with the DC component of IMS, for example, so the protocols and data formats of that component must be understood and complied with.

Other systems may allow direct communication with their transactions. It is then necessary to know the protocols generated by user-written code in the transactions. In particular, it is necessary to know how BB (begin bracket), EB (end bracket) and CD (change direction) indicators are generated and responded to.

In any case, the following problems will need attention during the design of the application.

- How the required transaction is to be attached in the remote system. It may be necessary to send an attach header, in which case the remote transaction could have a name up to 8 characters long.
- The structure of the messages passing between the local and remote transactions, and how any mapping component of the remote system is to be used.
- The possible replies to each type of request, together with the SNA indicators that may need to be present on the request and replies.
- Ensuring that the SNA indicators are followed precisely by the transactions at both ends of a session.
- Which transaction or transactions may end a conversation.
- Whether synchronization points are to be used, and if so, whether they are to be on single or multiple sessions. The remote system may support only single session synchronization points.

Part 2. Installation

This part of the intercommunication guide discusses the installation requirements for a CICS system that is to participate in intersystem communication or multiregion operation. You should be familiar with the general requirements for CICS installation, which are described in the *CICS/MVS Installation Guide*. You may also have to refer to the *CICS/MVS Resource Definition (Macro)* manual for information on coding the CICS system initialization table (SIT) and to the *CICS/MVS Customization Guide* for information on coding CICS system generation macros.

"Chapter 2.1. Installation considerations for multiregion operation" on page 79 describes the CICS installation requirements for multiregion operation.

"Chapter 2.2. Installation considerations for intersystem communication" on page 81 describes the CICS installation requirements for CICS intersystem communication. It also contains notes on the installation requirements of ACF/VTAM and IMS when these products are to be used with CICS in an intersystem communication environment.

Chapter 2.1. Installation considerations for multiregion operation

This chapter discusses those aspects of installation that apply particularly to CICS multiregion operation.

To use CICS MRO, you must:

1. Install the CICS Type 2 SVC
2. Define CICS as an MVS subsystem
3. Ensure that the required CICS modules are included in your CICS system
4. Place some modules in the LPA.

Installing the CICS type 2 SVC routine

Multiregion operation requires the CICS interregion communication modules to run in supervisor state to transfer data between different regions.

CICS uses a normal supervisor call to a bootstrap SVC routine which is supplied on the pregenerated system load library (CICS212.LOADLIB) under the name DFHCSVC. You will have to link edit this routine into your system nucleus. Information on how to do this is given in the *CICS/MVS Installation Guide*.

The number of the supplied SVC is 216. You can change the number of the SVC if required; details are given in the *CICS/MVS Installation Guide*.

Adding CICS as an MVS subsystem

Multiregion operation with CICS/MVS requires OS/VS Subsystem Interface (SSI) support. You must therefore install CICS as an MVS subsystem. The procedure for doing this is detailed in the *CICS/MVS Installation Guide*.

Modules required for MRO

The standard pregenerated system supplied on the CICS distribution volume includes a pregenerated version of each of the CICS management modules required to support multiregion operation.

You must include the following management programs in your system (by using the SIT or startup overrides):

- The EXEC interface programs. (Specify EXEC=YES or allow it to default.)

  These programs are not required if you have no command-level application programs. The only MRO facility available in this case is transaction routing.
- The intersystem communication programs. (Specify ISC=YES.)
- A terminal control program generated by DFHSG PROGRAM=TCP.

All versions of TCP contain support for interregion communication. The other requirements for the terminal control program will depend upon your total installation requirements. See the *CICS/MVS Customization Guide* for information on how to generate a suitable version, and the *CICS/MVS Resource Definition (Macro)* manual for information on coding the SIT operands.

- The system recovery program. (specify SRT=YES or SRT=xx, where xx is the suffix of your system recovery table.)

Refer to the *CICS/MVS Installation Guide* to obtain the suffixes, if any, of the pregenerated versions of these programs.

MRO modules in the link pack area

For multiregion operation, there are certain modules which, for integrity reasons, must be resident in the shared area or loaded into protected storage.

You must place the following module in link pack area (LPA) of MVS.

- DFHIRP — the CICS Interregion Communication Program.

Module DFHCRC, the interregion communication ESTAE exit module, can be placed in the LPA to enable it to be shared by CICS address spaces. However, if it is not in the LPA, it is always loaded into protected storage in the CICS address space.

Logging on to the IRC access method

Before a CICS system can use the MRO facilities it must "log on" to the IRC access method. You can specify that CICS is to log on when it is initialized by coding IRCSTRT=YES in the SIT or the startup overrides. If this is not done, the CEMT SET IRC OPEN command must be used to effect the log on.

Chapter 2.2. Installation considerations for intersystem communication

This chapter discusses those aspects of installation that apply particularly when CICS is used in an intersystem communication environment. It also contains notes on the installation requirements of ACF/VTAM and IMS when these products are to be used with CICS in an intersystem communication environment.

The information on ACF/VTAM and IMS given in this chapter is for guidance only. Always consult the current ACF/VTAM or IMS publications for the latest information. Some publications references are given under "Books from related libraries" on page viii.

Modules required for ISC

The standard pregenerated system supplied on the CICS distribution volume includes a pregenerated version of each of the CICS management modules required to support intersystem communication. It is therefore not normally necessary to do a system generation to add basic ISC support to your system. A partial regeneration is necessary, however, if you wish to add DL/I support; this is discussed later in this chapter.

You must include the following management programs in your system (by using the SIT or initialization overrides):

- The EXEC interface programs. (Specify EXEC=YES or allow it to default.)

  These programs are not required if you have no command-level application programs. The only ISC facility available in this case is transaction routing (LUTYPE6.2 links only).
- The intersystem communication programs. (Specify ISC=YES.)
- The terminal control program generated by DFHSG PROGRAM=TCP. A version specifying ACCMETH=VTAM, CHNASSY=YES, and VTAMDEV=LUTYPE6 is required.

  The other requirements for the terminal control program will depend upon your total installation requirements. See the *CICS/MVS Customization Guide* for information on how to generate a suitable version, and the *CICS/MVS Resource Definition (Macro)* manual for information on coding the SIT operands.

Refer to the *CICS/MVS Installation Guide* to obtain the suffixes, if any, of the pregenerated versions of these programs.

If, for any reason, you have to perform a partial or complete system generation, VTAM=YES must be specified, or allowed to default, on the DFHSG TYPE=INITIAL macro.

Installing DL/I facilities

If your system is required to access DL/I databases, you will have to regenerate some of the pregenerated CICS management programs.

There are three types of DL/I access to be considered:

1. Your system will access only local DL/I databases
2. Your system will access only remote DL/I databases
3. Your system will access both local and remote DL/I databases.

Each of these access types requires a different combination of CICS modules to be generated. Details on how to do this are given in the *CICS/MVS Installation Guide*.

If only remote access is required, DL/I need not be installed and an IMS licence is not required.

Operating system requirements

There are no special operating system requirements for CICS intersystem communication.

ACF/VTAM definition for CICS

When you define your CICS system to ACF/VTAM you should include the following options on the VTAM APPL statement:

- MODETAB=logon-mode-table-name

  This option should name the VTAM logon mode table that contains your customized logon mode entries (see "ACF/VTAM LOGMODE table entries for CICS" on page 83). You may omit this operand if you choose to add your MODEENT entries to the IBM-supplied default logon mode table (without renaming it).

- AUTH=(ACQ,SPO,VPACE[,PASS])

  ACQ is required to allow CICS to acquire LUTYPE 6 sessions. SPO is required to allow CICS to issue the MVS MODIFY vtamname USERVAR command (see the *CICS/MVS XRF Guide* for further information). VPACE is required to allow pacing of the intersystem flows.

  PASS is required if you intend to use the EXEC CICS ISSUE PASS command, which passes existing terminal sessions to other VTAM applications.

- VPACING=number

  This option specifies the maximum number of normal-flow requests that another logical unit can send on an intersystem session before waiting to receive a pacing response.

  Care is needed in the selection of a suitable pacing count. Too low a value can lead to poor throughput because of the number of line turnarounds required. Too high a value can lead to excessive storage requirements.

- EAS=number

  This option specifies the number of network addressable units that CICS can establish sessions with. The number must include the total number of parallel sessions for this CICS system.

- PARSESS=YES

  This option specifies LUTYPE6 parallel session support.

- SONSCIP=YES

  This option specifies session outage notification (SON) support. SON enables CICS, in certain cases, to recover a session following session failure without requiring operator intervention.

- APPC=NO

  For ACF/VTAM Version 3.2 and above, this is necessary to allow CICS to use VTAM macros.

For further information, see the *Advanced Communication Function Products Installation Guide* manual.

ACF/VTAM LOGMODE table entries for CICS

For LUTYPE6.2 sessions, you can use the MODENAME operand (see "Defining logical unit type 6.2 links" on page 116) to identify a logmode entry that in turn identifies the required entry in the VTAM class-of-service table. Every modename that you supply when you are defining LUTYPE6.2 links must be matched by a VTAM LOGMODE name. All that is required are entries of the following form:

```
MODEENT LOGMODE=modename
MODEEND
```

An entry is also required for the LU services manager modeset (SNASVCMG).

```
MODEENT LOGMODE=SNASVCMG
MODEEND
```

If you plan to use autoinstall for single-session LUTYPE6.2 (APPC) terminals, additional information is required in the MODEENT entry. Details are given in the *CICS/MVS Customization Guide*.

For CICS-to-IMS links that are cross-domain, you must associate the IMS LOGMODE entry with the CICS APPLID, (the generic applid for XRF systems), using the DLOGMOD or MODETAB option.

Considerations for IMS

If your CICS installation is to use CICS-to-IMS intersystem communication, you must ensure that the CICS and the IMS installations are fully compatible.

The following sections are intended to help you communicate effectively with the person responsible for installing the IMS system. They may also be helpful if you have that responsibility. You should also refer to "Chapter 3.1. Defining links to remote systems" on page 91, especially the section on defining compatible CICS and IMS nodes. For full details of IMS installation, refer to the installation guide for the IMS product.

ACF/VTAM definition for IMS

When the IMS system is defined to VTAM the following options should be included on the VTAM APPL statement:

- AUTH=(ACQ,VPACE)

  ACQ is required to allow IMS to acquire LUTYPE 6 sessions. VPACE is required to allow pacing of the intersystem flows.

- VPACING=number

  This option specifies the maximum number of normal-flow requests that another logical unit can send on an intersystem session before waiting to receive a pacing response. An initial value of 5 is suggested.

- EAS=number

  The number of addressable units must include the total number of parallel sessions for this IMS system.

- PARSESS=YES

  This option specifies LUTYPE6 parallel session support.

For further information, see the *Advanced Communication Function Products Installation Guide* manual.

ACF/VTAM LOGMODE table entries for IMS

IMS allows the user to specify some BIND parameters in a VTAM logmode table entry. The CICS logmode table entry must match that of the IMS system. IMS uses the mode table entry specified here in order of priority:

1. The MODETBL parameter of the TERMINAL macro
2. The mode table entry specified in CINIT
3. The DLOGMODE parameter in the VTAMLST APPL statement or the MODE parameter in the IMS /OPNDST command
4. The ACF/VTAM defaults.

Figure 18 shows a typical IMS logmode table entry:

```
LU6NEGPS  MODEENT LOGMODE=LU6NEGPS,  NEGOTIABLE BIND
            PSNDPAC=X'01',           PRIMARY SEND PACING COUNT
            SRCVPAC=X'01',           SECONDARY RECEIVE PACING COUNT
            SSNDPAC=X'01',           SECONDARY SEND PACING COUNT
            TYPE=0,                  NEGOTIABLE
            FMPROF=X'12',            FM PROFILE 18
            TSPROF=X'04',            TS PROFILE 4
            PRIPROT=X'B1',           PRIMARY PROTOCOLS
            SECPROT=X'B1',           SECONDARY PROTOCOLS
            COMPROT=X'70A0',         COMMON PROTOCOLS
            RUSIZES=X'8585',         RU SIZES 256
            PSERVIC=X'060038000000380000000000'  SYSMSG/Q MODEL
          MODEEND
```

Figure 18. A typical IMS logmode table entry

It is very important that the values specified in the MODEENT entry are acceptable to IMS. For further information, see the programming guide for the relevant release of IMS that you are using.

IMS system definition for intersystem communication

This section summarizes the IMS ISC-related macro instructions and parameters that are used in IMS system definition. You should also refer to "Defining compatible CICS and IMS nodes" on page 108. For full details of IMS installation, refer to the installation guide for the IMS product.

The COMM macro instruction

APPLID=name

specifies the applid of the IMS system. For an IMS Version 1 system, and for an IMS Version 2 system generated without XRF support, this is the name that is specified in the NETNAME operand of DEFINE CONNECTION or DFHTCT TYPE=SYSTEM when you define the IMS system to CICS.

However, for an IMS Version 2 system with XRF, the CICS NETNAME operand should specify the USERVAR (that is, the generic applid) that is defined in the DFSHSBxx member of IMSVS.PROCLIB, not the applid from the COMM macro.

RECANY=(number,size)

specifies the number and size of the IMS buffers that are used for VTAM "receive any" commands. For ISC sessions, the buffer size has a 22-byte overhead. It must therefore be at least 22 bytes larger than the CICS buffer size specified in the SENDSIZE operand of DEFINE SESSIONS or the BUFFER operand of DFHTCT TYPE=TERMINAL for the intersystem sessions.

This size applies to all other ACF/VTAM terminals attached to the IMS system, and must be large enough for input from any terminal in the IMS network.

EDTNAME=name
: specifies an alias for ISCEDT in the IMS system. For CICS-IMS ISC, an alias name must not be longer than four characters.

The TYPE macro instruction

UNITYPE=LUTYPE6
: must be specified for ISC.

Parameters of the TERMINAL macro can also be specified in the TYPE macro if they are common to all the terminals defined for this type.

The TERMINAL macro instruction

The TERMINAL macro identifies the remote CICS system to IMS. It therefore serves the equivalent purpose to the DEFINE CONNECTION command or the DFHTCT TYPE=SYSTEM macro in CICS.

NAME=name
: identifies the CICS node to IMS. It must be the same as the APPLID name of the CICS system (the generic applid for XRF systems).

OUTBUF=number
: specifies the size of the IMS output buffer. It must be equal to or greater than 256, and should include the size of any function management headers sent with the data. It must not be greater than the value specified in the RECEIVESIZE operand of the DEFINE SESSIONS commands or the RUSIZE operand of the CICS DFHTCT TYPE=TERMINAL macros for the intersystem sessions.

SEGSIZE=number
: specifies the size of the work area that IMS uses for deblocking incoming messages. We recommend that you use the size of the longest chain that CICS may send. However, if IMS record mode (VLVB) is used exclusively, you could specify the largest record (RU) size.

MODETBL=name
: specifies the name of the VTAM mode table entry to be used. You must omit this parameter if the CICS system resides in a different SNA domain.

OPTIONS=[<u>NOLTWA</u>|LTWA]
: specifies whether Log Tape Write Ahead (LTWA) is required. For LTWA, IMS will log session restart information for all active parallel sessions before sending a syncpoint request. LTWA is recommended for integrity reasons, but it can carry a performance overhead.

OPTIONS=[<u>SYNCSESS</u>|FORCSESS]
: specifies the message resynchronization requirement following an abnormal session termination. SYNCSESS requires both the inbound and the outbound sequence numbers to match (or CICS to be cold started) to allow the session to be restarted. FORCSESS allows the session to be restarted even if a mismatch occurs. SYNCSESS is recommended.

OPTIONS=[TRANSRESP|NORESP|FORCRESP]
specifies the required response mode.

TRANSRESP
specifies that the response mode will be determined on a transaction-by-transaction basis.

NORESP
specifies that response-mode transactions are not allowed. In CICS terms, this means that a CICS application cannot initiate an IMS transaction by using a SEND command, but only by means of a START command.

FORCRESP
forces response mode for all transactions. In CICS terms, this means that a CICS application cannot initiate an IMS transaction by using a START command, but only by means of a SEND command.

OPTIONS=[OPNDST|NOPNDST]
specifies whether sessions can be established from this IMS system.

{COMPT1|COMPT2|COMPT3|COMPT4}={SINGLEn|MULTIn}
specifies the IMS components for the IMS ISC node. Up to four components can be defined for each node. The input and output components to be used for each session are then selected by the ICOMPT and COMPT operands of the SUBPOOL macro.

The following types of component can be defined:

SINGLE1
Used by IMS for asynchronous output. One output message is sent per SNA bracket. The message may or may not begin the bracket, but it always ends the bracket.

SINGLE2
Each message is sent with the SNA change-direction indicator (CD).

MULT1
All asynchronous messages for a given LTERM are sent before the bracket is ended. The end-bracket (EB) occurs after the last message for the LTERM is acknowledged and dequeued.

MULT2
The same as MULT1, but CD is sent instead of EB.

SESSION=number
specifies the number of parallel sessions for the link. Each session is represented by an IMS SUBPOOL macro and by a CICS DEFINE SESSIONS command or a DFHTCT TYPE=TERMINAL macro.

EDIT=[{ NO|YES}][,{ NO|YES}]
: specifies whether user-supplied physical output and input edit routines are to be used.

The VTAMPOOL macro instruction

The SUBPOOL macro heads the list of SUBPOOL macros that define the individual sessions to the remote system.

The SUBPOOL macro instruction

A SUBPOOL macro is required for each session to the remote system.

NAME=subpool-name
: specifies the IMS name for this session. A CICS-IMS session is identified by a "session-qualifier pair" formed from the CICS name for the session and the IMS subpool name.

The CICS name for the session is specified in the SESSNAME operand of the DEFINE SESSIONS command or the TRMIDNT operand of the DFHTCT TYPE=TERMINAL macro for the session.

The IMS subpool name is specified to CICS in the NETNAMEQ operand of the DEFINE SESSIONS command or the NETNAMQ operand of the DFHTCT TYPE=TERMINAL macro.

The NAME macro instruction

The NAME macro defines the logical terminal names associated with the subpool. Multiple LTERMs can be defined per subpool.

COMPT={1|2|3|4}
: specifies the output component associated with this session. The component specified determines the protocol that IMS ISC will use to process messages. A SINGLE1 output component is strongly recommended.

ICOMPT={1|2|3|4}
: specifies the input component associated with this session. When IMS receives a message, it determines the input source terminal by finding the NAME macro that has the matching input component number. A COMPT1 input component must be defined for each session that CICS uses to send START commands.

EDIT=[{ NO|YES}][,{ ULC|UC}]
: The first parameter specifies whether the user-supplied logical terminal edit routine (DFSCNTEO) is to be used.

The second parameter specifies whether the output is to be translated to upper case (UC) before transmission or not (ULC).

Part 3. Resource definition and master terminal operation

This part tells you how to define the various resources that may be required in a CICS intercommunication environment.

CICS holds its resource information in tables (for example the Terminal Control Table, or the File Control Table) that are loaded during CICS startup. You can define resources by coding CICS table definition macro instructions or, for some resource types, by using resource definition online (RDO). You should refer to the *CICS/MVS Resource Definition (Online)* manual and the *CICS/MVS Resource Definition (Macro)* manual for details of these procedures.

"Chapter 3.1. Defining links to remote systems" on page 91 tells you how to define links to remote systems. The links can be MRO links, LUTYPE6.1 links to remote CICS or IMS systems, or LUTYPE6.2 links to remote CICS systems or to other LUTYPE6.2 (APPC) systems or terminals.

"Chapter 3.2. Defining remote resources" on page 133 tells you how to define remote resources to the local CICS system. The resources can be:

- Remote files
- Remote DL/I PSBs
- Remote transient data destination
- Remote temporary storage queues
- Remote terminals
- Remote transactions.

"Chapter 3.3. Defining local resources" on page 151 tells you how to define local resources for ISC and MRO. In general, these resources are those that are required for ISC and MRO and are obtained by including the relevant functional groups in the appropriate tables. However, you do get the opportunity to modify some of the supplied definitions and to provide your own communication profiles.

Chapter 3.1. Defining links to remote systems

This chapter tells you how to define communication links to other systems or to other CICS regions.

You can use either **resource definition online** (RDO) or **macro-level resource definition** to define links to remote systems. Both methods are described in this chapter.

Four basic types of link are described:

1. Links for multiregion operation
2. Links to remote systems using logical unit type 6.1 protocols
3. Links to remote systems using logical unit type 6.2 (APPC) protocols
4. Indirect links for CICS transaction routing.

Links using the ACF/VTAM application-to-application facilities are treated exactly as though they are intersystem links, and can be defined as either LUTYPE6.1 or LUTYPE6.2 links.

Introduction to link definition

The definition of a link to a remote system consists of two basic parts:

1. The definition of the remote system itself
2. The definition of sessions with the remote system.

If the remote system is CICS, or any other system that uses resource definition to define intersystem sessions (for example, IMS), the link definition must be matched by a compatible definition in the remote system. For remote systems with little or no flexibility in their session properties (for example, LUTYPE6.2 terminals), the link definition must match the fixed attributes of the remote system concerned.

| You are recommended to use resource definition online to define links to remote systems.

Resource definition online (RDO)

With resource definition online, the definitions of the remote system and the sessions are always separate, and are not associated with each other until they are installed.

The remote system is defined by the DEFINE CONNECTION command. Each session, or group of parallel sessions, is defined by the DEFINE SESSIONS command.

For single-session APPC terminals, an alternative method of definition, using DEFINE TERMINAL and DEFINE TYPETERM, is available.

Macro-level resource definition

With macro-level resource definition, remote systems are defined by means of the DFHTCT TYPE=SYSTEM macro.

For MRO and LUTYPE6.1 links, the sessions are, in general, also defined in the TYPE=SYSTEM macro. In some circumstances, however, you can (or even must) write separate TYPE=TERMINAL macros for the individual sessions; these are described in the appropriate sections of this chapter.

For LUTYPE6.2 (APPC) links, the DFHTCT TYPE=SYSTEM macro is used to define the remote system. Each group of sessions is then defined by means of a DFHTCT TYPE=MODESET macro. However, a single session to an APPC terminal is defined by means of a DFHTCT TYPE=SYSTEM macro.

Naming the local CICS system

Each of your CICS/MVS systems requires three names: a generic application identifier (APPLID), a specific application identifier, and a system identifier (SYSIDNT).

The APPLIDs of the local CICS system

A CICS/MVS system requires two APPLID names: a **generic** name and a **specific** name. The names are specified in the APPLID operand of the system initialization table:

```
DFHSIT
    APPLID=(generic-id,specific-id)
```

The default value for the generic-id is DBDCCICS. The default value for the specific-id is the value of the generic-id. Either or both of these values can be overridden during CICS start-up.

As explained in the *CICS/MVS XRF Guide*, the active and alternate systems in an XRF pair of CICS systems must have the same generic applid and different specific applids. Note that a CICS system initialized with XRF=NO still has a generic and a specific applid, even if they have the same value.

For ISC, the generic APPLID of a CICS system is the name by which it is known in the intercommunication network; that is, its NETNAME.

For MRO, CICS uses the generic APPLID name to identify itself when it signs on to the CICS interregion SVC, either during startup or in response to a SET IRC OPEN master terminal command.

All APPLIDs in your intercommunication network should be unique. Also, if you plan to use the CICS Monitoring Facilities with SYSEVENT reporting to the resource measurement facility (RMF), you must ensure, particularly in an MRO environment, that the first **four** characters of the APPLIDs are unique.

The SYSIDNT of the local CICS system

The SYSIDNT of a CICS system is a one-to four-character name known only to the CICS system itself.

It is obtained (in order of priority) from:

1. The startup override.
2. The SYSIDNT operand of the DFHSIT macro.
3. The default value **CICS**.

The CICS active and alternate systems that have the same generic applid must also have the same sysidnt.

The SYSIDNT of your CICS system may also have to be specified in the DFHTCT TYPE=INITIAL macro if you are using macro-level resource definition. The only purpose of the SYSIDNT operand of DFHTCT TYPE=INITIAL is to control the assembly of local and remote terminal definitions in the terminal control table. (Terminal definition is described in "Chapter 3.2. Defining remote resources" on page 133.) The sysidnt of a running CICS system is always the one specified in DFHSIT (or the default or override value).

Identifying remote systems

As well as having a SYSIDNT for itself, a CICS system requires a SYSIDNT for every other system with which it can communicate. SYSIDNT names are used to relate session definitions to system definitions; to identify the systems on which remote resources, such as files, reside; and to refer to specific systems in application programs.

SYSIDNT names are private to the CICS system in which they are defined; they are not known by other systems. In particular, the SYSIDNT defined for a remote CICS system is independent of the SYSIDNT by which the remote system knows itself; you need not make them the same.

The mapping between the local, private, SYSIDNT assigned to a remote system and the APPLID by which the remote system is known globally in the network is made when you define the intercommunication link. For remote CICS/MVS systems, this is the **generic** APPLID.

Resource definition online

```
DEFINE CONNECTION(sysidnt) The local name for the remote system
       NETNAME(applid)     The (generic) applid of the remote system
```

Macro-level resource definition:

```
DFHTCT TYPE=SYSTEM,
       SYSIDNT=sysidnt,    The local name for the remote system
       NETNAME=applid      The (generic) applid of the remote system
```

In both cases, if NETNAME is omitted, the sysidnt is taken to be the applid of the remote system. Each sysidnt name must be unique in a CICS system.

Defining links for multiregion operation

This section describes how to define an interregion communication link between the local CICS system and another CICS region in the same processor.

You can use RDO to define a CONNECTION-SESSIONS pair. Alternatively, you can use a single DFHTCT TYPE=SYSTEM macro instruction to create a pool of parallel sessions between the local and the remote CICS system.

From the point of view of the local CICS system, each session on the link is characterized as either a SEND session or a RECEIVE session. SEND sessions are used to carry an initial request from the local to the remote system and to carry any other data flows associated with the initial request. Similarly, RECEIVE sessions are used to receive initial requests from the remote system.

Interregion communication protocols are basically similar to SNA protocols, and an initial request is a request that carries a begin bracket indicator. However, there is no concept of bidding on an interregion link, so that initial requests can never be sent on a RECEIVE session. You should keep this fact in mind when you decide how many send and receive sessions you will require.

You must always specify at least one send session and one receive session.

Resource definition online

The RDO definition for an MRO link is shown in Figure 19 on page 95. (This figure also shows the macro form to show how the operands are related.)

For RDO, you define the **connection** and the associated group of **sessions** separately. The two definitions are individual "objects" on the CICS system definition file (CSD), and they are not associated with each other until the group is installed. The following rules apply for MRO links:

1. The CONNECTION and SESSIONS must be in the same GROUP.
2. The SESSIONS must have PROTOCOL(LU61), but the PROTOCOL operand of CONNECTION must be left blank.
3. The CONNECTION operand of SESSIONS must match the sysidnt specified for the CONNECTION.
4. Only one SESSIONS definition can be related to an MRO CONNECTION.

You can specify ACCESSMETHOD(XM) to select MVS cross-memory services for the link. Cross-memory services are used only if the other end of the link also specifies cross-memory. To select the CICS Type 2 SVC for interregion communication, use ACCESSMETHOD(IRC).

As explained earlier in this chapter, the **sysidnt** is the local name for the CICS system to which the link is being defined. The NETNAME must be the name by which the remote system is known to IRC (CICS interregion communication), that is, its applid as defined in the system initialization table (SIT) (or, in an XRF environment, its generic applid). If you do not specify a NETNAME, sysidnt must satisfy this requirement.

```
RDO Definition                 Macro-Level Definition

DEFINE                         DFHTCT  TYPE=SYSTEM
  CONNECTION(sysidnt)             ,SYSIDNT=sysidnt
  GROUP(groupname)
  ACCESSMETHOD(IRC|XM)            ,ACCMETH={IRC|(IRC,XM)}
  NETNAME(name)                   ,NETNAME=name
  SECURITYNAME(name)              ,XSNAME=name
  INSERVICE(NO)

DEFINE
  SESSIONS(csdname)
  GROUP(groupname)
  CONNECTION(sysidnt)
  PROTOCOL(LU61)
  RECEIVEPFX(prefix1)             ,RECEIVE=(prefix1,number1)
  RECEIVECOUNT(number1)
  SENDPFX(prefix2)                ,SEND=(prefix2,number2)
  SENDCOUNT(number2)

  OPERPRIORITY(number)            ,OPERPRI=number
  OPERRSL(number)                 ,OPERRSL=number
  OPERSECURITY(number)            ,OPERSEC=number
  IOAREALEN(value)                ,TIOAL=value
  SESSPRIORITY(number)            ,TRMPRTY=number
  INSERVICE(NO)                   ,TRMSTAT='OUT OF SERVICE'
```

Figure 19. Defining an MRO link

You must specify the number of SEND and RECEIVE sessions that are required (at least one of each) and you must also specify prefixes to allow the sessions to be named.

With RDO, the prefixes and the number of sessions are specified separately (for example, in SENDPFX and SENDCOUNT).

A prefix is a one-character or two-character string that is used to generate session identifiers (TRMIDNTs). The count specifies the number of parallel sessions that is required. The combination of the prefix and the count must not exceed four characters.

For example:

```
RECEIVEPFX(RR)
RECEIVECOUNT(10)
```

generates 10 receive sessions with identifiers RR1 through RR10.

```
RECEIVEPFX(R)
RECEIVECOUNT(150)
```

generates 15U receive sessions with identifiers R1 through R150.

Example of MRO link definition: The following example shows a typical definition for an MRO link.

```
DEFINE
  CONNECTION(CICB)      local name for remote system
  GROUP(groupname)      groupname of related definitions
  ACCESSMETHOD(XM)      cross-memory services
  NETNAME(CICSB)        global name of remote system
  SECURITYNAME(OPA)
  INSERVICE(NO)

DEFINE
  SESSIONS(csdname)     unique csd name
  GROUP(groupname)      same group as the connection
  CONNECTION(CICB)      related connection
  PROTOCOL(LU61)
  RECEIVEPFX(RB)
  RECEIVECOUNT(5)       5 receive sessions RB1 through RB5
  SENDPFX(SB)
  SENDCOUNT(3)          3 send sessions SB1 through SB3
  OPERPRIORITY(35)
  OPERRSL(1)
  OPERSECURITY(15)
  IOAREALEN(300)        minimum TIOA size for sessions
  SESSPRIORITY(100)
  INSERVICE(NO)
```

Figure 20. MRO link definition example — RDO

Macro-level resource definition

The macro-level definition for an MRO link is shown in Figure 19 on page 95. (This figure also shows the RDO form to show how the operands are related.)

You can specify ACCMETH=(IRC,XM) to select MVS cross-memory services for the link. Cross-memory services are used only if the other end of the link also specifies cross-memory. To select the CICS type 2 SVC for interregion communication, use ACCMETH=IRC.

As explained earlier in this chapter, the **sysidnt** is the local name for the CICS system to which the link is being defined. The NETNAME must be the name by which the remote system is known to IRC (CICS interregion communication), that is, its applid as defined in the system initialization table (SIT) (or, in an XRF environment, its generic applid). If you do not specify a NETNAME, sysidnt must satisfy this requirement.

You must specify the number of SEND and RECEIVE sessions that are required (at least one of each) and you must also specify prefixes to allow the sessions to be named. With macro-level definition, the prefixes and the number of sessions are both specified in the same operand (for example, in SEND).

A prefix is a one-character or two-character string that is used to generate session identifiers (TRMIDNTs). The count specifies the number of parallel sessions that is required. The combination of the prefix and the count must not exceed four characters.

For example:

```
RECEIVE=(RR,10)
```

generates 10 receive sessions with identifiers RR1 through RR10.

```
RECEIVE=(R,150)
```

generates 150 receive sessions with identifiers R1 through R150.

Example of MRO link definition: The following example shows a typical definition for an MRO link.

```
DFHTCT TYPE=SYSTEM,
   ACCMETH=(IRC,XM),   cross-memory services
   SYSIDNT=CICB,       local name for remote system
   NETNAME=CICSB,      global name of remote system
   RECEIVE=(RB,5),     5 receive sessions RB1 through RB5
   SEND=(SB,3),        3 send sessions SB1 through SB3
   XSNAME=OPA,
   OPERPRI=35,
   OPERRSL=1
   OPERSEC=15,
   TIOAL=(300),        minimum TIOA size for sessions
   TRMPRTY=100
   TRMSTAT='OUT OF SERVICE'
```

Figure 21. MRO link definition example — macro-level

Choosing the access method for MRO

You can specify ACCESSMETHOD(XM) to select MVS cross-memory services for an MRO link. Cross-memory services are available only if both ends of the link select it; otherwise ACCESSMETHOD(IRC) is used. ACCESSMETHOD(IRC) uses the CICS type 2 SVC for communication between the address spaces.

The use of MVS cross-memory services:

- Reduces the path length for communication
- Uses less MVS CSA storage (as shared buffers are not needed) than SVC operation
- Requires CICS address spaces to be non-swappable.

However, MVS cross-memory services can create a security exposure (see "Use of MVS cross-memory services" on page 327).

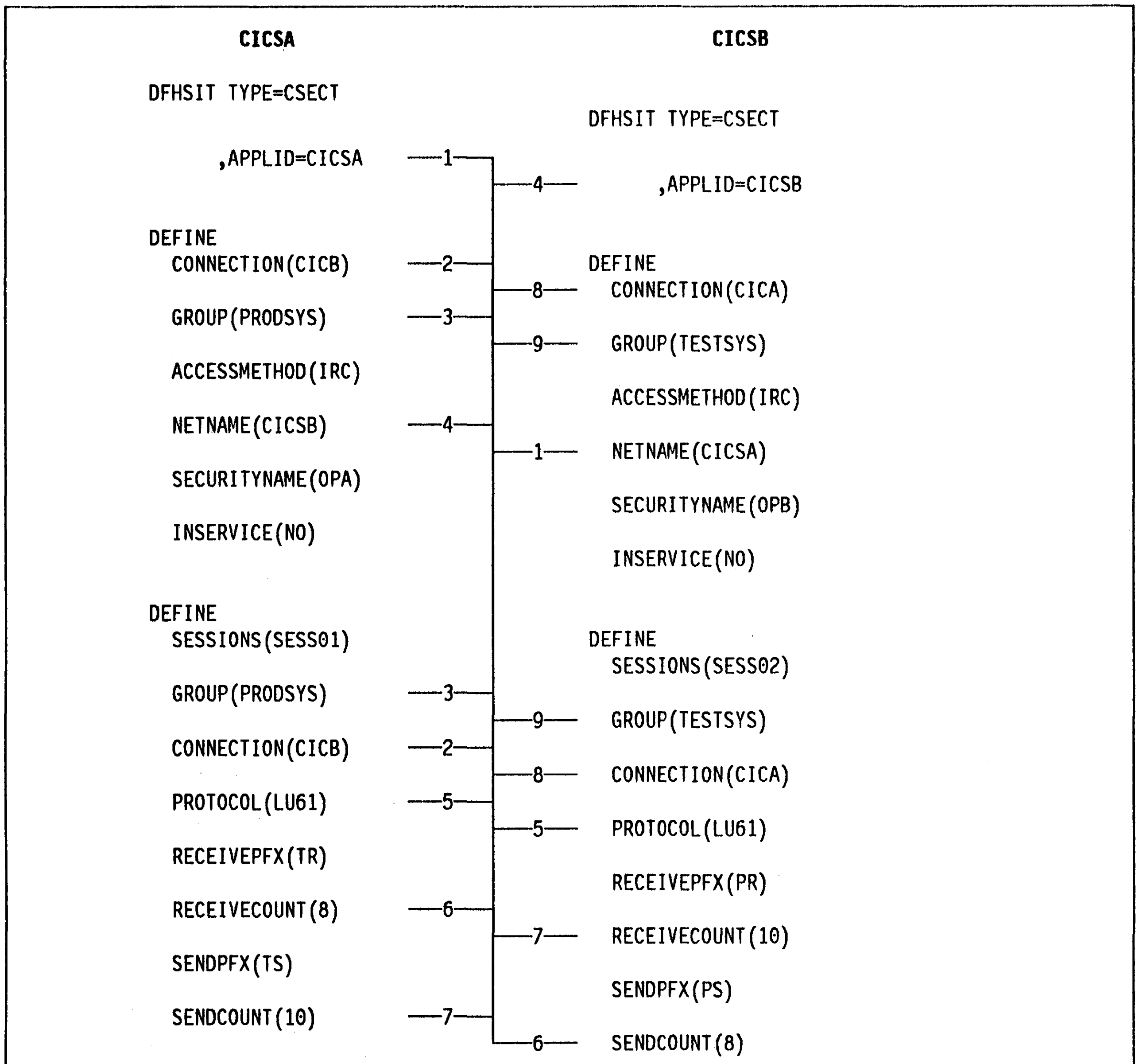
Defining compatible MRO nodes

An MRO link must be defined in both of the systems that it connects. You must ensure that the two definitions are compatible with each other. For example, if one definition specifies 6 sending sessions, the other definition requires 6 receiving sessions.

The following sections describe how the operands of the two definitions are related to each other. Three types of definition are shown:

1. Resource definition online used in both systems.
2. Macro definition used in both systems.
3. RDO used in one system and macro definition used in the other.

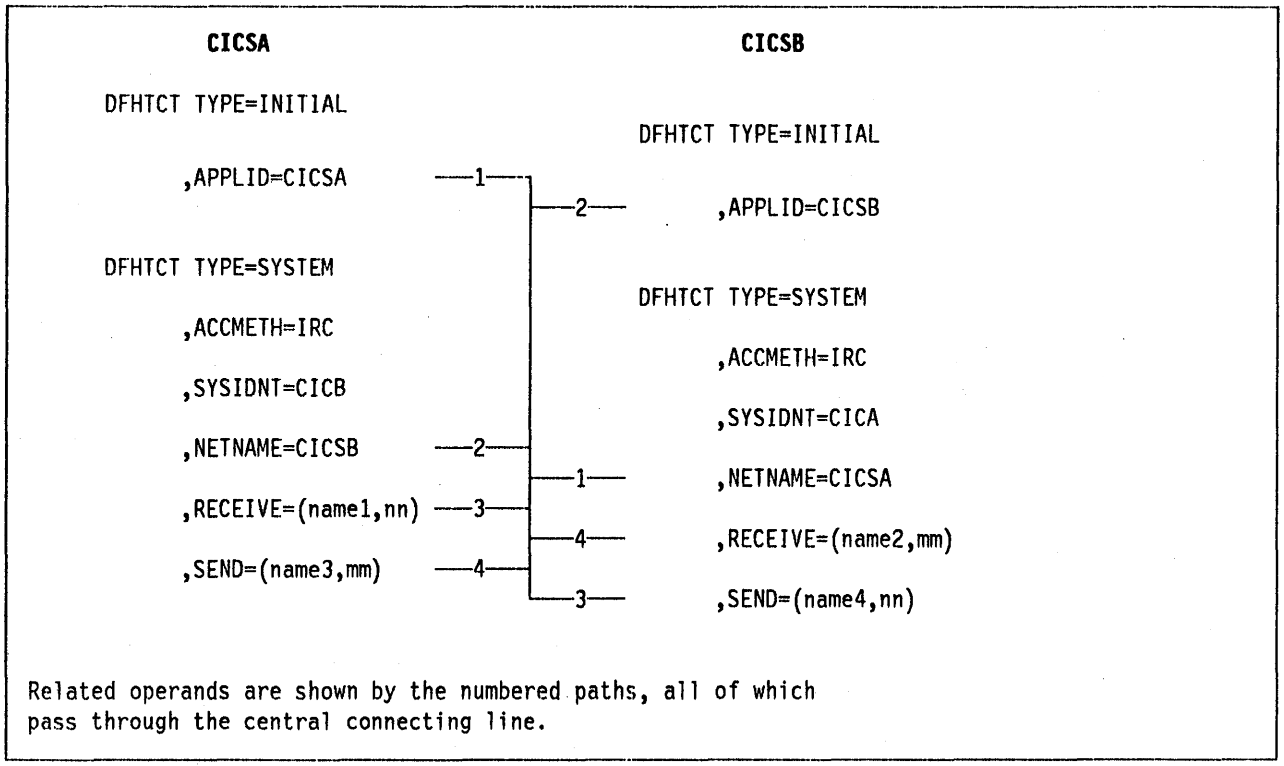
The compatibility requirements are shown in Figure 22 on page 99, Figure 23 on page 100, and Figure 24 on page 101.



```
                 CICSA                                          CICSB

        DFHSIT TYPE=CSECT
                                                       DFHSIT TYPE=CSECT
              ,APPLID=CICSA         —1—┐
                                       ├—4—                  ,APPLID=CICSB

        DEFINE
          CONNECTION(CICB)          —2—┤                   DEFINE
                                       ├—8—                  CONNECTION(CICA)
          GROUP(PRODSYS)            —3—┤
                                       ├—9—                  GROUP(TESTSYS)
          ACCESSMETHOD(IRC)            │
                                       │                     ACCESSMETHOD(IRC)
          NETNAME(CICSB)            —4—┤
                                       ├—1—                  NETNAME(CICSA)
          SECURITYNAME(OPA)            │
                                       │                     SECURITYNAME(OPB)
          INSERVICE(NO)                │
                                       │                     INSERVICE(NO)

        DEFINE                         │
          SESSIONS(SESS01)             │                   DEFINE
                                       │                     SESSIONS(SESS02)
          GROUP(PRODSYS)            —3—┤
                                       ├—9—                  GROUP(TESTSYS)
          CONNECTION(CICB)          —2—┤
                                       ├—8—                  CONNECTION(CICA)
          PROTOCOL(LU61)            —5—┤
                                       ├—5—                  PROTOCOL(LU61)
          RECEIVEPFX(TR)               │
                                       │                     RECEIVEPFX(PR)
          RECEIVECOUNT(8)           —6—┤
                                       ├—7—                  RECEIVECOUNT(10)
          SENDPFX(TS)                  │
                                       │                     SENDPFX(PS)
          SENDCOUNT(10)             —7—┤
                                       └—6—                  SENDCOUNT(8)
```

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

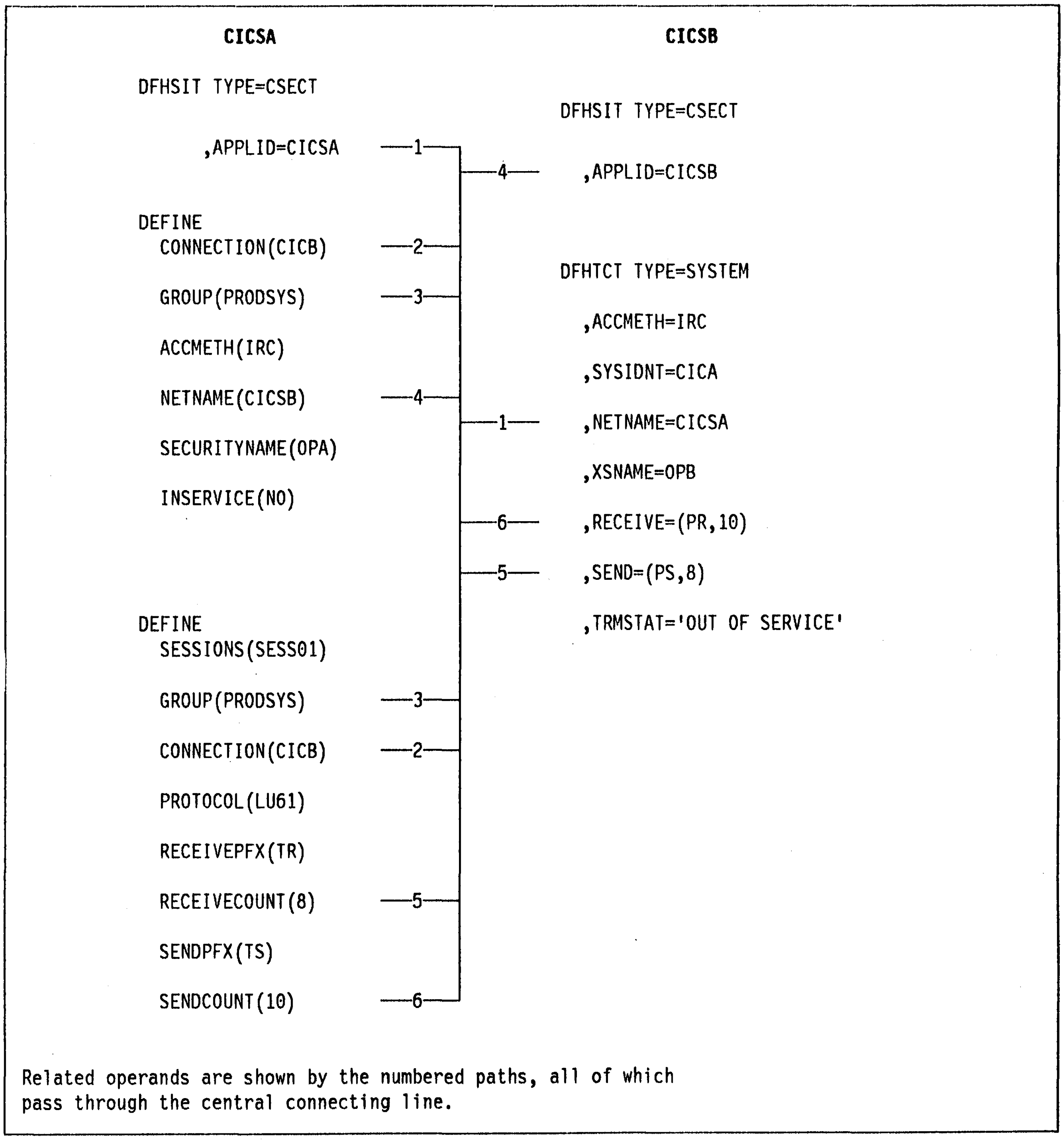
Figure 22. Defining compatible MRO nodes — RDO



```
          CICSA                                        CICSB

    DFHTCT TYPE=INITIAL
                                          DFHTCT TYPE=INITIAL

          ,APPLID=CICSA         ──1──┐
                                     ├──2──      ,APPLID=CICSB
                                     │
    DFHTCT TYPE=SYSTEM               │
                                     │    DFHTCT TYPE=SYSTEM
          ,ACCMETH=IRC               │
                                     │           ,ACCMETH=IRC
          ,SYSIDNT=CICB              │
                                     │           ,SYSIDNT=CICA
          ,NETNAME=CICSB        ──2──┤
                                     ├──1──      ,NETNAME=CICSA
          ,RECEIVE=(name1,nn)   ──3──┤
                                     ├──4──      ,RECEIVE=(name2,mm)
          ,SEND=(name3,mm)      ──4──┤
                                     └──3──      ,SEND=(name4,nn)
```

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

Figure 23. Defining compatible MRO nodes — macro-level



```
              CICSA                                    CICSB

        DFHSIT TYPE=CSECT
                                              DFHSIT TYPE=CSECT

              ,APPLID=CICSA      ──1──┐
                                      ├──4──    ,APPLID=CICSB

        DEFINE
          CONNECTION(CICB)       ──2──┤
                                      │       DFHTCT TYPE=SYSTEM
          GROUP(PRODSYS)         ──3──┤
                                      │         ,ACCMETH=IRC
          ACCMETH(IRC)                │
                                      │         ,SYSIDNT=CICA
          NETNAME(CICSB)         ──4──┤
                                      ├──1──    ,NETNAME=CICSA
          SECURITYNAME(OPA)           │
                                      │         ,XSNAME=OPB
          INSERVICE(NO)               │
                                      ├──6──    ,RECEIVE=(PR,10)
                                      │
                                      ├──5──    ,SEND=(PS,8)
                                      │
        DEFINE                        │         ,TRMSTAT='OUT OF SERVICE'
          SESSIONS(SESS01)            │
                                      │
          GROUP(PRODSYS)         ──3──┤
                                      │
          CONNECTION(CICB)       ──2──┤
                                      │
          PROTOCOL(LU61)              │
                                      │
          RECEIVEPFX(TR)              │
                                      │
          RECEIVECOUNT(8)        ──5──┤
                                      │
          SENDPFX(TS)                 │
                                      │
          SENDCOUNT(10)          ──6──┘

Related operands are shown by the numbered paths, all of which
pass through the central connecting line.
```

Figure 24. Defining compatible MRO nodes — mixed RDO and macro

Defining logical unit type 6.1 links

LUTYPE6.1 links are necessary for intersystem communication between CICS and IMS. You can also define LUTYPE6.1 links between CICS systems. However, you are advised to use LUTYPE6.2 links for CICS-to-CICS communication whenever possible.

Methods of defining LUTYPE6.1 links

Both resource definition online and macro-level resource definition offer two methods of defining LUTYPE6.1 sessions. However, you are recommended to use RDO.

Resource definition online

With RDO, a DEFINE CONNECTION is always required to define the remote system. The sessions, however, can be defined in either of the following ways:

1. By using a single DEFINE SESSIONS command to define a pool of sessions with identical characteristics. This is the most convenient method for CICS-to-CICS communication.
2. By using a separate DEFINE SESSIONS command to define each individual session. This method must be used to define sessions with systems such as IMS which require individual sessions to be explicitly named.

Macro-level resource definition

Using macro-level definition, the two methods are:

1. Using a single DFHTCT TYPE=SYSTEM macro to define a pool of sessions with identical characteristics. This is the most convenient method for CICS-to-CICS communication.
2. Using a DFHTCT TYPE=SYSTEM macro to define the remote system, followed by DFHTCT TYPE=TERMINAL macros to define the individual sessions. This method must be used to define sessions with systems such as IMS which require individual sessions to be explicitly named.

Defining CICS-to-CICS LUTYPE6.1 links

This section describes how to define a pool of LUTYPE6.1 sessions of identical characteristics.

This method of link definition is the most convenient for CICS-to-CICS ISC links. If, however, you have a requirement for sessions of differing characteristics, you can use the definition method described under "Defining CICS-to-IMS LUTYPE6.1 links" on page 108.

From the point of view of the local CICS system, each session on the link is characterized as either a SEND session or a RECEIVE session. A SEND session is one in which the local CICS is the secondary (that is, bind receiver) and is the contention winner. A RECEIVE session is one in which the local CICS is the primary (that is, bind sender) and is the contention loser. When CICS allocates a intersystem session to the remote system, it always tries to allocate a contention winner. Only if no contention winners are available will it select a contention loser. It will then have to bid for permission to begin a bracket.

To avoid the overhead of bidding, you should base the numbers of SEND and RECEIVE sessions on the expected directions and frequencies of flows between the two systems.

Resource definition online

The RDO definition for an LUTYPE6.1 link is shown in Figure 25 on page 104. (This figure also shows the macro form to show how the operands are related.)

For RDO, you define the **connection** and the associated group of **sessions** separately. The two definitions are individual "objects" on the CICS system definition file (CSD), and they are not associated with each other until the group is installed. The following rules apply for LUTYPE6.1 links:

1. The CONNECTION and SESSIONS must be in the same GROUP.
2. Both the CONNECTION and the SESSIONS must have PROTOCOL(LU61).
3. The CONNECTION operand of SESSIONS must match the sysid specified for the CONNECTION.

The AUTOCONNECT and INSERVICE operands

The AUTOCONNECT operand on the DEFINE CONNECTION command has no function for a LUTYPE6.1 connection.

On the DEFINE SESSIONS commands, AUTOCONNECT(YES|ALL) specifies that CICS is to bind all the sessions of the group as part of the initialization of the system. For this to take effect, however, INSERVICE(YES) must also be specified on the DEFINE CONNECTION command.

INSERVICE(NO) on the DEFINE CONNECTION command initializes sessions to an out-of-service state only if AUTOCONNECT(NO) is specified in the associated DEFINE SESSIONS command.

Each CICS system binds its own contention losers; that is, its receive sessions. At the same time, it passes an indication to request the remote system to do the same. In this way, all sessions are bound in one operation.

Macro-level resource definition

The form of the TYPE=SYSTEM macro used to define a pool of LUTYPE6.1 sessions is shown in Figure 25 on page 104. (This figure also shows the RDO form to show how the operands are related.)

```
RDO Definition                  Macro-Level Definition

DEFINE                          DFHTCT   TYPE=SYSTEM
  CONNECTION(sysidnt)             ,SYSIDNT=sysidnt
  GROUP(groupname)
  NETNAME(name)                   ,NETNAME=name
  ACCESSMETHOD(VTAM)              ,ACCMETH=VTAM
  PROTOCOL(LU61)
  DATASTREAM(USER|3270|           ,DATASTR={USER|3270|
             SCS|STRFIELD|                 SCS|STRFIELD|
             LMS)                          LMS}
  RECORDFORMAT(U|VB)              ,RECFM={U|VB}
  SECURITYNAME(name)              ,XSNAME=name

DEFINE
  SESSIONS(csdname)
  GROUP(groupname)
  CONNECTION(sysidnt)
  PROTOCOL(LU61)
  RECEIVEPFX(prefix1)             ,RECEIVE=(prefix1,number1)
  RECEIVECOUNT(number1)
  SENDPFX(prefix2)                ,SEND=(prefix2,number2)
  SENDCOUNT(number2)
  SENDSIZE(size)                  ,BUFFER=size
  RECEIVESIZE(size)               ,RUSIZE=size
  BUILDCHAIN(Y)                   ,CHNASSY=YES
  AUTOCONNECT(YES|NO)            [,CONNECT=AUTO]
  INSERVICE(YES)
  OPERID(operator-id)             ,OPERID=operator-id
  OPERPRIORITY(number)            ,OPERPRI=number
  OPERRSL(number)                 ,OPERRSL=number
  OPERSECURITY(number)            ,OPERSEC=number
  IOAREALEN(value)                ,TIOAL=value
  SESSPRIORITY(number)            ,TRMPRTY=number
                                  ,TRMSTAT=TRANSCEIVE
```

Figure 25. Defining an LUTYPE6.1 link

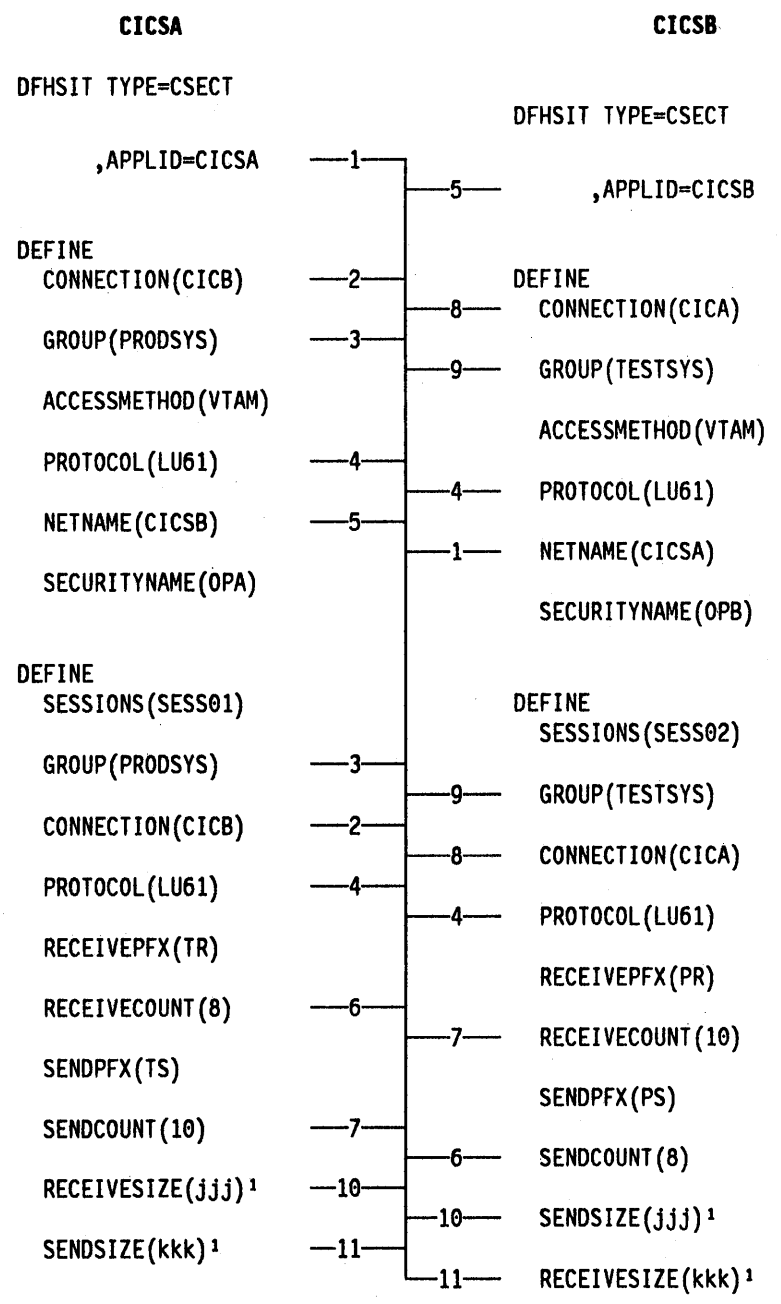
Defining compatible CICS LUTYPE6.1 nodes

When you are defining an LUTYPE6.1 link between two CICS systems, you must ensure that the definitions of the link in each of the systems are compatible.

The following sections describe how the operands of the two definitions are related to each other. Three types of definition are shown:

1. Resource definition online used in both systems.
2. Macro definition used in both systems.
3. RDO used in one system and macro definition used in the other.

The compatibility requirements are shown in Figure 26 on page 105, Figure 27 on page 106, and Figure 28 on page 107.



```
            CICSA                                       CICSB

      DFHSIT TYPE=CSECT
                                                  DFHSIT TYPE=CSECT
            ,APPLID=CICSA       —1—┐
                                   ├—5—           ,APPLID=CICSB

      DEFINE
        CONNECTION(CICB)        —2—┤              DEFINE
                                   ├—8—             CONNECTION(CICA)
        GROUP(PRODSYS)          —3—┤
                                   ├—9—             GROUP(TESTSYS)
        ACCESSMETHOD(VTAM)
                                   │                ACCESSMETHOD(VTAM)
        PROTOCOL(LU61)          —4—┤
                                   ├—4—             PROTOCOL(LU61)
        NETNAME(CICSB)          —5—┤
                                   ├—1—             NETNAME(CICSA)
        SECURITYNAME(OPA)          │
                                   │                SECURITYNAME(OPB)

      DEFINE                       │
        SESSIONS(SESS01)           │              DEFINE
                                   │                SESSIONS(SESS02)
        GROUP(PRODSYS)          —3—┤
                                   ├—9—             GROUP(TESTSYS)
        CONNECTION(CICB)        —2—┤
                                   ├—8—             CONNECTION(CICA)
        PROTOCOL(LU61)          —4—┤
                                   ├—4—             PROTOCOL(LU61)
        RECEIVEPFX(TR)             │
                                   │                RECEIVEPFX(PR)
        RECEIVECOUNT(8)         —6—┤
                                   ├—7—             RECEIVECOUNT(10)
        SENDPFX(TS)                │
                                   │                SENDPFX(PS)
        SENDCOUNT(10)           —7—┤
                                   ├—6—             SENDCOUNT(8)
        RECEIVESIZE(jjj)¹      —10—┤
                                   ├—10—            SENDSIZE(jjj)¹
        SENDSIZE(kkk)¹         —11—┤
                                   └—11—            RECEIVESIZE(kkk)¹
```

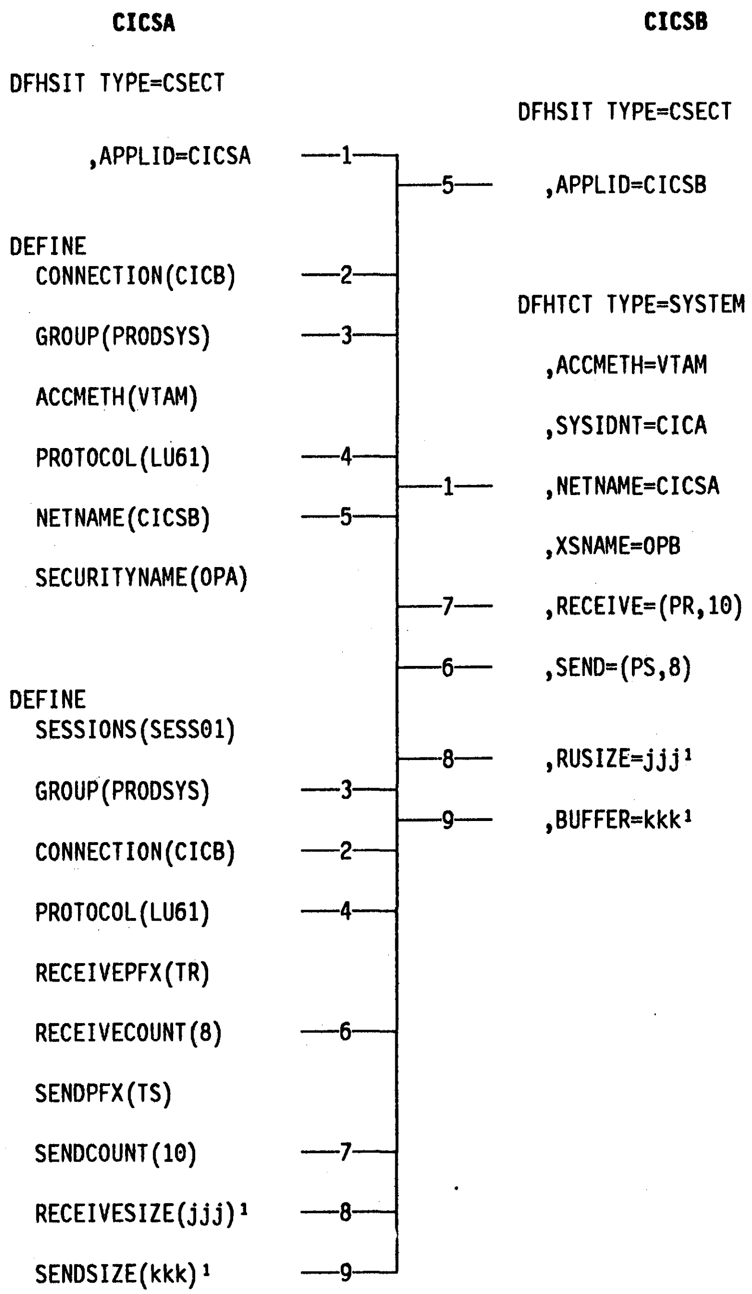
Related operands are shown by the numbered paths, all of which pass through the central connecting line.

[1] CICS will negotiate RECEIVESIZE and SENDSIZE at BIND time if they do not match.

Figure 26. Defining compatible CICS LUTYPE6.1 ISC nodes — RDO

```
          CICSA                                   CICSB

  DFHSIT TYPE=CSECT
                                          DFHSIT TYPE=CSECT

        ,APPLID=CICSA        ——1——┐
                                  ├——2——       ,APPLID=CICSB

  DFHTCT TYPE=INITIAL             │
                                  │       DFHTCT TYPE=INITIAL
        ,SYSIDNT=(sysa)           │
                                  │             ,SYSIDNT=(sysb)

  DFHTCT TYPE=SYSTEM              │
                                  │       DFHTCT TYPE=SYSTEM
        ,ACCMETH=VTAM             │
                                  │             ,ACCMETH=VTAM
        ,SYSIDNT=CICB             │
                                  │             ,SYSIDNT=CICA
        ,NETNAME=CICSB       ——2——┤
                                  ├——1——        ,NETNAME=CICSA
        ,RECEIVE=(name1,nn)  ——3——┤
                                  ├——4——        ,RECEIVE=(name2,mm)
        ,SEND=(name3,mm)     ——4——┤
                                  ├——3——        ,SEND=(name4,nn)
        ,RUSIZE=jjj¹         ——5——┤
                                  ├——6——        ,RUSIZE=kkk¹
        ,BUFFER=kkk¹         ——6——┤
                                  └——5——        ,BUFFER=jjj¹

 Related operands are shown by the numbered paths, all of which
 pass through the central connecting line.

 ¹ CICS will negotiate RUSIZE and BUFFER at BIND time if
 they do not match.
```

Figure 27. Defining compatible CICS LUTYPE6.1 ISC nodes — macro-level



```
              CICSA                                        CICSB

       DFHSIT TYPE=CSECT
                                                 DFHSIT TYPE=CSECT

            ,APPLID=CICSA       ──1──┐
                                     ├──5──        ,APPLID=CICSB
                                     │
       DEFINE                        │
         CONNECTION(CICB)       ──2──┤
                                     │           DFHTCT TYPE=SYSTEM
         GROUP(PRODSYS)         ──3──┤
                                     │             ,ACCMETH=VTAM
         ACCMETH(VTAM)               │
                                     │             ,SYSIDNT=CICA
         PROTOCOL(LU61)         ──4──┤
                                     ├──1──        ,NETNAME=CICSA
         NETNAME(CICSB)         ──5──┤
                                     │             ,XSNAME=OPB
         SECURITYNAME(OPA)           │
                                     ├──7──        ,RECEIVE=(PR,10)
                                     │
                                     ├──6──        ,SEND=(PS,8)
       DEFINE                        │
         SESSIONS(SESS01)            │
                                     ├──8──        ,RUSIZE=jjj¹
         GROUP(PRODSYS)         ──3──┤
                                     ├──9──        ,BUFFER=kkk¹
         CONNECTION(CICB)       ──2──┤
                                     │
         PROTOCOL(LU61)         ──4──┤
                                     │
         RECEIVEPFX(TR)              │
                                     │
         RECEIVECOUNT(8)        ──6──┤
                                     │
         SENDPFX(TS)                 │
                                     │
         SENDCOUNT(10)          ──7──┤
                                     │
         RECEIVESIZE(jjj)¹      ──8──┤
                                     │
         SENDSIZE(kkk)¹         ──9──┘
```

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

¹ CICS will negotiate these values at BIND time if they do not match.

Figure 28. Defining compatible CICS LUTYPE6.1 ISC nodes — mixed RDO and macro

Defining CICS-to-IMS LUTYPE6.1 links

A link to an IMS system requires a definition of the connection (or system) and a separate definition of each of the sessions.

| You are recommended to use resource definition online to define links to remote systems.

Resource definition online

The RDO form of definition for individual LUTYPE6.1 sessions is shown in Figure 29 on page 109. (This figure also shows the macro form to show how the operands are related.)

Macro-level resource definition

The macro-level form of definition for individual LUTYPE6.1 sessions is shown in Figure 29 on page 109. (This figure also shows the RDO form to show how the operands are related.)

The TRMTYPE, TRMIDNT, SYSIDNT, NETNAMQ, and SESTYPE operands must be coded for each session that you define. The remaining operands of TYPE=TERMINAL can optionally be coded on the TYPE=SYSTEM macro to provide defaults for all the defined sessions. Also, the CONNECT, DATASTR, and RECFM operands of TYPE=SYSTEM can be coded for individual sessions if required.

Defining compatible CICS and IMS nodes

The definition of CICS-IMS ISC links requires you to understand the relationship between the way remote systems and sessions are defined in CICS and the way they are defined in IMS. This section is intended to enable you to write suitable CICS definitions and to ensure that they are compatible with the corresponding IMS definitions.

An overview of IMS system definition is given in "Chapter 2.2. Installation considerations for intersystem communication" on page 81. The relationships between CICS and IMS definitions are summarized in Figure 30 on page 112 (RDO) and in Figure 31 on page 113 (macro-level definition).

RDO terms are used in the following discussion of the compatibility requirements. Refer to Figure 29 on page 109 for the equivalent macro-level operands.

System names

The network name of the CICS system (its generic applid) is specified in the APPLID operand of the DFHSIT macro. (It could be provided as an override during CICS startup or in the APPLID operand of the DFHTCT TYPE=INITIAL macro.) This name must be specified in the NAME operand of the IMS TERMINAL macro that defines the CICS system.

For IMS Version 1 systems, and for IMS Version 2 systems generated without XRF support, the network name of the IMS system is specified in the APPLID operand of the IMS COMM macro.

For IMS Version 2 systems with XRF, the network name is the USERVAR that is defined in the DFSHSBxx member of IMSVS.PROCLIB.

You must specify the network name in the NETNAME operand of the DEFINE CONNECTION command that defines the IMS system.

```
RDO Definition                    Macro-Level Definition

DEFINE                            DFHTCT   TYPE=SYSTEM
  CONNECTION(sysidnt)                ,SYSIDNT=sysidnt
  GROUP(groupname)
  NETNAME(name)                      ,NETNAME=name
  ACCESSMETHOD(VTAM)                 ,ACCMETH=VTAM
  PROTOCOL(LU61)
  DATASTREAM(USER|3270|              ,DATASTR=({USER|3270|
             SCS|STRFIELD|                      SCS|STRFIELD|
             LMS)                               LMS})
  RECORDFORMAT(U|VB)                 ,RECFM={U|VB}
  SECURITYNAME(name)                 ,XSNAME=name


Each individual session is then defined as follows:

DEFINE                            DFHTCT   TYPE=TERMINAL
  SESSIONS(csdname)
  GROUP(groupname)
  SESSNAME(name)                     ,TRMIDNT=name
  CONNECTION(sysidnt)                ,SYSIDNT=sysidnt
  NETNAMEQ(name)                     ,NETNAMQ=name
  PROTOCOL(LU61)                     ,TRMTYPE=LUTYPE6
  SENDCOUNT(0|1)                     ,SESTYPE= SEND|RECEIVE
  RECEIVECOUNT(1|0)
  SENDSIZE(size)                     ,BUFFER=size
  RECEIVESIZE(size)                  ,RUSIZE=size
  BUILDCHAIN(Y)                      ,CHNASSY=YES
  OPERID(operator-id)                ,OPERID=operator-id
  OPERPRIORITY(number)               ,OPERPRI=number
  OPERRSL(number)                    ,OPERRSL=number
  OPERSECURITY(number)               ,OPERSEC=number
  AUTOCONNECT(NO|YES)               [,CONNECT=AUTO|ALL]
  INSERVICE(YES)
  IOAREALEN(value)                   ,TIOAL=value
  SESSPRIORITY(number)               ,TRMPRTY=number
                                     ,TRMSTAT=TRANSCEIVE
```

Figure 29. Defining an LUTYPE6.1 link with individual sessions

Number of sessions

In IMS, the number of parallel sessions that are required between the CICS and IMS system must be specified in the SESSION operand of the IMS TERMINAL macro. Each session is then represented by a SUBPOOL entry in the IMS VTAMPOOL. In CICS, each of these sessions is represented by an individual session definition.

Session names

Each CICS-IMS session is uniquely identified by a "session-qualifier pair", which is formed from the CICS name for the session and the IMS name for the session.

The CICS name for the session is specified in the SESSNAME operand of the DEFINE SESSIONS command. For sessions that are to be initiated by IMS, this name must correspond to the ID parameter of the IMS OPNDST command for the session. For sessions initiated by CICS, the name is supplied on the CICS OPNDST command and is saved by IMS.

The IMS name for the session is specified in the NAME operand of the IMS SUBPOOL macro. You must make the relationship between the session names explicit by coding this name in the NETNAMEQ operand of the corresponding DEFINE SESSIONS command.

The CICS and the IMS names for a session can be the same, and this approach is recommended for operational convenience.

Other session parameters

This section lists the remaining operands of the DEFINE CONNECTION and DEFINE SESSIONS commands that are of significance for CICS-IMS sessions.

SENDSIZE

This operand specifies the maximum request unit (RU) size that the remote IMS system can receive. The equivalent IMS value is specified in the RECANY parameter of the IMS COMM macro. You must specify a size that is:

1. Not less than 256 bytes
2. At least 22 bytes less than the value in the RECANY parameter.

BUILDCHAIN(Y)

specifies that multiple RU chains are to be assembled before being passed to the application program. A complete chain will be passed to the application program in response to each RECEIVE command, and the application will have to perform any required deblocking.

BUILDCHAIN(Y) must be specified for LUTYPE6.1 sessions.

DATASTREAM(USER)

must be specified or allowed to default.

This operand is used only when CICS is communicating with IMS by using the START command (asynchronous processing). CICS messages generated by the START command always cause IMS to interpret the data stream profile as input for component 1.

The data stream profile for distributed transaction processing can be specified by the application program by means of the DATASTR option of the BUILD ATTACH command.

RECORDFORMAT(U|VB)
specifies the type of chaining that CICS is to use for transmissions on this session that are initiated by START commands (asynchronous processing).

Two types of data handling algorithms are supported between CICS and IMS:

1. chained

   Messages are sent as SNA chains. The user can use private blocking and deblocking algorithms. This format corresponds to RECORDFORMAT(U).

2. variable length variable blocked records (VLVB)

   Messages are sent in variable length variable blocked format with a halfword length field before each record. This format corresponds to RECORDFORMAT(VB).

The data stream format for distributed transaction processing can be specified by the application program by means of the RECFM option of the BUILD ATTACH command.

Additional information on these data formats is given in "Chapter 4.8. CICS-to-IMS applications" on page 263.

SENDCOUNT and RECEIVECOUNT
These operands are used to specify whether the session is a SEND session or a RECEIVE session. (In macro-level definition, this is specified in the SESTYPE= SEND|RECEIVE operand.)

A SEND session is one in which the local CICS is the secondary and is the contention winner. It is specified by:

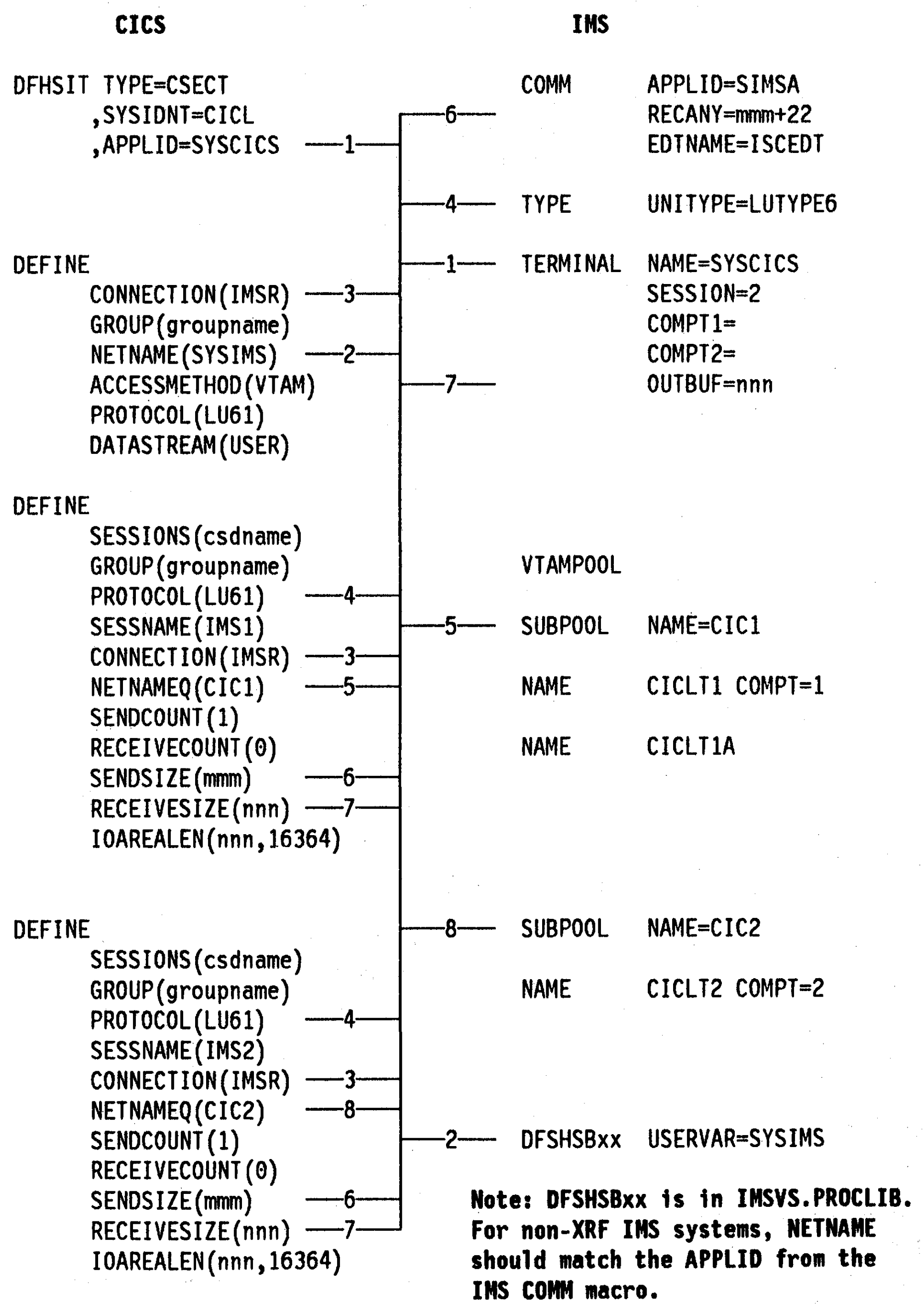
```
SENDCOUNT(1)
RECEIVECOUNT(0)
```

A RECEIVE session is one in which the local CICS is the primary and is the contention loser. It is specified by:

```
SENDCOUNT(0)
RECEIVECOUNT(1)
```

SEND sessions are recommended for all CICS-IMS sessions.

You need not specify a SENDPFX or a RECEIVEPFX; the name of the session is taken from the SESSNAME operand.

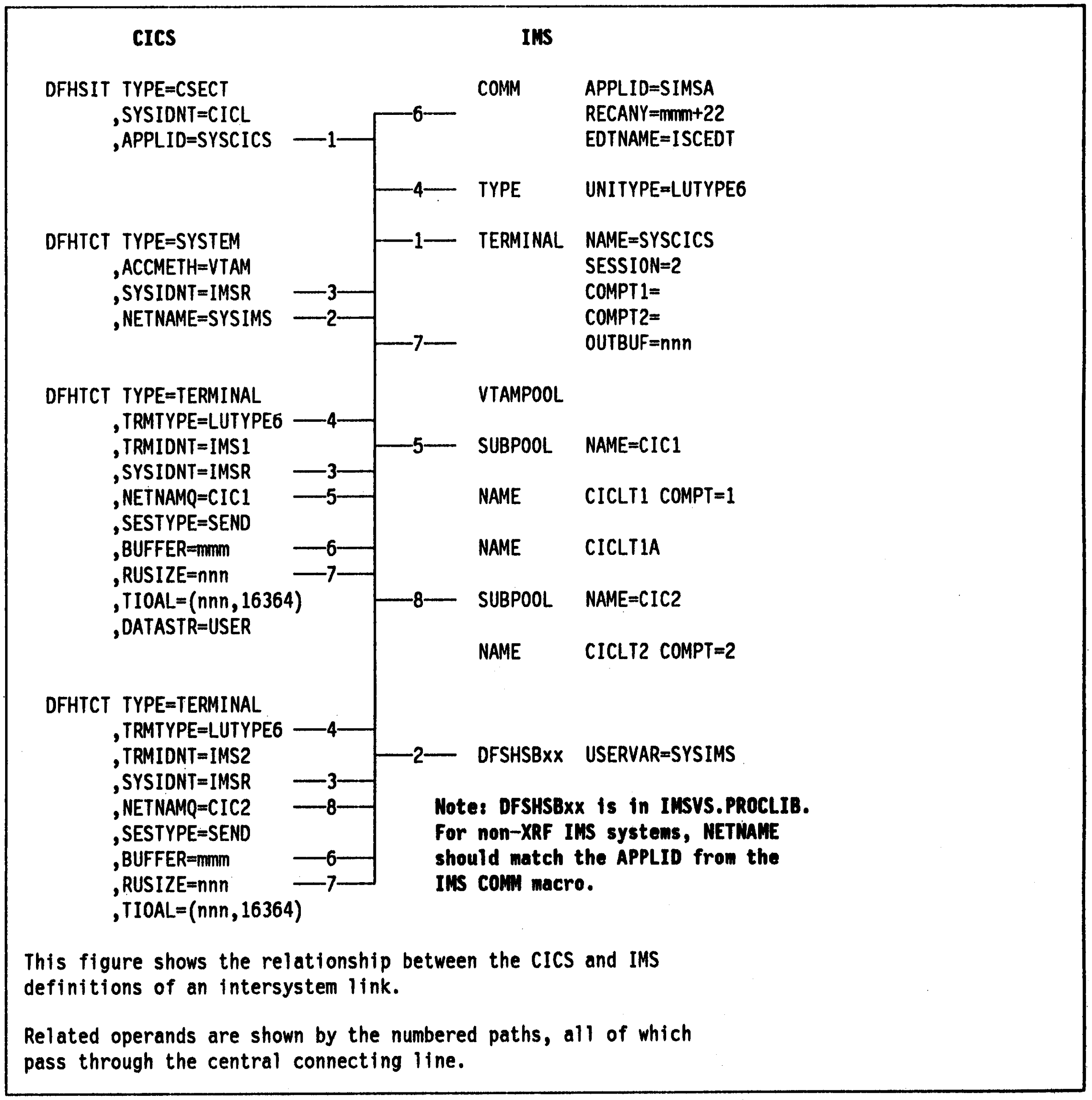


```
        CICS                                   IMS

 DFHSIT TYPE=CSECT                             COMM      APPLID=SIMSA
       ,SYSIDNT=CICL                ┌──6──               RECANY=mmm+22
       ,APPLID=SYSCICS   ──1────────┤                    EDTNAME=ISCEDT
                                    │
                                    ├──4──     TYPE      UNITYPE=LUTYPE6
                                    │
 DEFINE                             ├──1──     TERMINAL  NAME=SYSCICS
       CONNECTION(IMSR)  ──3────────┤                    SESSION=2
       GROUP(groupname)             │                    COMPT1=
       NETNAME(SYSIMS)   ──2────────┤                    COMPT2=
       ACCESSMETHOD(VTAM)           ├──7──               OUTBUF=nnn
       PROTOCOL(LU61)               │
       DATASTREAM(USER)             │
                                    │
 DEFINE                             │
       SESSIONS(csdname)            │
       GROUP(groupname)             │          VTAMPOOL
       PROTOCOL(LU61)    ──4────────┤
       SESSNAME(IMS1)               ├──5──     SUBPOOL   NAME=CIC1
       CONNECTION(IMSR)  ──3────────┤
       NETNAMEQ(CIC1)    ──5────────┤          NAME      CICLT1 COMPT=1
       SENDCOUNT(1)                 │
       RECEIVECOUNT(0)              │          NAME      CICLT1A
       SENDSIZE(mmm)     ──6────────┤
       RECEIVESIZE(nnn)  ──7────────┤
       IOAREALEN(nnn,16364)         │
                                    │
 DEFINE                             ├──8──     SUBPOOL   NAME=CIC2
       SESSIONS(csdname)            │
       GROUP(groupname)             │          NAME      CICLT2 COMPT=2
       PROTOCOL(LU61)    ──4────────┤
       SESSNAME(IMS2)               │
       CONNECTION(IMSR)  ──3────────┤
       NETNAMEQ(CIC2)    ──8────────┤
       SENDCOUNT(1)                 ├──2──     DFSHSBxx  USERVAR=SYSIMS
       RECEIVECOUNT(0)              │
       SENDSIZE(mmm)     ──6────────┤       Note: DFSHSBxx is in IMSVS.PROCLIB.
       RECEIVESIZE(nnn)  ──7────────┘       For non-XRF IMS systems, NETNAME
       IOAREALEN(nnn,16364)                 should match the APPLID from the
                                            IMS COMM macro.
```

This figure shows the relationship between the CICS and IMS definitions of an intersystem link.

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

Figure 30. Defining compatible CICS and IMS nodes — RDO



```
         CICS                                   IMS

  DFHSIT TYPE=CSECT                          COMM      APPLID=SIMSA
         ,SYSIDNT=CICL               ┌──6──            RECANY=mmm+22
         ,APPLID=SYSCICS    ──1──────┤                 EDTNAME=ISCEDT
                                     │
                                     ├──4──  TYPE      UNITYPE=LUTYPE6
                                     │
  DFHTCT TYPE=SYSTEM                 ├──1──  TERMINAL  NAME=SYSCICS
         ,ACCMETH=VTAM               │                 SESSION=2
         ,SYSIDNT=IMSR      ──3──────┤                 COMPT1=
         ,NETNAME=SYSIMS    ──2──────┤                 COMPT2=
                                     ├──7──            OUTBUF=nnn
                                     │
  DFHTCT TYPE=TERMINAL               │       VTAMPOOL
         ,TRMTYPE=LUTYPE6   ──4──────┤
         ,TRMIDNT=IMS1               ├──5──  SUBPOOL   NAME=CIC1
         ,SYSIDNT=IMSR      ──3──────┤
         ,NETNAMQ=CIC1      ──5──────┤       NAME      CICLT1 COMPT=1
         ,SESTYPE=SEND               │
         ,BUFFER=mmm        ──6──────┤       NAME      CICLT1A
         ,RUSIZE=nnn        ──7──────┤
         ,TIOAL=(nnn,16364)          ├──8──  SUBPOOL   NAME=CIC2
         ,DATASTR=USER               │
                                     │       NAME      CICLT2 COMPT=2
                                     │
  DFHTCT TYPE=TERMINAL               │
         ,TRMTYPE=LUTYPE6   ──4──────┤
         ,TRMIDNT=IMS2               ├──2──  DFSHSBxx  USERVAR=SYSIMS
         ,SYSIDNT=IMSR      ──3──────┤
         ,NETNAMQ=CIC2      ──8──────┤    Note: DFSHSBxx is in IMSVS.PROCLIB.
         ,SESTYPE=SEND               │    For non-XRF IMS systems, NETNAME
         ,BUFFER=mmm        ──6──────┤    should match the APPLID from the
         ,RUSIZE=nnn        ──7──────┘    IMS COMM macro.
         ,TIOAL=(nnn,16364)
```

This figure shows the relationship between the CICS and IMS definitions of an intersystem link.

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

Figure 31. Defining compatible CICS and IMS nodes — macro level

Defining multiple links to an IMS system

You can define more than one intersystem link between a CICS and an IMS system. This is done by defining two or more connections (systems), with their associated session definitions, with the same NETNAME but with different SYSIDNTs (Figure 32 on page 115). Although all the system definitions resolve to the same netname, and therefore to the same IMS system, the use of a SYSID name in CICS will cause CICS to allocate a session from the link with the specified SYSIDNT.

It is recommended that you define up to three links (that is, groups of sessions) between a CICS and an IMS system, depending upon the application requirements of your installation:

1. A group of sessions for CICS-initiated distributed transaction processing (synchronous processing).

   CICS applications that use the SEND/RECEIVE interface can use the SYSIDNT of this group to allocate a session to the remote system. The session will be held ("busy") until the conversation is terminated.

2. A group of sessions for CICS-initiated asynchronous processing.

   CICS applications that use the START command can name the SYSIDNT of this group. CICS will use the first "non-busy" session to ship the start request.

   IMS sends a positive response to CICS as soon as it has queued the start request, so that the session is in use for a relatively short period. Consequently, the first session in the group will show the heaviest usage, and the frequency of usage will decrease towards the last session in the group.

3. A group of sessions for IMS-initiated asynchronous processing.

   This group is also useful as part of the solution to a performance problem that can arise with CICS-initiated asynchronous processing. An IMS transaction that is initiated as a result of a start command shipped on a particular session will normally use the same session to ship its "reply" start command to CICS. For the reasons given in (2) above, the CICS start command was probably shipped on the busiest session, and, because the session is busy and CICS is the contention winner, the replies from IMS may back up waiting for a chance to use the session.

   However, facilities exist in IMS for a transaction to alter its default output session, and a switch to a session in this third group can reduce backup problems.

```
  RDO definition                   Macro-level definition

  DFHSIT TYPE=CSECT                DFHSIT TYPE=CSECT
        ,SYSIDNT=CICL                    ,SYSIDNT=CICL
        ,APPLID=SYSCICS                  ,APPLID=SYSCICS

CICS-initiated distributed transaction processing

  DEFINE CONNECTION(IMSA)          DFHTCT TYPE=SYSTEM
         ACCESSMETHOD(VTAM)              ,ACCMETH=VTAM
         NETNAME(SYSIMS)                 ,SYSIDNT=IMSA
                                         ,NETNAME=SYSIMS

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         PROTOCOL(LU61)                  ,TRMTYPE=LUTYPE6
         SESSNAME(IMS1)                  ,TRMIDNT=IMS1
         CONNECTION(IMSA)                ,SYSIDNT=IMSA
         NETNAMEQ(DTP1)                  ,NETNAMQ=DTP1

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         .                               .

CICS-initiated asynchronous processing

  DEFINE CONNECTION(IMSB)          DFHTCT TYPE=SYSTEM
         ACCESSMETHOD(VTAM)              ,ACCMETH=VTAM
         NETNAME(SYSIMS)                 ,SYSIDNT=IMSB
                                         ,NETNAME=SYSIMS

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         PROTOCOL(LU61)                  ,TRMTYPE=LUTYPE6
         SESSNAME(IMS1)                  ,TRMIDNT=IMS1
         CONNECTION(IMSB)                ,SYSIDNT=IMSB
         NETNAMEQ(ASP1)                  ,NETNAMQ=ASP1

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         .                               .

IMS-initiated asynchronous processing

  DEFINE CONNECTION(IMSC)          DFHTCT TYPE=SYSTEM
         ACCESSMETHOD(VTAM)              ,ACCMETH=VTAM
         NETNAME(SYSIMS)                 ,SYSIDNT=IMSC
                                         ,NETNAME=SYSIMS

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         PROTOCOL(LU61)                  ,TRMTYPE=LUTYPE6
         SESSNAME(IMS1)                  ,TRMIDNT=IMS1
         CONNECTION(IMSC)                ,SYSIDNT=IMSC
         NETNAMEQ(IST1)                  ,NETNAMQ=IST1

  DEFINE SESSIONS(csdname)         DFHTCT TYPE=TERMINAL
         .                               .
```

Figure 32. Defining multiple links to an IMS node

Defining logical unit type 6.2 links

An LUTYPE6.2 link consists of one or more "sets" of sessions. The sessions in each set have identical characteristics, apart from being either contention winners or contention losers. Each set of sessions can be assigned a **modename** which enables it to be mapped to a VTAM logmode name and thence to a class of service (COS). A set of LUTYPE6.2 sessions is therefore referred to as a **modeset**.

| You are recommended to use resource definition online (RDO) to define links to
| remote systems.

Note: An LUTYPE6.2 (APPC) terminal is considered to be a special case of an LUTYPE6.2 system that supports only a single session and which does not support an LU services manager. There are several ways of defining APPC terminals; further details are given under "Defining single-session APPC terminals" on page 123. This section describes the definition of one or more modesets containing more than one session.

To define a logical unit type 6.2 link to a remote system you must:

1. Resource Definition Online
   a. Use DEFINE CONNECTION to define the remote system.
   b. Use DEFINE SESSIONS to define each set of sessions to the remote system.
2. Macro-Level Definition
   a. Write a DFHTCT TYPE=SYSTEM macro to define the remote system.
   b. Write a DFHTCT TYPE=MODESET macro to define each set of sessions to the remote system.

For all LUTYPE6.2 links, except single-session links to LUTYPE6.2 terminals, CICS automatically builds a set of special sessions for the exclusive use of the LU services manager, using the modename SNASVCMG. This is a reserved name, and should not be used for any of the sets that you define.

If you are defining a VTAM logon mode table you should remember to include an entry for the SNASVCMG sessions (see "ACF/VTAM LOGMODE table entries for CICS" on page 83).

Defining the remote LUTYPE6.2 system

The RDO and macro-level forms of definition for an LUTYPE6.2 system are shown in Figure 33.

```
RDO definition                  Macro-level definition


DEFINE                          DFHTCT  TYPE=SYSTEM
  CONNECTION(name)                 ,SYSIDNT=name
  GROUP(groupname)
  ACCESSMETHOD(VTAM)               ,ACCMETH=VTAM
  PROTOCOL(APPC)                   ,TRMTYPE=LUTYPE62
  SINGLESESS(N)                    ,FEATURE=PARALLEL
  NETNAME(name)                    ,NETNAME=name
  BINDPASSWORD(password)           ,BINDPWD=password
  ATTACHSEC(LOCAL|IDENTIFY|        ,USERSEC={LOCAL|IDENTIFY|
            VERIFY)                          VERIFY}
  AUTOCONNECT(NO|YES|ALL)          ,CONNECT={AUTO|ALL}
  SECURITYNAME(value)              ,XSNAME=value

For LUTYPE6.1 applications on LUTYPE6.2

  DATASTREAM(USER|3270|            ,DATASTR={USER|3270|
             SCS|STRFIELD|                   SCS|STRFIELD|
             LMS)                            LMS}
  RECORDFORMAT(U|VB)               ,RECFM={U|VB}
```

Figure 33. Defining an LUTYPE6.2 system

You must specify ACCESSMETHOD(VTAM) and PROTOCOL(APPC) to define an LUTYPE6.2 system. The CONNECTION name (that is, the sysidnt) and the NETNAME name have the meanings explained in "Identifying remote systems" on page 93.

Because this connection will have multiple sessions, you must specify SINGLESESS(N), or allow it to default. (The definition of single-session APPC terminals is described in "Defining single-session APPC terminals" on page 123.)

The AUTOCONNECT operand specifies which of the sessions that are associated with the connection are to bound when CICS is initialized. Further information is given in "The AUTOCONNECT operand" on page 125.

If the intersystem link is to be used by existing applications that were designed to run on LUTYPE6.1 links, you can use the DATASTREAM and RECORDFORMAT operands to specify data stream information for asynchronous processing. The information provided by these operands is not used by LUTYPE6.2 application programs.

Defining groups of LUTYPE6.2 sessions

Each group of sessions for an LUTYPE6.2 system is defined by means of a DEFINE SESSIONS command (RDO) or a DFHTCT TYPE=MODESET macro (macro-level definition). The two forms of definition are shown in Figure 34.

Each individual group of sessions is referred to as a **modeset**.

```
RDO definition                    Macro-level definition


DEFINE                           DFHTCT  TYPE=MODESET
   SESSIONS(csdname)
   GROUP(groupname)
   PROTOCOL(APPC)
   CONNECTION(name)                 ,SYSIDNT=name
   MODENAME(name)                   ,MODENAM=name
   MAXIMUM(m1,m2)                   ,MAXSESS=(m1,m2)
   AUTOCONNECT(NO|YES|ALL)          ,CONNECT={AUTO|ALL}

   SENDSIZE(size)                  [,BUFFER=size]
   RECEIVESIZE(size)¹              [,RUSIZE=size¹]
   OPERID(operator-id)             [,OPERID=operator-id]
   OPERPRIORITY(number)            [,OPERPRI=number]
   OPERRSL(number)                 [,OPERRSL=number]
   OPERSECURITY(number)            [,OPERSEC=number]
   USERAREALEN(value)              [,TCTUAL=value]
   SESSPRIORITY(number)            [,TRMPRTY=number]
   TRANSACTION(name)               [,TRANSID=name]
                                   [,TRMSTAT=TRANSCEIVE]

 ¹ Minimum value 256.  In the SIT, the RAMAX value must be
  greater than or equal to these values.
```

Figure 34. Defining a group of LUTYPE6.2 sessions

The CONNECTION operand specifies the one to four character name of the LUTYPE6.2 system for which the group is being defined; that is, the CONNECTION name in the associated DEFINE CONNECTION command. Note that, for macro-level definition, the associated TYPE=SYSTEM macro must be coded immediately before any TYPE=MODESET macros that refer to it.

The MODENAME operand enables you to specify a one-to eight-character name that is to identify this group of related sessions. The name must be unique among the modenames for any one LUTYPE6.2 intersystem link, and you must not use the reserved name SNASVCMG.

The MAXIMUM(m1,m2) operand specifies the maximum number of sessions that are to be supported for the group. The parameters of this operand have the following meanings:

m1
: specifies the maximum number of sessions in the group. The default value is 1.

m2
: specifies the maximum number of sessions that are to be supported as contention winners. The number specified for m2 must not be greater than the number specified for m1. The default value for m2 is zero.

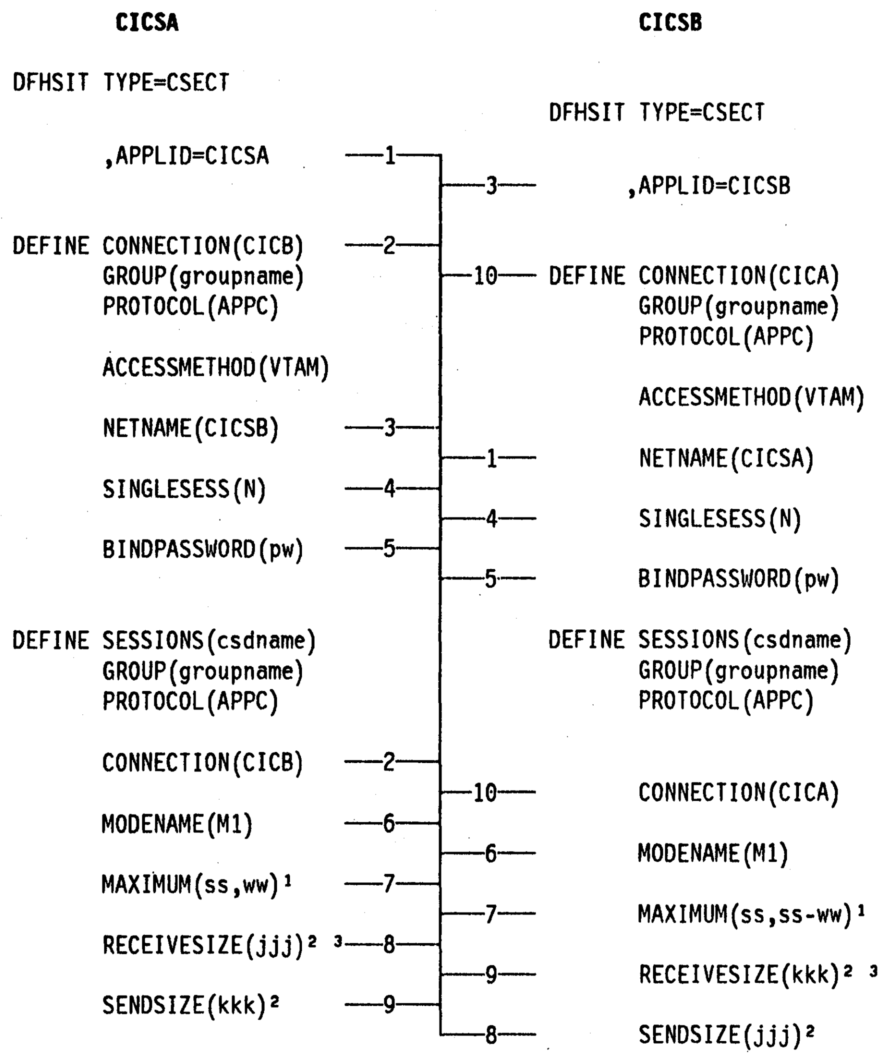
The AUTOCONNECT operand specifies whether the sessions are to bound when CICS is initialized. Further information is given in "The AUTOCONNECT operand" on page 125.

For macro-level definition, the operands shown in brackets ([ ]) can also be coded on the TYPE=SYSTEM macro to provide default values for all the associated modesets.

Defining compatible CICS LUTYPE6.2 nodes

When you are defining an LUTYPE6.2 link between two CICS systems, you must ensure that the definitions of the link in each of the systems are compatible.

The compatibility requirements are summarized in Figure 35 on page 120 (RDO used in both systems), Figure 36 on page 121 (macro definition used in both systems), and Figure 37 on page 122 (RDO used in one system and macro definition used in the other).



```
            CICSA                                         CICSB

     DFHSIT TYPE=CSECT
                                                  DFHSIT TYPE=CSECT

            ,APPLID=CICSA     ──1──┐
                                   ├──3──           ,APPLID=CICSB

     DEFINE CONNECTION(CICB)  ──2──┤
            GROUP(groupname)       ├──10── DEFINE CONNECTION(CICA)
            PROTOCOL(APPC)         │              GROUP(groupname)
                                   │              PROTOCOL(APPC)
            ACCESSMETHOD(VTAM)     │
                                   │              ACCESSMETHOD(VTAM)
            NETNAME(CICSB)    ──3──┤
                                   ├──1──         NETNAME(CICSA)
            SINGLESESS(N)     ──4──┤
                                   ├──4──         SINGLESESS(N)
            BINDPASSWORD(pw)  ──5──┤
                                   ├──5──         BINDPASSWORD(pw)

     DEFINE SESSIONS(csdname)      │       DEFINE SESSIONS(csdname)
            GROUP(groupname)       │              GROUP(groupname)
            PROTOCOL(APPC)         │              PROTOCOL(APPC)

            CONNECTION(CICB)  ──2──┤
                                   ├──10──        CONNECTION(CICA)
            MODENAME(M1)      ──6──┤
                                   ├──6──         MODENAME(M1)
            MAXIMUM(ss,ww)¹   ──7──┤
                                   ├──7──         MAXIMUM(ss,ss-ww)¹
            RECEIVESIZE(jjj)² ³──8──┤
                                   ├──9──         RECEIVESIZE(kkk)² ³
            SENDSIZE(kkk)²    ──9──┤
                                   └──8──         SENDSIZE(jjj)²
```

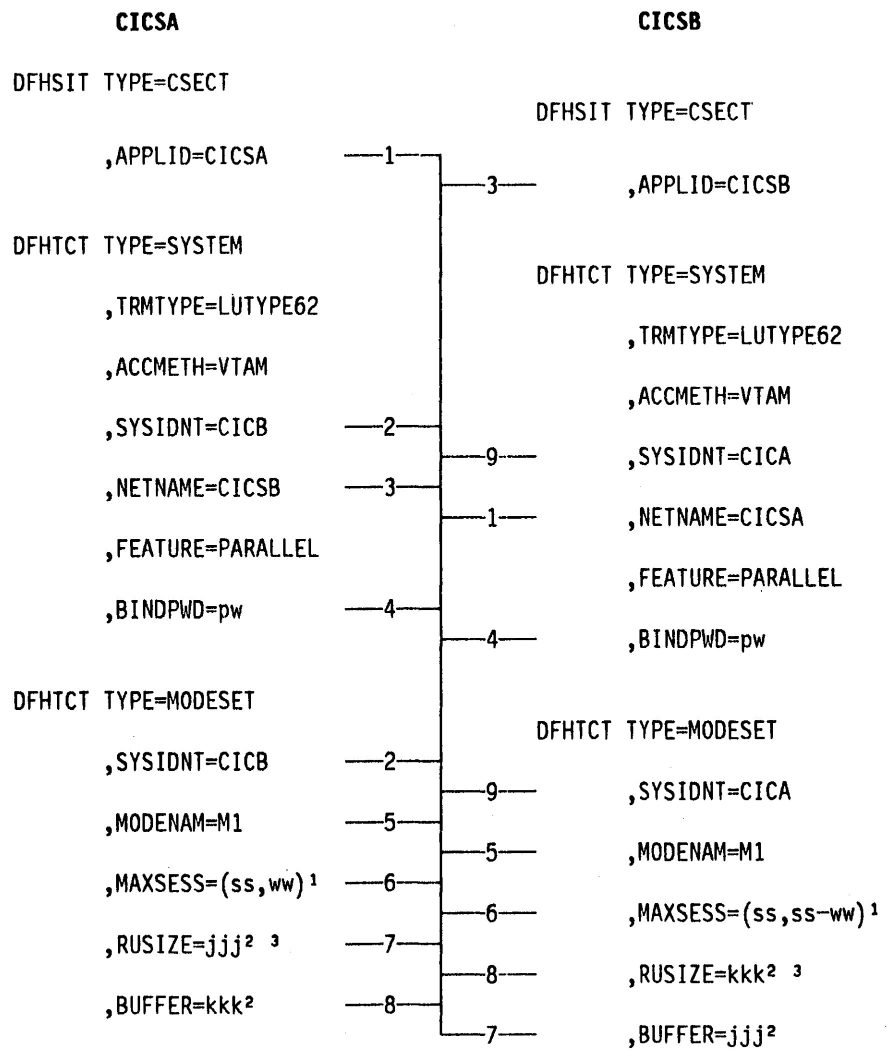
Related operands are shown by the numbered paths, all of which pass through the central connecting line.

[1] These values need not match, because they are negotiated by the LU services managers. However, a matching specification will avoid unusable TCTTE entries, and will also avoid unexpected bidding because of the "contention winners" negotiation.

[2] CICS will negotiate these values at BIND time if they do not match.

| [3] These values must greater than or equal to 256.

Figure 35. Defining compatible CICS LUTYPE6.2 ISC nodes — RDO



```
        CICSA                                        CICSB

DFHSIT TYPE=CSECT
                                          DFHSIT TYPE=CSECT
        ,APPLID=CICSA     —1—┐
                             ├—3—               ,APPLID=CICSB
DFHTCT TYPE=SYSTEM           │
                             │            DFHTCT TYPE=SYSTEM
        ,TRMTYPE=LUTYPE62    │
                             │                  ,TRMTYPE=LUTYPE62
        ,ACCMETH=VTAM        │
                             │                  ,ACCMETH=VTAM
        ,SYSIDNT=CICB     —2—┤
                             ├—9—               ,SYSIDNT=CICA
        ,NETNAME=CICSB    —3—┤
                             ├—1—               ,NETNAME=CICSA
        ,FEATURE=PARALLEL    │
                             │                  ,FEATURE=PARALLEL
        ,BINDPWD=pw       —4—┤
                             ├—4—               ,BINDPWD=pw
DFHTCT TYPE=MODESET          │
                             │            DFHTCT TYPE=MODESET
        ,SYSIDNT=CICB     —2—┤
                             ├—9—               ,SYSIDNT=CICA
        ,MODENAM=M1       —5—┤
                             ├—5—               ,MODENAM=M1
        ,MAXSESS=(ss,ww)¹ —6—┤
                             ├—6—               ,MAXSESS=(ss,ss-ww)¹
        ,RUSIZE=jjj² ³    —7—┤
                             ├—8—               ,RUSIZE=kkk² ³
        ,BUFFER=kkk²      —8—┤
                             └—7—               ,BUFFER=jjj²
```

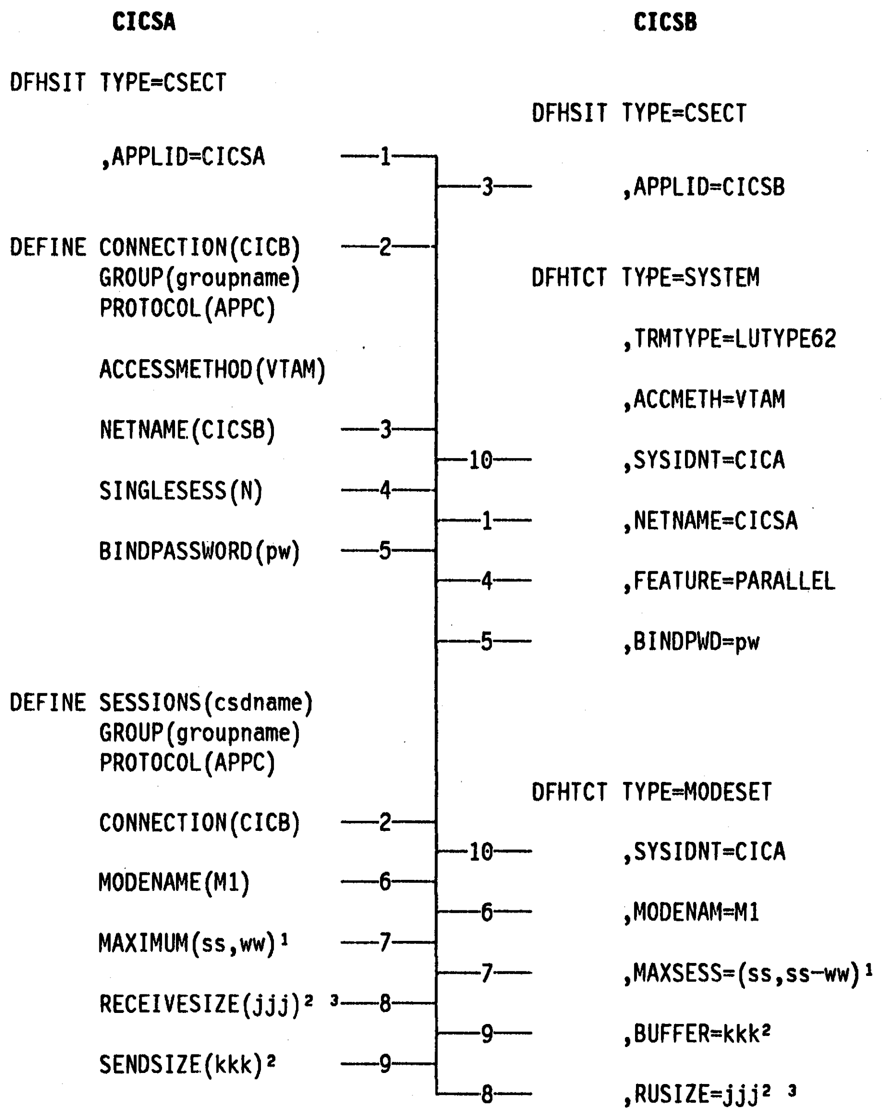
Related operands are shown by the numbered paths, all of which pass through the central connecting line.

[1] MAXSESS need not match, because this parameter is negotiated by the LU services managers. However, a matching specification will avoid unusable TCTTE entries, and will also avoid unexpected bidding because of the "contention winners" negotiation.

[2] CICS will negotiate RUSIZE and BUFFER at BIND time if they do not match.

| [3] These values must greater than or equal to 256.

Figure 36. Defining compatible CICS LUTYPE6.2 ISC nodes — macro-level



```
         CICSA                                       CICSB

DFHSIT TYPE=CSECT
                                          DFHSIT TYPE=CSECT

       ,APPLID=CICSA        ──1──┐
                                 ├──3──          ,APPLID=CICSB

DEFINE CONNECTION(CICB)     ──2──┤
       GROUP(groupname)          │        DFHTCT TYPE=SYSTEM
       PROTOCOL(APPC)            │
                                 │               ,TRMTYPE=LUTYPE62
       ACCESSMETHOD(VTAM)        │
                                 │               ,ACCMETH=VTAM
       NETNAME(CICSB)       ──3──┤
                                 ├──10──         ,SYSIDNT=CICA
       SINGLESESS(N)        ──4──┤
                                 ├──1──          ,NETNAME=CICSA
       BINDPASSWORD(pw)     ──5──┤
                                 ├──4──          ,FEATURE=PARALLEL
                                 │
                                 ├──5──          ,BINDPWD=pw
                                 │
DEFINE SESSIONS(csdname)         │
       GROUP(groupname)          │
       PROTOCOL(APPC)            │
                                 │        DFHTCT TYPE=MODESET
       CONNECTION(CICB)     ──2──┤
                                 ├──10──         ,SYSIDNT=CICA
       MODENAME(M1)         ──6──┤
                                 ├──6──          ,MODENAM=M1
       MAXIMUM(ss,ww)¹      ──7──┤
                                 ├──7──          ,MAXSESS=(ss,ss-ww)¹
       RECEIVESIZE(jjj)² ³──8──┤
                                 ├──9──          ,BUFFER=kkk²
       SENDSIZE(kkk)²       ──9──┤
                                 └──8──          ,RUSIZE=jjj² ³
```

Related operands are shown by the numbered paths, all of which pass through the central connecting line.

[1] These values need not match, because they are negotiated by the LU services managers. However, a matching specification will avoid unusable TCTTE entries, and will also avoid unexpected bidding because of the "contention winners" negotiation.

[2] CICS will negotiate these values at BIND time if they do not match.

| [3] These values must greater than or equal to 256.

Figure 37. Defining compatible CICS LUTYPE6.2 ISC nodes — mixed definition

Defining single-session APPC terminals

There are three ways of defining a single-session APPC terminal, two using RDO, and one using a macro. The recommended method is an RDO TERMINAL-TYPETERM pair.

Resource definition online: Two methods are available with RDO. You can define a TERMINAL-TYPETERM pair, or you can define a CONNECTION-SESSIONS pair, with SINGLESESS(Y) specified for the connection.

Macro-level definition: You code a single DFHTCT TYPE=SYSTEM macro to define both the APPC terminal and its single session; you must not code a DFHTCT TYPE=MODESET macro.

No matter how it is defined, an APPC terminal is always represented by a system entry (TCTSE) in the terminal control table, and its single session is represented by a terminal entry (TCTTE) in the terminal control table.

Resource definition online — TERMINAL-TYPETERM pair

You can define an APPC terminal as a TERMINAL with an associated TYPETERM. This method of definition has two principal advantages:

1. You can use a single TYPETERM for all your APPC terminals of the same type.
2. It makes the AUTOINSTALL facility available for APPC terminals.

The basic method for defining an APPC terminal is as follows:

```
DEFINE TERMINAL(sysid)
       MODENAME(modename)
       TYPETERM(typeterm)
       .
       .

DEFINE TYPETERM(typeterm)
       DEVICE(APPC)
       .
       .
```

Note that, because all LUTYPE6.2 devices are seen as systems by CICS, the name in the TERMINAL operand is effectively a system name. You would, for example, use CEMT INQUIRE CONNECTION, not CEMT INQUIRE TERMINAL, to inquire about an APPC terminal.

A single, contention-winning, session is implied by DEFINE TERMINAL. However, for APPC terminals, CICS will accept a negotiated bind in which it is changed to contention loser.

The CICS-supplied CSD group DFHTYPE contains a TYPETERM, DFHLU62T, suitable for APPC terminals. You can either use this TYPETERM as it stands, or use it as the basis for your own definition.

If you plan to use automatic installation for your APPC terminals, you will need the model terminal definition (LU62) that is provided in the CICS-supplied CSD group DFHTERM. You will also have to write an autoinstall user program, and provide suitable VTAM LOGMODE entries.

For details of TERMINAL/TYPETERM definition, for details of the CICS-supplied CSD groups, and for an introduction to automatic installation, see the *CICS/MVS Resource Definition (Online)* manual. For details of autoinstall user programs and VTAM LOGMODE entries, see the *CICS/MVS Customization Guide*.

Resource definition online — CONNECTION-SESSIONS pair

You can define a CONNECTION-SESSIONS pair to represent an APPC terminal.

The forms of DEFINE CONNECTION and DEFINE SESSIONS commands that are required are similar to those shown in Figure 33 on page 117 and Figure 34 on page 118. The differences are shown below:

```
DEFINE CONNECTION(sysidnt)
       .
       SINGLESESS(Y)
       .


DEFINE SESSIONS(csdname)
       .
       MAXIMUM(1,1)
       .
```

You **must** specify SINGLESESS(Y) for the connection. The MAXIMUM operand must specify only one session, and you are strongly recommended to make it a contention winner (as shown). CICS will then attempt to bind as a contention winner, but will accept a negotiated bind in which it is changed to the contention loser.

Macro-level definition

You can define an APPC terminal by means of a single DFHTCT TYPE=SYSTEM macro:

```
DFHTCT  TYPE=SYSTEM
    ,SYSIDNT=name
    ,ACCMETH=VTAM
    ,TRMTYPE=LUTYPE62
    ,FEATURE=SINGLE
    ,MODENAME=modename
    ,NETNAME=name
    ,BINDPWD=password
    ,USERSEC={LOCAL|IDENTIFY|VERIFY}
    ,CONNECT={AUTO|ALL}
    ,XSNAME=value
```

You **must** specify FEATURE=SINGLE. Optionally, you can specify a MODENAME for the session.

In addition to the operands shown, you can specify any of the optional session-related operands that are allowed on DFHTCT TYPE=MODESET (see Figure 34 on page 118).

The AUTOCONNECT operand

The AUTOCONNECT operand of DEFINE CONNECTION and DEFINE SESSIONS maps to the CONNECT operand of DFHTCT TYPE=SYSTEM and DFHTCT TYPE=MODESET in the following way:

```
AUTOCONNECT(NO)        Omit CONNECT operand
AUTOCONNECT(YES)       CONNECT=AUTO
AUTOCONNECT(ALL)       CONNECT=ALL
```

You can use the AUTOCONNECT operand of DEFINE CONNECTION and DEFINE SESSIONS (and of DEFINE TYPETERM for APPC terminals) to control when CICS attempts to establish communication with the remote LUTYPE6.2 system.

Except for single-session APPC terminals (see "Defining single-session APPC terminals" on page 123), two events are necessary to establish sessions to a remote LUTYPE6.2 system.

1. The connection to the remote system must be established. This effectively means binding the LU services manager sessions (SNASVCMG) and carrying out initial negotiations.
2. The sessions of the modeset in question must be bound.

These events are controlled in part by the AUTOCONNECT operand of the DEFINE CONNECTION command and in part by the AUTOCONNECT of the DEFINE SESSIONS command.

The AUTOCONNECT operand of DEFINE CONNECTION

On the DEFINE CONNECTION command, the AUTOCONNECT operand specifies whether CICS is to attempt to bind the LU services manager sessions at the earliest opportunity (when the VTAM ACB is opened). It has the following meanings:

AUTOCONNECT(NO)
specifies that CICS **is not** to attempt to bind the LU services manager sessions.

AUTOCONNECT(YES)
specifies that CICS **is** to attempt to bind the LU services manager sessions.

AUTOCONNECT(ALL)
the same as YES; you could, however, use it as a reminder that the associated DEFINE SESSIONS specify ALL.

The LU services manager sessions cannot be bound if the remote system is not available. If for any reason they are not bound during CICS initialization, they can be bound by means of a CEMT SET CONNECTION INSERVICE ACQUIRED command. They are also bound if the remote system itself initiates communication. For a single-session APPC terminal, specifying AUTOCONNECT(YES) or AUTOCONNECT(ALL) on the DEFINE CONNECTION command has no effect. This is because a single-session connection has no LU services manager.

The AUTOCONNECT operand of DEFINE SESSIONS

On the DEFINE SESSIONS command, the AUTOCONNECT operand specifies which sessions are to be bound when the associated LU services manager sessions have been bound. (No user sessions can be bound before this time.)

The operand has the following meanings:

AUTOCONNECT(NO)
specifies that no sessions are to be bound.

AUTOCONNECT(YES)
specifies that the contention-winning sessions are to be bound.

AUTOCONNECT(ALL)
specifies that the contention-winning and the contention-losing sessions are to be bound.

AUTOCONNECT(ALL) allows CICS to bind contention-losing sessions with remote systems that **cannot** send bind requests.

Never specify AUTOCONNECT(ALL) for sessions to another CICS system, or to any system that may send a bind request. This could lead to bind-race conditions that CICS cannot resolve.

If AUTOCONNECT(NO) is specified, the sessions can be bound by means of a CEMT SET MODENAME ACQUIRED command. If this is not done, sessions are bound individually according to the demands of your application program.

For a single-session APPC terminal, the value specified for AUTOCONNECT on DEFINE SESSIONS or DEFINE TYPETERM determines whether CICS attempts to bind the single session or not.

Note: Specifying AUTOCONNECT(ALL) may cause CICS to bind a number of contention winners other than the number originally specified in this system. This depends on the partner system's reply to the request to initiate sessions (CNOS EXCHANGE). CICS attempts to bind as contention winners any sessions that are not designated as contention losers in the CNOS reply.

Indirect links for transaction routing

Indirect links allow transaction routing between two CICS systems even though you have not defined a direct link between them. The only requirement is that there is a path from one system to the other via one or more intermediate systems.

The following figure illustrates the concept of an indirect link.

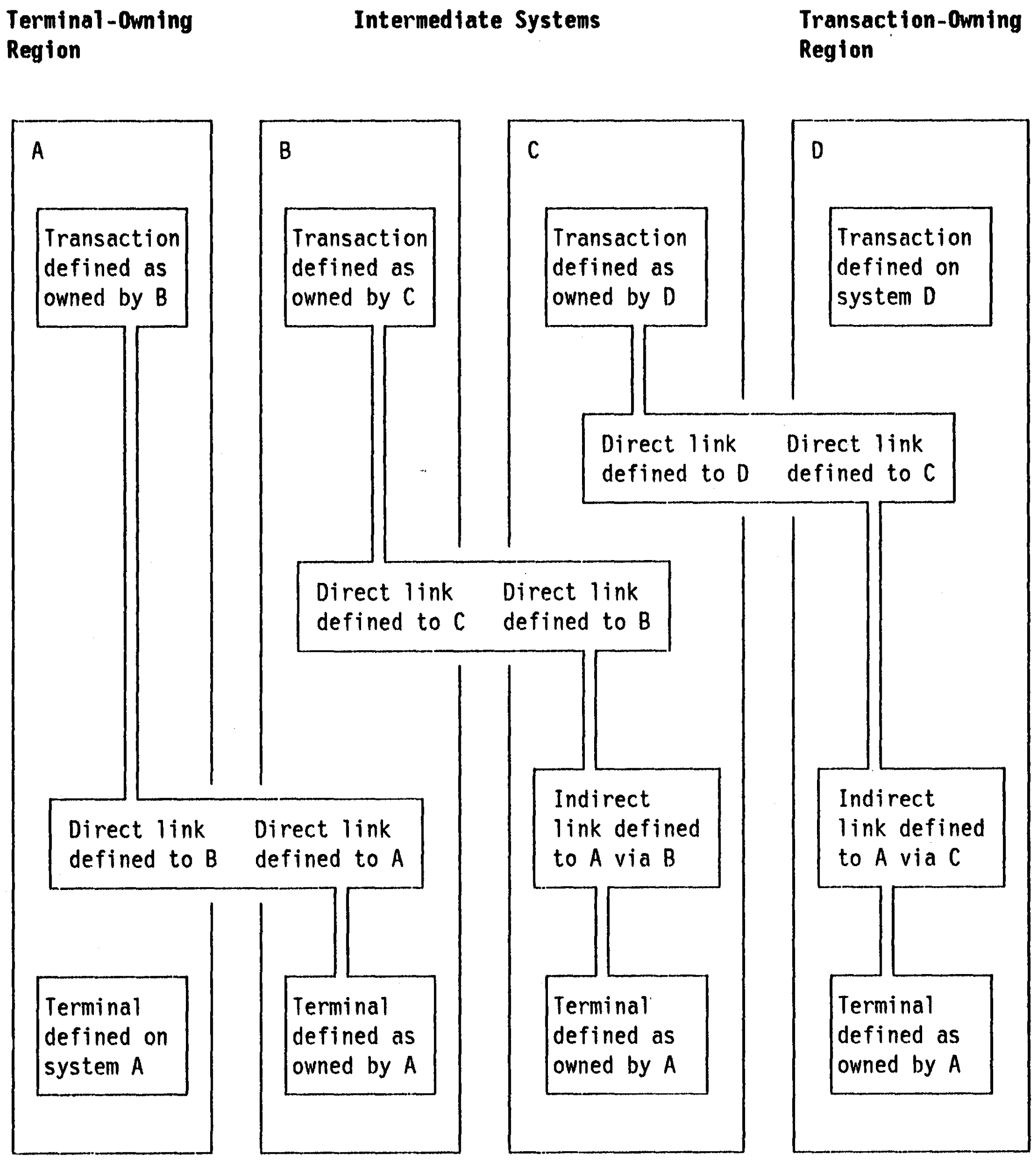

Terminal-Owning Region
Intermediate Systems
Transaction-Owning Region
A
B
C
D
Transaction defined as owned by B
Transaction defined as owned by C
Transaction defined as owned by D
Transaction defined on system D
Direct link defined to D
Direct link defined to C
Direct link defined to C
Direct link defined to B
Direct link defined to B
Direct link defined to A
Indirect link defined to A via B
Indirect link defined to A via C
Terminal defined on system A
Terminal defined as owned by A
Terminal defined as owned by A
Terminal defined as owned by A

Figure 38. Transaction routing via indirect links

This figure illustrates a chain of systems (A, B, C, D) linked by MRO or LUTYPE6.2 links (you cannot do transaction routing over LUTYPE6.1 links).

It is assumed that you want to establish a transaction-routing path between a terminal-owning region A and an application-owning region D. There is no direct link available between region A and region D, but a path is available via the **intermediate** regions B and C.

To enable transaction-routing requests to pass along the path, resource definitions for both the terminal and the transaction must be available in all four regions. The terminal is a local resource in the terminal-owning-region A, and a remote resource in systems B, C, and D. Similarly, the transaction is a local resource in the application-owning-region D, and a remote resource in the regions A, B, and C. The definition of remote terminals and transactions is described in "Chapter 3.2. Defining remote resources" on page 133.

Note: The transaction routing path between the terminal and the transaction must not turn back on itself. If, for example, in Figure 38 on page 127, the transaction definition in system D is replaced by a remote definition of a transaction in system C, the attempt to use the transaction from system A will be abended when system D tries to route back to system C.

Why indirect links are required

As explained in "Chapter 3.2. Defining remote resources" on page 133, CICS systems reference remote terminals by means of a unique identifier that is formed from:

1. The APPLID of the terminal-owning region.
2. The identifier by which the terminal is known on the terminal-owning region.

To enable CICS to form the fully-qualified terminal identifier, a remote terminal definition must specify the system identifier of a link whose NETNAME is the APPLID of the terminal-owning region.

If there is no direct link with the required netname, an indirect link must be defined.

The indirect link definition has two purposes:

1. It specifies the NETNAME of the terminal-owning region.
2. It identifies the **direct** link that is the start of the path to the terminal-owning region.

Thus, in Figure 38 on page 127, the indirect link definition in region D provides the NETNAME of region A and identifies region C as the next region in the path. Similarly, the indirect link definition in region C provides the NETNAME of region A and identifies region B as the next region in the path. Region B has a direct link to region A, and therefore does not require an indirect link.

Resource definition for indirect transaction routing

This section outlines the definitions required to establish a transaction-routing path between a terminal-owning region SYS01 and an application-owning region SYS04 via two intermediate regions SYS02 and SYS03.

The definitions required are shown in Figure 39 on page 129 (resource definition online) and Figure 40 on page 131 (macro-level definition). You can, of course, use any combination of resource definition online and macro-level definition for the various resources.

```
SYS01                  SYS02                  SYS03                  SYS04

DFHSIT                 DFHSIT                 DFHSIT                 DFHSIT
APPLID=SYS01           APPLID=SYS02           APPLID=SYS03           APPLID=SYS04
.                      .                      .                      .

      Link between SYS01 and SYS02                  Link between SYS03 and SYS04

DEFINE             DEFINE                      DEFINE             DEFINE
CONNECTION(NEXT)   CONNECTION(PREV)            CONNECTION(NEXT)   CONNECTION(PREV)
NETNAME(SYS02)     NETNAME(SYS01)              NETNAME(SYS04)     NETNAME(SYS03)
.                  .                           .                  .

DEFINE             DEFINE                      DEFINE             DEFINE
SESSIONS(csdname)  SESSIONS(csdname)           SESSIONS(csdname)  SESSIONS(csdname)
CONNECTION(NEXT)   CONNECTION(PREV)            CONNECTION(NEXT)   CONNECTION(PREV)
.                  .                           .                  .

                          Link between SYS02 and SYS03        Indirect Link from SYS04 to
                                                              SYS01, routed via SYS03
                       DEFINE             DEFINE
                       CONNECTION(NEXT)   CONNECTION(PREV)    DEFINE
                       NETNAME(SYS03)     NETNAME(SYS02)      CONNECTION(REMT)
                       .                  .                   NETNAME  (SYS01)
                                                              ACCESSMETHOD
                       DEFINE             DEFINE                 (INDIRECT)
                       SESSIONS(csdname)  SESSIONS(csdname)   INDSYS(PREV)
                       CONNECTION(NEXT)   CONNECTION(PREV)
                       .                  .

                                          Indirect Link from
                                          SYS03 to SYS01
                                          routed via SYS02

                                          DEFINE
                                          CONNECTION(REMT)
                                          NETNAME(SYS01)
                                          ACCESSMETHOD
                                             (INDIRECT)
                                          INDSYS(PREV)

  The Terminal          The Terminal          The Terminal          The Terminal

DEFINE                DEFINE                DEFINE                DEFINE
TERMINAL(T42A)        TERMINAL(T42A)        TERMINAL(T42A)        TERMINAL(T42A)
NETNAME(XXXXX)        REMOTESYSTEM(PREV)    REMOTESYSTEM(REMT)    REMOTESYSTEM(REMT)
TYPETERM(DFHLU2)      TYPETERM(DFHLU2)      TYPETERM(DFHLU2)      TYPETERM(DFHLU2)
.                     .                     .                     .

The Transaction       The Transaction       The Transaction       The Transaction

DEFINE                DEFINE                DEFINE                DEFINE
TRANSACTION(TRTN)     TRANSACTION(TRTN)     TRANSACTION(TRTN)     TRANSACTION(TRTN)
REMOTESYSTEM(NEXT)    REMOTESYSTEM(NEXT)    REMOTESYSTEM(NEXT)    PROGRAM(TRNP)
.                     .                     .                     .
```

Figure 39. Defining indirect links for transaction routing — RDO

Defining the direct links

The direct links between SYS01 and SYS02, SYS02 and SYS03, and SYS03 and SYS04 are MRO or LUTYPE6.2 links defined as described earlier in this chapter.

Defining the indirect links

An indirect link for transaction routing must be defined in every system in a transaction-routing path from a terminal-owning region to an application-owning region, except for the terminal-owning region itself and the first region in the path (that is, the region to which the terminal-owning region has a direct link).

In the current example, therefore, indirect links must be defined in SYS04 and SYS03. The following rules apply to the definition of an indirect link:

1. Resource Definition Online
   a. The ACCESSMETHOD must be INDIRECT.
   b. The NETNAME must be the APPLID of the terminal-owning region.
   c. INDSYS (meaning indirect system) must name the CONNECTION name of an MRO or LUTYPE6.2 link that is the start of the path to the terminal-owning region.
   d. No SESSIONS definition is required for the indirect connection; the sessions that are used are those of the direct link named in the INDSYS operand.
2. Macro-Level Definition
   a. The ACCMETH must be INDIRECT.
   b. The NETNAME must be the APPLID of the terminal-owning region.
   c. INDSYS (meaning indirect system) must name the SYSIDNT name of an MRO or LUTYPE6.2 link that is the start of a path to the terminal-owning region.
   d. No session-related operands are required for the indirect link definition; the sessions that are used are those of the direct link named in the INDSYS operand.

Shippable terminals

Transaction routing ships CEDA-installed or autoinstalled terminals across indirect links. However, the global user exits XALTENF and XICTENF (see "Shipping terminals for automatic transaction initiation" on page 48) can be used only if the TOR and AOR are connected directly.

Defining the terminal

Unless you plan to use **shippable** terminal definitions (see "Shipping terminal definitions" on page 141), the terminal must be defined as a remote resource in every region in the transaction-routing path except the terminal-owning region itself.

If you do use shippable terminal definitions, you must still define all the necessary indirect links.

SYS01

```
DFHSIT
APPLID=SYS01
.
```

SYS02

```
DFHSIT
APPLID=SYS02
.
```

SYS03

```
DFHSIT
APPLID=SYS03
.
```

SYS04

```
DFHSIT
APPLID=SYS04
.
```

Link between SYS01 and SYS02

```
DFHTCT               DFHTCT
TYPE=SYSTEM,         TYPE=SYSTEM,
SYSIDNT=NEXT,        SYSIDNT=PREV,
NETNAME=SYS02,       NETNAME=SYS01,
.                    .
```

Link between SYS03 and SYS04

```
DFHTCT               DFHTCT
TYPE=SYSTEM,         TYPE=SYSTEM,
SYSIDNT=NEXT,        SYSIDNT=PREV,
NETNAME=SYS04,       NETNAME=SYS03,
.
```

Link between SYS02 and SYS03

```
DFHTCT               DFHTCT
TYPE=SYSTEM,         TYPE=SYSTEM,
SYSIDNT=NEXT,        SYSIDNT=PREV,
NETNAME=SYS03,       NETNAME=SYS02,
.                    .
```

Indirect Link from SYS04 to SYS01 routed via SYS03

```
DFHTCT
TYPE=SYSTEM,
ACCMETH=INDIRECT,
SYSIDNT=REMT,
NETNAME=SYS01,
INDSYS=PREV
```

Indirect Link from SYS03 to SYS01 routed via SYS02

```
DFHTCT
TYPE=SYSTEM,
ACCMETH=INDIRECT,
SYSIDNT=REMT,
NETNAME=SYS01,
INDSYS=PREV
```

The Terminal (SYS01)

```
DFHTCT
TYPE=TERMINAL,
TRMIDNT=T42A,
NETNAME=      ,
.
```

The Terminal (SYS02)

```
DFHTCT
TYPE=REMOTE,
TRMIDNT=T42A,
SYSIDNT=PREV,
.
```

The Terminal (SYS03)

```
DFHTCT
TYPE=REMOTE,
TRMIDNT=T42A,
SYSIDNT=REMT,
.
```

The Terminal (SYS04)

```
DFHTCT
TYPE=REMOTE,
TRMIDNT=T42A,
SYSIDNT=REMT,
.
```

The Transaction (SYS01)

```
DFHPCT
TYPE=REMOTE,
TRANSID=TRTN,
SYSIDNT=NEXT
.
```

The Transaction (SYS02)

```
DFHPCT
TYPE=REMOTE,
TRANSID=TRTN,
SYSIDNT=NEXT
.
```

The Transaction (SYS03)

```
DFHPCT
TYPE=REMOTE,
TRANSID=TRTN,
SYSIDNT=NEXT
.
```

The Transaction (SYS04)

```
DFHPCT
TYPE=ENTRY,
TRANSID=TRTN,
PROGRAM=TRNP,
.
```

Figure 40. Defining indirect links for transaction routing — macro

The definition of remote terminals is described in "Chapter 3.2. Defining remote resources" on page 133. The REMOTESYSTEM (or SYSIDNT) operand in a remote terminal definition must always name a link whose NETNAME is the APPLID of the terminal-owning region. The named link must be the direct link to the terminal-owning region if one exists. Otherwise, it must be an indirect link.

Defining the transaction

The transaction must be defined as a remote resource in every region in the transaction-routing path except the application-owning region itself. In all cases, the REMOTESYSTEM (or SYSIDNT) operand must name a direct link to the next region in the transaction-routing path. (The definition of remote transactions is described in "Chapter 3.2. Defining remote resources" on page 133.)

Multiple transaction-routing paths

If you have a chain of three or more regions, you may wish to allow transaction-routing between any pair of regions in the chain. The link definitions that you will require in this case are shown in the following figure:

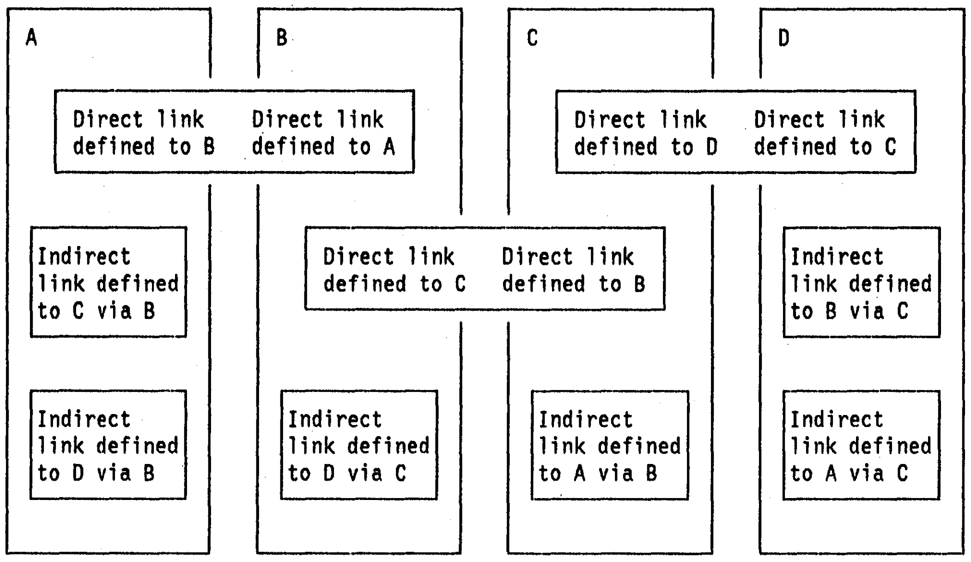

A
B
C
D
Direct link defined to B
Direct link defined to A
Direct link defined to D
Direct link defined to C
Indirect link defined to C via B
Direct link defined to C
Direct link defined to B
Indirect link defined to B via C
Indirect link defined to D via B
Indirect link defined to D via C
Indirect link defined to A via B
Indirect link defined to A via C

Figure 41. Multiple indirect links for transaction routing

For your terminal and transaction definitions, you must consider each possible transaction-routing path in turn and apply the rules that have be explained previously:

1. Remote terminal definitions must always refer to a link definition that specifies the NETNAME of the system that owns the terminal.
2. Remote transaction definitions must always name the next system in the path to the system that owns the transaction; they must not name an indirect link.

Chapter 3.2. Defining remote resources

Remote resources are resources that reside on a remote system but which need to be accessed by the local CICS system. In general, you will have to define all these resources in your local CICS system, in much the same way as you define your local resources, by using CICS resource definition macros or, for remote transactions and VTAM terminals only, by resource definition online (RDO).

This chapter tells you how to define the remote resources that may be required for CICS function shipping, CICS transaction routing, and asynchronous processing (START command shipping). No remote resource definition is required for distributed transaction processing.

The remote resources that can be defined are:

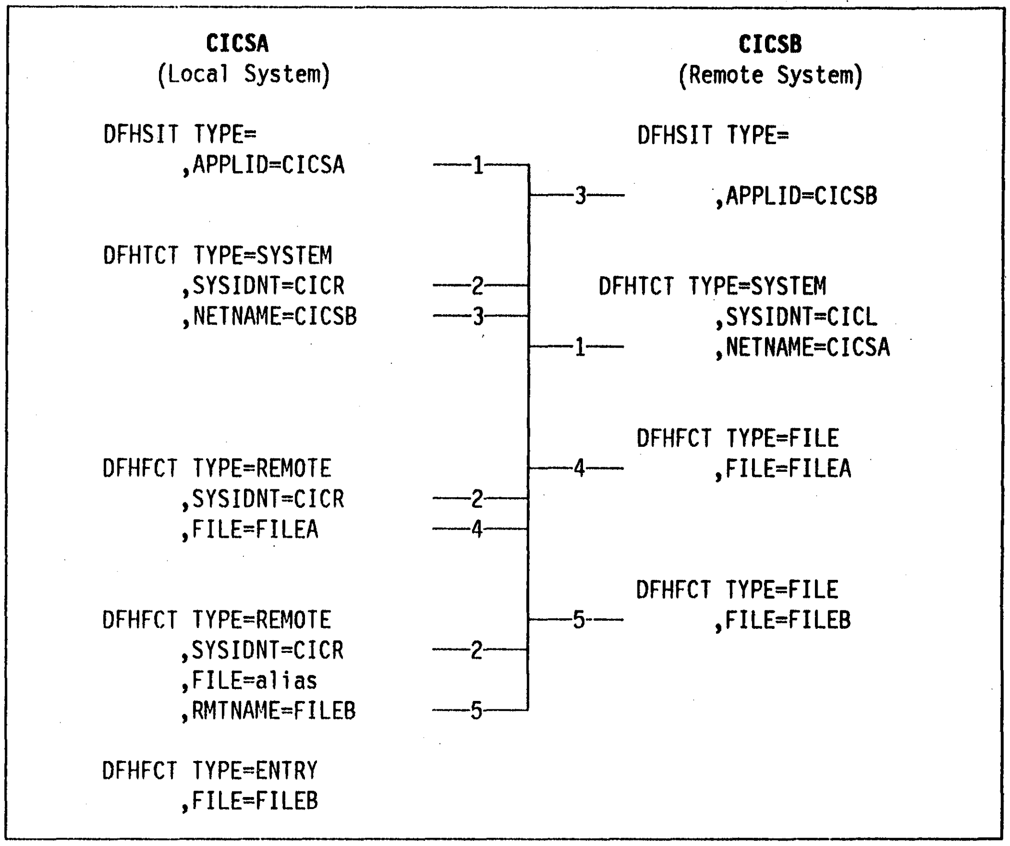
1. Remote files (function shipping)
2. Remote DL/I PSBs (function shipping)
3. Remote transient data destinations (function shipping)
4. Remote temporary storage queues (function shipping)
5. Remote terminals (transaction routing)
6. Remote transactions (transaction routing and asynchronous processing).

All remote resources must, of course, also be defined on the systems that own them.

Local and remote names for resources

CICS resources are usually referred to by name; a file name for a file, a data identifier for a temporary storage queue, and so on. When you are defining remote resources, you must consider both the name of the resource on the remote system and the name by which it is known in the local system. The CICS definition macros for remote resources all have a RMTNAME operand to enable you to specify the name of the resource on the remote system. If you omit this operand, CICS will assume that the local and remote names of the resource are identical.

Local and remote resource naming is illustrated in Figure 42 on page 134.



```
          CICSA                                          CICSB
      (Local System)                                (Remote System)

  DFHSIT TYPE=                                   DFHSIT TYPE=
        ,APPLID=CICSA        ──1──┐
                                  ├──3──             ,APPLID=CICSB
                                  │
  DFHTCT TYPE=SYSTEM              │
        ,SYSIDNT=CICR        ──2──┤             DFHTCT TYPE=SYSTEM
        ,NETNAME=CICSB       ──3──┤                     ,SYSIDNT=CICL
                                  ├──1──                ,NETNAME=CICSA
                                  │
                                  │              DFHFCT TYPE=FILE
  DFHFCT TYPE=REMOTE              ├──4──               ,FILE=FILEA
        ,SYSIDNT=CICR        ──2──┤
        ,FILE=FILEA          ──4──┤
                                  │
                                  │              DFHFCT TYPE=FILE
  DFHFCT TYPE=REMOTE              ├──5──               ,FILE=FILEB
        ,SYSIDNT=CICR        ──2──┤
        ,FILE=alias                │
        ,RMTNAME=FILEB       ──5──┘

  DFHFCT TYPE=ENTRY
        ,FILE=FILEB
```

Figure 42. Local and remote resource names

This figure illustrates the relationship between local and remote resource names. It shows two files, FILEA and FILEB, which are owned by a remote CICS system (CICSB), together with their definitions as remote resources in the local CICS system CICSA.

FILEA has the same name on both systems, so that a reference to FILEA on either system means the same file.

FILEB is provided with an alias name on the local system, so that the file is referred to by its alias in the local system and by FILEB on the remote system. The "real" name of the remote file is specified in the RMTNAME operand. This enables CICSA to own a local data set called FILEB.

CICS function shipping

The remote resources that you may have to define if you are using CICS function shipping are:

1. Remote files
2. Remote DL/I PSBs
3. Remote transient data destinations
4. Remote temporary storage queues.

Defining remote files

A remote file is a file that resides on another CICS system. CICS file control requests that are made against a remote file are shipped to the remote system by means of CICS function shipping.

CICS application programs can name a remote system explicitly on file control requests, by means of the SYSID option. If this is done, there is no need for the remote file to be defined on the local CICS system.

More generally, however, applications are designed to access files without being aware of their location, and in this case the remote file must be defined in the local file control table.

Remote file entries in the file control table

A remote file entry in the file control table provides CICS with sufficient information to enable it to ship file control requests to a specified remote system. It is defined by means of a DFHFCT TYPE=REMOTE resource definition macro. The format of this macro is given in the *CICS/MVS Resource Definition (Macro)* manual and is reproduced here for ease of reference.

```
DFHFCT   TYPE=REMOTE
         ,SYSIDNT=name
         ,FILE=name
         [,RMTNAME=name]
         [,KEYLEN=key-length]
         [,LRECL=record-length]
         [,RSL={0|number|PUBLIC}]
```

Figure 43. Defining a remote file (function shipping)

Although MRO is supported for both user-maintained and CICS-maintained remote data tables, CICS does not allow you to define a local data table based on a remote source data set. However, there are ways around this restriction (see the *CICS/MVS Data Tables Guide* for further information).

The name of the remote system

The name of the remote system to which file control requests for this file are to be shipped is specified in the SYSIDNT option. A link to this system must have been defined as described in "Chapter 3.1. Defining links to remote systems" on page 91.

The name specified in the SYSIDNT option must not be the name of the local system.

File names

The name by which the file is known on the local CICS system is specified in the FILE option. This is the name that is used in file control requests by application programs in the local system.

The name by which the file is known on the remote CICS system is specified in the RMTNAME option. This is the name that is used in file control requests that are shipped by CICS to the remote system.

If the name of the file is to be the same on both the local and the remote system, the RMTNAME operand need not be specified. You should, however, consider carefully the desirability of using the FILE option to provide a local alias for the file called RMTNAME on the remote system. This technique is, of course, essential if files of the same name reside on both systems.

Record lengths

The record length of a remote file can be specified in the LRECL option.

If your installation uses COBOL, you should specify the record length for any file that has fixed length records.

In all other cases, the record length is either a mandatory option on file control commands or can be deduced by the command-language translator.

Defining remote DL/I PSBs

To enable the local CICS system to access remote DL/I PSBs, you must define the remote PSBs in the local PSB directory (PDIR). The form of macro used for this purpose is:

```
DFHDLPSB TYPE=ENTRY
         ,PSB=psbname
         ,SYSIDNT=name
         ,MXSSASZ=value
         [,RMTNAME=name]
```

This entry refers to a PSB that is known to the IMS DB system on the system identified by the SYSIDNT operand.

A database descriptor (DBD) entry in the local CICS DMB directory (DDIR) is not required if the DBD resides on a remote system.

If there are no local DL/I databases on your CICS/MVS system, **all** the entries in the PDIR will be defined as remote by inclusion of the SYSIDNT operand. In this case, you must provide an "empty" DDIR, as follows:

```
DFHDLDBD TYPE=INITIAL
         [,SUFFIX=xx]
DFHDLDBD TYPE=FINAL
```

Defining remote transient data destinations

A remote transient data destination is one that resides on another CICS system. CICS transient data requests that are made against a remote destination are shipped to the remote system by means of CICS function shipping.

CICS application programs can name a remote system explicitly on transient data requests, by means of the SYSID option. If this is done, there is no need for the remote transient data destination to be defined on the local CICS system.

More generally, however, applications are designed to access transient data destinations without being aware of their location, and in this case the remote destination must be defined in the local destination control table.

A remote entry in the destination control table provides CICS with sufficient information to enable it to ship transient data requests to a specified remote system. It is defined by means of a DFHDCT TYPE=REMOTE resource definition macro. The format of this macro is given in the *CICS/MVS Resource Definition (Macro)* manual, and is reproduced here for ease of reference.

```
DFHDCT   TYPE=REMOTE
         ,DESTID=name
         ,SYSIDNT=name
         [,LENGTH=length]
         [,RMTNAME=name]
         [,RSL={0|number|PUBLIC}]
```

Defining remote temporary storage queues

A remote temporary storage queue is one that resides on another CICS system. CICS temporary storage requests that are made against a remote queue are shipped to the remote system by means of CICS function shipping.

CICS application programs can name a remote system explicitly on temporary storage requests, by means of the SYSID option. If this is done, there is no need for the remote temporary storage queue to be defined on the local CICS system.

More generally, however, applications are designed to access temporary storage queues without being aware of their location, and in this case the remote destination must be defined in the local temporary storage table.

A remote entry in the temporary storage table provides CICS with sufficient information to enable it to ship temporary storage requests to a specified remote system. It is defined by means of a DFHTST TYPE=REMOTE resource definition macro. The format of this macro is given in the *CICS/MVS Resource Definition (Macro)* manual, and is reproduced here for ease of reference.

```
DFHTST   TYPE=REMOTE
         ,SYSIDNT=name
         ,DATAID=character-string
         [,RMTNAME=character-string]
```

Asynchronous processing

The only remote resource definitions needed for asynchronous processing are for transactions that are named in the TRANSID option of CICS START commands.

Note, however, that an application can use the CICS RETRIEVE command to obtain the name of a remote temporary storage queue which it subsequently names in a function shipping request.

Defining remote transactions

A remote transaction for CICS asynchronous processing is a transaction that is owned by another system and which is invoked from the local CICS system only by means of START commands.

CICS application programs can name a remote system explicitly on START commands, by means of the SYSID option. If this is done, there is no need for the remote transaction to be defined on the local CICS system. However, if the transaction may be invoked by CICS transaction routing as well as by START commands, the remote transaction must be defined in the local program control table to enable transaction routing.

More generally, however, applications are designed to start transactions without being aware of their location, and in this case the remote transaction must be defined in the local program control table.

Remote transactions that are invoked only by START commands require only basic information in the local program control table. The form of resource definition required for this purpose is:

```
RDO Definition                Macro-Level Definition

DEFINE                        DFHPCT TYPE=REMOTE
  TRANSACTION(name)                  ,TRANSID=name
  GROUP(groupname)
  REMOTESYSTEM(sysidnt-name)         ,SYSIDNT=name
  REMOTENAME(name)                   [,RMTNAME=name]
  LOCALQ(NO|YES)                     [,LOCALQ={NO|YES}]
  RSL(0|number|PUBLIC)               [,RSL={0|number|PUBLIC}]
```

Figure 44. Defining a remote transaction (asynchronous processing)

Local queuing (LOCALQ) can be specified for remote transactions that are initiated by START requests. For further details, see "Chapter 1.5. Asynchronous processing" on page 33.

CICS transaction routing

CICS transaction routing enables a terminal that is owned by a particular CICS region to invoke a transaction that is owned by another CICS region. The two regions must be connected either by MRO or by an LUTYPE6.2 (APPC) link.

Both the terminal and the transaction must be defined in both CICS regions, as follows:

1. In the terminal-owning region:
   a. The terminal must be defined as a local resource.
   b. The transaction must be defined as a remote resource.
2. In the application-owning region:
   a. The terminal must be defined as a remote resource (unless a shipped terminal definition will be available; see "Shipping terminal definitions" on page 141).
   b. The transaction must be defined as a local resource.

If indirect routing is to be used, the rules that have just been stated still apply. In addition, both the terminal and the transaction must be defined as remote resources in the intermediate CICS region.

Transactions can be defined either by macro-level resource definition or by resource definition online (RDO). VTAM terminals can also be defined using either method, but for non-VTAM terminals you must use macro-level definition.

There is no requirement for the same method to be used in the two systems involved in transaction routing, but using the same method can simplify the preparation of local and remote definitions for the same resource.

Not all terminals are eligible for transaction routing. The following terminals and logical units cannot use transaction routing and therefore cannot be defined as remote:

- APPC (LUTYPE6.2) terminals
- Pooled TCAM terminals
- IBM 7770 or 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- MVS operator console.

Defining remote terminals with RDO (VTAM terminals only)

Remote terminals are terminals owned by a remote system which need to be able to run with transactions on the local system, using the CICS transaction routing facility.

The following section tells you how to define remote VTAM terminals using RDO. However, if the terminal-owning region is using RDO, you do not necessarily have to define the terminal on the application-owning region. Instead, you can arrange for a suitable definition to be **shipped** from the terminal-owning region when it is required. This method is described in "Shipping terminal definitions" on page 141.

With resource definition online (RDO), remote terminals are defined by means of a DEFINE TERMINAL command that specifies a REMOTESYSTEM name that is different from the SYSIDNT of the region on which the terminal definition is being installed. Only a few of the various terminal properties need be specified for a remote terminal definition. They are:

```
DEFINE
   TERMINAL
   GROUP

terminal identifiers
   TYPETERM
   REMOTENAME
   REMOTESYSTEM

operator defaults
   OPERRSL
   OPERSECURITY
```

Figure 45. Defining a remote VTAM terminal (transaction routing)

The TYPETERM referenced by a remote terminal definition can be a CICS-supplied version for the particular terminal type, or one defined by means of a DEFINE TYPETERM command. If you are defining a TYPETERM that will be used **only** for remote terminals, you can ignore the **session properties**, the **paging properties**, and the **operational properties**. You can also ignore BUILDCHAIN in the **application features**.

Sharing terminal definitions

If you have two or more CICS regions within the same MVS image, they can share a common CICS system definition file (CSD). In this case, for MRO transaction routing, you need define each terminal only once.

The terminal must be fully defined by means of DEFINE TERMINAL, and must have an associated TYPETERM definition, just like a local terminal definition. In addition, the REMOTESYSTEM operand must specify the SYSIDNT of the terminal-owning region. When such a terminal is installed on the terminal-owning region, a full, local, terminal definition is built. On any other system, a remote terminal definition is built.

Shipping terminal definitions

If you are using RDO on a terminal-owning region that is involved in transaction routing, you can arrange for a terminal definition to be shipped from the terminal-owning region to the application-owning region whenever it is required. If you use this method, you need not define the terminal on the application-owning region.

If you require a transaction that is started by ATI to acquire a remote terminal, you will normally have to define the terminal on the application-owning region. For example, specifying a remote terminal in the DESTFAC=(TERMINAL,trmidnt) operand for an intrapartition transient data queue (see "Intrapartition transient data queues and remote transactions" on page 157) does not cause a terminal definition to be shipped from the remote system, but once a shipped terminal definition has been received, the terminal is eligible for ATI requests.

However, CICS does allow you to cause terminal definitions to be shipped to the AOR in support of ATI requests. If you enable the user exit XALTENF in the AOR, CICS invokes this exit whenever it meets a *terminal not known* condition. The program you code has access to parameters, giving details of the origin and nature of the ATI request. You use these to decide the identity of the region that owns the terminal definition you want CICS to ship for you.

A similar user exit, XICTENF, is available for start requests that result from EXEC CICS START. See "Shipping terminals for automatic transaction initiation" on page 48 for more information.

To make a terminal definition eligible for shipping, you must associate it with a TYPETERM that specifies SHIPPABLE:

```
DEFINE
   TERMINAL(trmidnt)
   GROUP(groupname)
   AUTINSTMODEL(YES|NO|ONLY)
   AUTINSTNAME(name)
   TYPETERM(TRTERM1)
   .
   .

DEFINE
   TYPETERM(TRTERM1)
   .
   .
   SHIPPABLE(YES)
```

Figure 46. Defining a shippable terminal (transaction routing)

This method can be used for any VTAM terminal. For AUTOINSTALL terminals, this method **must** be used. In effect, it gives automatic installation of remote terminals. (For details of AUTOINSTALL, see the *CICS/MVS Resource Definition (Online)* manual and the *CICS/MVS Customization Guide*.)

When a remote transaction is invoked from a shippable terminal, the request that is transmitted to the application-owning region is flagged to show that a shippable terminal definition is available. If the application-owning region already has a definition of the terminal (which may have been shipped previously), it ignores the flag. Otherwise, it asks for the definition to be shipped. A shipped terminal definition is retained until one of the following events occurs:

1. A CEDA INSTALL command causes the terminal definition on the running CICS system that shipped the definition to be replaced or deleted.
2. The terminal is logged off on the system that shipped the definition (autoinstalled terminals only).
3. The system that shipped the terminal definition is cold-started.

If either 1 or 2 above occurs while communication between the two systems has been lost, the action is deferred until communication has been restored.

Defining remote terminals (macro-level definition)

Remote terminals are terminals owned by a remote system which need to be able to run with transactions on the local system, using the CICS transaction routing facility.

A remote terminal requires a full terminal control table entry in the remote system, and a local terminal control table entry that contains sufficient information about the terminal to enable CICS to perform the transaction routing. Data set control information and line information is not required for the local definition of a remote terminal. With resource definition macros, you can define remote terminals in either of two ways:

1. By means of DFHTCT TYPE=REMOTE macro instructions
2. By means of normal DFHTCT TYPE=TERMINAL macro instructions preceded by a DFHTCT TYPE=REGION macro instruction.

The choice of a method is largely a matter of convenience in the particular circumstances. Both methods allow the same terminal definitions to be used to generate the required entries in both the local and the remote system.

Definition using DFHTCT TYPE=REMOTE

The format of the DFHTCT TYPE=REMOTE macro instruction is given in the *CICS/MVS Resource Definition (Macro)* manual, and is reproduced here for ease of reference.

```
DFHTCT   TYPE=REMOTE
     ,ACCMETH=access-method
     ,SYSIDNT=name
     ,TRMIDNT=name
     [,RMTNAME=name]
     ,TRMTYPE=terminal-type

     [,ALTPGE=(lines,columns)]
     [,ALTSCRN=(lines,columns)]
     [,ALTSFX=number]
     [,DEFSCRN=(lines,columns)]
     [,ERRATT={NO
       |([LASTLINE][,INTENSIFY][,color][,highlight])}]
     [,FEATURE=(feature[,feature],...)]
     [,OPERRSL={0|(number[,...]}]
     [,OPERSEC={1|(number[,number],...)}]
     [,PGESIZE=(lines,columns)]
     [,TCTUAL=number]
     [,TIOAL={value|(value1,value2)}]
     [,TRMMODL=numbercharacter]

     Non-VTAM

     [,DISMSG=name]
     [,LPLEN={132|value}]
     [,STN2980=number]
     [,TAB2980={1|value}]

     VTAM and TCAM SNA Only

     [,BMSFEAT=(FMHPARM,NOROUTE,NOROUTEALL,OBFMT,OBOPID)]
     [,HF={NO|YES}]
     [,LDC={listname|(aa[=nnn],bb[=nnn],cc[=nnn],...)}]
     [,SESTYPE=session-type]
     [,VF={NO|YES}]

     VTAM Only

     [,FF={NO|YES}]
```

Figure 47. Defining a remote terminal

With the exception of SYSIDNT, the operands of this macro instruction form a subset of those that can specified with DFHTCT TYPE=TERMINAL. Any of the remaining operands can be specified. They are ignored unless the SYSIDNT operand names the local system, in which case the macro instruction becomes equivalent to the DFHTCT TYPE=TERMINAL form.

A single DFHTCT TYPE=REMOTE macro instruction can therefore be used to define the same terminal in both the local and the remote systems. A typical use of this method of definition is shown in Figure 48 on page 144.

Local System CICSL	Remote System CICSR (terminal-owning system)
DFHSIT TYPE=	DFHSIT TYPE=
APPLID=CICSL,	APPLID=CICSR,
SYSIDNT=CICL,	SYSIDNT=CICR,
.	.
DFHTCT TYPE=INITIAL,	DFHTCT TYPE=INITIAL,
ACCMETH=VTAM	ACCMETH=VTAM
.	.
DFHTCT TYPE=SYSTEM,	DFHTCT TYPE=SYSTEM,
SYSIDNT=CICR,	SYSIDNT=CICL,
NETNAME=CICSR,	NETNAME=CICSL,
.	.
.	.
DFHTCT TYPE=REMOTE,	DFHTCT TYPE=REMOTE,
SYSIDNT=CICR,	SYSIDNT=CICR,
TRMIDNT=aaaa,	TRMIDNT=aaaa,
TRMTYPE=LUTYPE2,	TRMTYPE=LUTYPE2,
TRMMODL=2,	TRMMODL=2,
ALTSCRN=(43,80)	ALTSCRN=(43,80)
.	.
.	.
DFHTCT TYPE=FINAL	DFHTCT TYPE=FINAL

Figure 48. Typical use of DFHTCT TYPE=REMOTE

In this example, the same terminal definition is used in both the local and the remote systems.

In the local system, because the terminal SYSIDNT differs from that specified on the DFHTCT TYPE=INITIAL macro, a remote terminal entry is built. In the remote system, because the terminal SYSIDNT is that of the remote system itself, the TYPE=REMOTE macro is treated exactly as if it were a TYPE=TERMINAL macro.

The terminal identification is "aaaa" in both systems.

Definition using DFHTCT TYPE=REGION

If this method is used, terminals can be defined in the same way as local terminals, using DFHTCT TYPE=SDSCI, TYPE=LINE, and TYPE=TERMINAL macro instructions. The definitions must, however, be preceded by a DFHTCT TYPE=REGION macro instruction, which has the following form:

```
DFHTCT   TYPE=REGION
         ,SYSIDNT={name|LOCAL}
```

Here, SYSIDNT=name specifies the terminal-owning region. If this operand does not name the local system, only the information required to build a remote terminal entry is extracted from the succeeding definitions. DFHTCT TYPE=SDSCI and TYPE=LINE definitions are ignored. Operands of

TYPE=TERMINAL definitions that are not part of the TYPE=REMOTE subset are also ignored.

A return to local system definitions is made by use of DFHTCT TYPE=REGION,SYSIDNT=LOCAL.

A typical use of this method of definition is shown in Figure 49.

```
  Local System CICSL                 Remote System CICSR
                                    (terminal-owning system)

  DFHSIT TYPE=                       DFHSIT TYPE=
         APPLID=CICSL,                      APPLID=CICSR,
         SYSIDNT=CICL,                      SYSIDNT=CICR,
         .                                  .

  DFHTCT TYPE=INITIAL,               DFHTCT TYPE=INITIAL,
         ACCMETH=VTAM                       ACCMETH=VTAM

  DFHTCT TYPE=SYSTEM,                DFHTCT TYPE=SYSTEM,
         SYSIDNT=CICR,                      SYSIDNT=CICL,
         NETNAME=CICSR,                     NETNAME=CICSL,
         .                                  .
  DFHTCT TYPE=REGION,
         SYSIDNT=CICR
  COPY TERMDEFS                      COPY TERMDEFS
  DFHTCT TYPE=REGION,
         SYSIDNT=LOCAL

  DFHTCT TYPE=FINAL                  DFHTCT TYPE=FINAL
```

```
  * TERMDEFS COPY BOOK

 L77A  DFHTCT  TYPE=TERMINAL,TRMIDNT=L77A,ACCMETH=VTAM,
               TRMTYPE=L3277,TRMMODL=2,CLASS=(CONV,VIDEO),
               TIOAL=1500,RELREQ=(YES,YES),
               FEATURE=(SELCTPEN,AUDALARM,UCTRAN),
               TCTUAL=8,CONNECT=AUTO,TRMSTAT=(TRANSCEIVE)
       DFHTCT  TYPE=SDSCI,CU=3272,DEVICE=L3277,LINELST=(035),
               DSCNAME=DDA11,BSCODE=EDCDIC
       DFHTCT  TYPE=LINE,ACCMETH=BTAM,TRMTYPE=L3277,
               DSCNAME=DDA11,INAREAL=512,TRMMODL=2,BTAMRLN=1,
               POOLADDR=L77B,BSCODE=EBCDIC,DUMMY=DUMMY
 L77B  DFHTCT  TYPE=TERMINAL,TRMIDNT=L77B,LVUNIT=1,
               FEATURE=(SELCTPEN,UCTRAN,AUDALARM),
               TRMSTAT=TRANSCEIVE,LASTTRM=POOL,
               TCTUAL=8,TIOAL=80
 and so on
```

Figure 49. Typical use of DFHTCT TYPE=REGION

In this example, the same copy book of terminal definitions is used in both the local and the remote system.

In the local system, the fact that the terminal SYSIDNT differs from that of the local system (specified on the DFHSIT macro or the DFHTCT TYPE=INITIAL macro) causes remote terminal entries to be built. Note that although the TYPE=SDSCI and TYPE=LINE macros are not expanded in the local system, any defaults that they imply (for example, ACCMETH=BTAM) are taken for the TYPE=TERMINAL expansions.

Local and remote names for terminals

CICS uses a unique identifier for every terminal that is involved in transaction routing. The identifier is formed from the APPLID of the CICS system that owns the terminal and the terminal identifier specified in the terminal definition on the terminal-owning region.

If, for example, the APPLID of the CICS system is PRODSYS and the terminal identifier is L77A, the fully-qualified terminal identifier is PRODSYS.L77A.

The following rules apply to all forms of remote terminal definition:

1. The definition must be associated with a system whose NETNAME is the APPLID of the system that owns the terminal.
2. The "real" terminal identifier must always be specified, either directly or by means of aliasing.

Referring to the correct netname

You must always ensure that the system identifier named in a remote terminal definition refers to a link whose NETNAME is the APPLID of the terminal-owning region. In the following examples, it is assumed that the APPLID of the terminal-owning region is PRODSYS.

```
Resource Definition Online

  DEFINE TERMINAL                DEFINE CONNECTION(PD1)
    REMOTESYSTEM(PD1)              NETNAME(PRODSYS)
    .                              .


Macro-Level Definition
(Method 1)

  DFHTCT TYPE=REMOTE,            DFHTCT TYPE=SYSTEM,
    SYSIDNT=PD1,                   SYSIDNT=PD1,
    .                              NETNAME=PRODSYS,
    .                              .


Macro-Level Definition
(Method 2)

  DFHTCT TYPE=REGION,            DFHTCT TYPE=SYSTEM,
    SYSIDNT=PD1                    SYSIDNT=PD1,
    .                              NETNAME=PRODSYS,
                                   .
  DFHTCT TYPE=TERMINAL,
    .
```

Figure 50. Identifying a terminal-owning region

These rules apply even if an indirect link is being used (see "Indirect links for transaction routing" on page 126). The NETNAME used must never be that of an indirect system.

Terminal aliases

The name by which a terminal is known in the application-owning region is usually the same as its name in the terminal-owning region. You can, however, choose to call the remote terminal by a different name (an alias) in the application-owning region.

You will have to provide an alias if the terminal-owning region and the application-owning region own a terminal with the same name; you cannot have a local terminal definition and a remote terminal definition with the same name.

If you use an alias, you must also specify the "real" name of the terminal as its remote name, as follows:

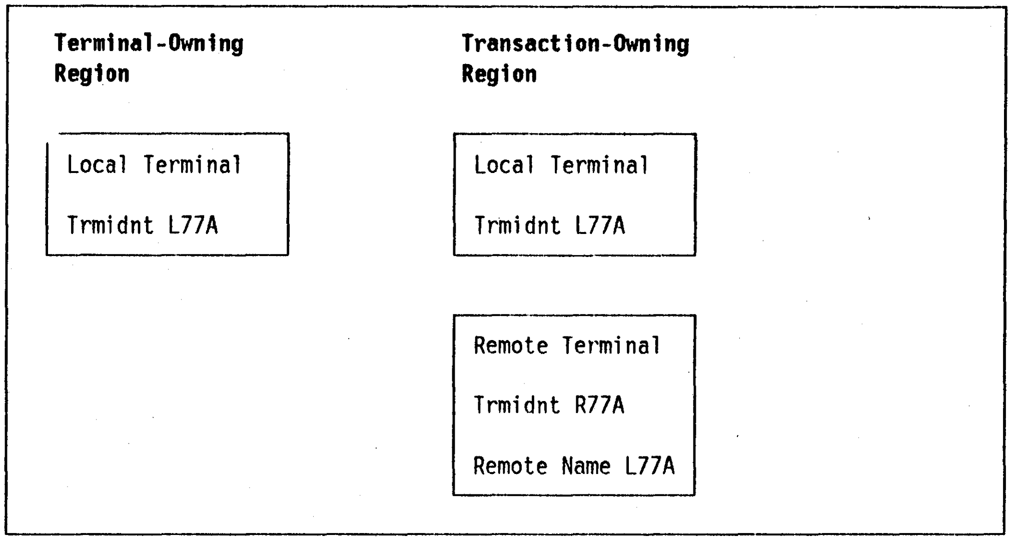


Figure 51. Local and remote names for remote terminals

You specify the remote name in the REMOTENAME operand of DEFINE TERMINAL or the RMTNAME operand of DFHTCT TYPE=REMOTE.

Defining remote transactions

A remote transaction for CICS transaction routing is a transaction that is owned by another CICS system and which can be invoked from the local CICS system by a terminal owned by the local system.

You define a remote transaction in exactly the same way as you define a local transaction, except that some of the operands are not required.

In what follows, it is assumed that the online resource definition transaction CEDA is being used. The form of the CEDA DEFINE command for remote transactions is:

```
    DEFINE
    TRANSACTION(name)
    GROUP(groupname)
    [PROGRAM(name)]
    [TWASIZE({0|value})]
    [PROFILE({DFHCICST|name})]
    [PARTITIONSET(name)]
    [STATUS({ENABLED|DISABLED})]
    [PRIMEDSIZE({0|value})]

Remote attributes
    REMOTESYSTEM(name)
    [REMOTENAME({local-name|remote-name})]
    [TRPROF({DFHCICSS|name})]
    [LOCALQ({NO|YES})]

Scheduling
    [PRIORITY({1|value})]
    [TCLASS({NO|value})]

Aliases
    [TASKREQ(value)]
    [XTRANID(value)]

Recovery
    [DTIMOUT({NO|value})]
    [INDOUBT({BACKOUT|COMMIT|WAIT})]
    [RESTART({NO|YES})]
    [SPURGE({NO|YES})]
    [TPURGE({NO|YES})]
    [DUMP({YES|NO})]
    [TRACE({YES|NO})]

Security
    [EXTSEC({NO|YES})]
    [TRANSEC({1|value})]
    [RSL({0|value|PUBLIC})]
    [RSLC({NO|YES|EXTERNAL})
```

Figure 52. Defining a remote transaction (transaction routing)

Note: For information on the use of the INDOUBT operand, see "Chapter 5.1. Recovery and restart in interconnected systems" on page 289.

The name in the TRANSACTION operand is the name by which the transaction is invoked in the terminal-owning region. TASKREQ can be specified if special inputs, such as a program attention (PA) key, program function (PF) key, light pen, magnetic stripe reader, or operator ID card reader, are used.

The REMOTESYSTEM operand names the system to which the transaction will be routed, and an MRO or LUTYPE6.2 link to this system must have been defined.

The PROFILE operand names the profile that is to be used for communication between the terminal and the relay transaction. The TRPROF operand names the profile that is to be used for communication on the session between the relay transaction and the remote, application-owning, region. Information about profiles is given under "Defining communication profiles" on page 151.

The program associated with a remote transaction is always the relay program DFHCRP, and the transaction is often referred to as the relay transaction (see "Chapter 1.6. CICS transaction routing" on page 45).

The attributes that you define apply to the execution of the relay transaction in the terminal-owning region, and not to the execution of the routed transaction in the application-owning region.

It may also be desirable to specify some operands for control of the relay transaction.

You can set TWASIZE to zero since the relay transaction does not require a TWA.

You should specify transaction security for routed transactions which are operator initiated. You do not need to specify resource security checking, since the relay transaction does not access resources. Security is discussed in "Chapter 6.1. Security in the intercommunication environment" on page 311.

You should code the DTIMOUT operand to cause the relay transaction to be timed out if the system to which a transaction is routed does not respond or if a session does not become available after a reasonable period of time.

Distributed transaction processing

There are no remote resource definition requirements for distributed transaction processing.

Chapter 3.3. Defining local resources

Defining communication profiles

When a transaction acquires an LUTYPE6.1 or LUTYPE6.2 session to another system, either explicitly by means of an ALLOCATE command or implicitly because it uses, for example, function shipping, a communication profile is associated with the communication between the transaction and the session. The communication profile specifies the following information:

1. Whether FMHs received from the session are to be passed on to the transaction.
2. Whether input and output messages are to be journaled, and if so the location of the journal.
3. The node error program (NEP) class for errors on the session.
4. For LUTYPE6.2 sessions, the modename of the group of sessions from which the session is to be allocated. (If the profile does not contain a modename, CICS selects a session from any available group.)

CICS provides a set of default profiles, described later in this chapter, which it uses for various forms of communication. Also, you can define your own profiles, and name a profile explicitly on an ALLOCATE command.

The general form of the CEDA command used to define a profile is shown in Figure 53. (For full details, see *CICS/MVS Resource Definition (Online)* manual).

```
DEFINE
     PROFILE(name)
     GROUP(groupname)
     [MODENAME(name)]

Protocols
     [INBFMH(NO|ALL)]

Journaling
     [JOURNAL(NO|value)]
     [MSGJRNL(NO|INPUT|OUTPUT|INOUT)]

Recovery
     [NEPCLASS(0|value)]
     [RTIMOUT(NO|value)]


Note:  The equivalent operands are also available on the
  DFHPCT TYPE=PROFILE macro.
```

Figure 53. Defining a communication profile

A profile is always required for a session acquired by an ALLOCATE command; either a profile that you have defined and which is named explicitly on the command, or the default profile DFHCICSA. If CICS cannot find the profile, the CBIDERR condition is raised in the application program.

For MRO sessions that are acquired by an ALLOCATE command, CICS always uses INBFMH(ALL), no matter what is specified in the profile.

For LUTYPE6.2 conversations, INBFMH specifications are ignored; LUTYPE6.2 FMHs are never passed to CICS application programs.

Communication profiles for principal facilities

A profile is also associated with the communication between a transaction and its principal facility. You can name the profile in the CEDA DEFINE TRANSACTION command, or you can allow the default to be taken. The CEDA DEFINE PROFILE command for a principal facility profile has more operands than the form required for alternate facilities. Details are given in the *CICS/MVS Resource Definition (Online)* manual.

Default profiles

CICS provides a set of communication profiles that it employs in cases where the user does not or cannot specify a profile explicitly. The default profiles are:

DFHCICST
is the default profile for principal facilities. You can specify a different profile for a particular transaction by means of the PROFILE option of the CEDA DEFINE TRANSACTION command.

DFHCICSV
is the profile for principal facilities of the CICS-supplied transactions CSNE, CSLG, and CSRS. It is the same as DFHCICST except that DVSUPRT(VTAM) is specified in place of DVSUPRT(ALL). You should not modify this profile.

DFHCICSE
is the error profile for principal facilities. CICS uses this profile to pass an error message to the principal facility when the required profile cannot be found.

DFHCICSA INBFMH(ALL)
is the default profile for alternate facilities that are acquired by means of an application program ALLOCATE command. A different profile can be named explicitly on the ALLOCATE command.

This profile is also used as a principal facility profile for some CICS-supplied transactions.

DFHCICSF INBFMH(ALL)
is the profile that CICS uses for the session to the remote system or region when a CICS application program issues a function shipping request.

DFHCICSS INBFMH(ALL)
is the profile that CICS uses in transaction routing for communication between the relay transaction (running in the terminal-owning region) and the MRO or LUTYPE6.2 link.

DFHCICSR INBFMH(ALL)
is the profile that CICS uses in transaction routing for communication between the user transaction (running in the application-owning region) and the interregion link or LUTYPE6.2 link.

Note that the user-transaction's principal facility is the surrogate TCTTE in the application-owning region, for which the default profile is DFHCICST.

Refer to the *CICS/MVS Resource Definition (Online)* manual or the *CICS/MVS Resource Definition (Macro)* manual for information on how to include the default profiles in your program control table.

Modifying the default profiles

You can modify a default profile by means of the CEDA transaction or, if you are using macro-level definition, by coding a new DFHPCT TYPE=PROFILE macro.

A typical reason for modification is to include a modename to provide class of service selection for, say, function shipping requests on LUTYPE6.2 links. If you do this, you must ensure that every LUTYPE6.2 link in your installation has a group of sessions with the specified modename.

If you modify DFHCICSA, you must retain INBFMH(ALL), because it is required by certain CICS-supplied transactions. Modifying this profile does not affect the profile options assumed for MRO sessions.

You must not modify DFHCICSV, which is used only by certain CICS-supplied transactions.

Architected processes

An architected process is an IBM-defined method of allowing dissimilar products to exchange intercommunication requests in a way that is understood by both products. For example, a typical requirement of intersystem communication is that one system should be able to schedule a transaction for execution on another system. Both CICS and IMS have transaction schedulers, but their implementation differs considerably. The intercommunication architecture overcomes this problem by defining a model of a "universal" transaction scheduling process. Both products implement this architected process, by mapping it to their own internal process, and are therefore able to exchange scheduling requests.

The architected processes implemented by CICS are:

- System message model — for handling messages containing various types of information that needs to passed between systems (typically, DFS messages from IMS)
- Scheduler model — for handling scheduling requests
- Queue model — for handling queuing requests (in CICS terms, temporary storage or transient data requests)

- DL/I model — for handling DL/I requests
- LU services model — for handling requests between LUTYPE6.2 service managers.

Note: With the exception of the LUTYPE6.2 LU services model, the architected processes are defined in the LUTYPE6.1 architecture. CICS, however, also uses them for function shipping on LUTYPE6.2 links by using LUTYPE6.2 migration mode.

The appropriate models are also used for CICS-CICS communication. The exception is CICS file control requests, which are handled by a CICS-defined file control model.

During resource definition, your only involvement with architected processes will be to ensure that the relevant transaction and programs are included in the CICS tables, and possibly to change their priorities.

Process names

Architected process names are one to four bytes long, and have a first byte value that is less than X'40'.

In CICS, the names are specified as four-byte hexadecimal transaction identifiers. If CICS receives an architected process name that is less than four bytes long, it pads the name with null characters (X'00') before searching for the transaction identifier.

CICS supplies the processes shown in Figure 54.

XTRANID	TRANSID	PROGRAM	DESCRIPTION
For CICS file control			
-	CSMI	DFHMIR	file control model
For LUTYPE6.1 architected processes			
01000000	CSM1	DFHMIR	system message model
02000000	CSM2	DFHMIR	scheduler model
03000000	CSM3	DFHMIR	queue model
05000000	CSM5	DFHMIR	DL/I model
For LUTYPE6.2 architected processes			
06F10000	CLS1	DFHLUP	LU services model
06F20000	CLS2	DFHLUP	LU services model

Figure 54. CICS architected process names

Modifying the architected process definitions

You will observe from the previous list that the CICS file control model and the architected processes for function shipping all map to program DFHMIR[2], the CICS mirror program. The inclusion of different transaction names for the various models enables you to modify some of the transaction attributes. You must not, however, change the XTRANID, the TRANSID, or the PROGRAM operand.

You can modify any of the definitions by means of the CEDA transaction or, if you are using macro-level definition, by coding a new DFHPCT macro.

Interregion function shipping

Function shipping over MRO links can employ reusable mirror tasks and the short-path transformer program (see "MRO function shipping" on page 26).

If you modify one or more of the mirror transaction definitions, you must evaluate the effect that this may have on interregion function shipping. **Any** suspended mirror task can be used to service an interregion request, and that mirror can have been attached originally by any process name, and hence transaction identifier.

The short-path transformer always specifies transaction CSMI. It is not, however, used for DL/I requests; they arrive as requests for process X'05000000', corresponding to transaction CSM5.

Selecting the required PCT and PPT entries

The profiles and architected processes described in this chapter, and other transactions and programs that are required for ISC and MRO, are contained in the IBM protected groups DFHISC and DFHSTAND. Details of how to include these pregenerated CEDA groups in your CICS system are given in the *CICS/MVS Resource Definition (Online)* manual. The contents of group DFHISC are summarized in the following figure.

[2] Transaction CVMI and program DFHMIRVM are also used for LU6.2 parallel connections operating at synclevel(1) and LU6.2 single-session connections.

TRANSACTIONS

XTRANID	TRANSID	PROGRAM	GROUP	
-	CSMI	DFHMIR	DFHISC	CICS file control model
-	CVMI	DFHMIRVM	DFHISC	Mirror program
01000000	CSM1	DFHMIR	DFHISC	system message model
02000000	CSM2	DFHMIR	DFHISC	scheduler model
03000000	CSM3	DFHMIR	DFHISC	queue model
05000000	CSM5	DFHMIR	DFHISC	DL/I model
06F10000	CLS1	DFHLUP	DFHISC	LU services model
06F20000	CLS2	DFHLUP	DFHISC	LU services model
-	CMPX	DFHMXP	DFHISC	
-	CRSQ	DFHCRQ	DFHISC	
-	CRSR	DFHCRS	DFHISC	
-	CRTE	DFHRTE	DFHISC	routing transaction
-	CSIR	DFHCRR	DFHISC	
-	CSNC	DFHCRNP	DFHISC	

PROGRAMS

NAME	GROUP	
DFHCRNP	DFHISC	Interregion new connection manager
DFHCRSP	DFHISC	Interregion control initialization program
DFHCRR	DFHISC	IRC session recovery program.
DFHCRS	DFHISC	Remote scheduler program.
DFHCRP	DFHISC	Transaction routing relay program.
DFHRTE	DFHISC	Transaction routing program.
DFHCRQ	DFHISC	ATI purge program.
DFHMXP	DFHISC	Local queuing shipper program
DFHMIR	DFHISC	Mirror program
DFHMIRVM	DFHISC	Mirror program
DFHLUP	DFHISC	LU services program
DFHSTLK	DFHSTAND	ISC Link and IRC statistics program

PROFILES

NAME	GROUP	
DFHCICSF	DFHISC	Function shipping profile
DFHCICSR	DFHISC	Transaction routing receive profile
DFHCICSS	DFHISC	Transaction routing send profile
DFHCICSA	DFHSTAND	Distributed transaction processing profile
DFHCICSE	DFHSTAND	Principal facility error profile
DFHCICST	DFHSTAND	Principal facility default profile
DFHCICSV	DFHSTAND	Principal facility special profile

Figure 55. The contents of group DFHISC

Intrapartition transient data queues and remote transactions

The general form of the resource definition macro for an intrapartition transient data queue is:

```
DFHDCT TYPE=INTRA
       ,DESTID=name
       [,DESTFAC={(TERMINAL[,termid])|FILE|(SYSTEM,sysid)}
       [,DESTRCV={NO|PH|LG}]
       [,REUSE={YES|NO}]
       [,RSL={0|number|PUBLIC}]
       [,TRANSID=name]
       [,TRIGLEV={1|number}]
```

Figure 56. Defining an intrapartition transient data queue

Full details of this macro are given in the *CICS/MVS Resource Definition (Macro)* manual. In this section we are concerned with the CICS intercommunication aspects of queues that:

1. Cause automatic transaction initiation (TRANSID specified)
2. Specify an associated principal facility (DESTFAC=TERMINAL or DESTFAC=SYSTEM).

Transactions

A transaction that is initiated by an intrapartition transient data queue must reside on the same system as the queue. That is, the transaction that you specify in the TRANSID operand must not be defined as a remote transaction.

Principal facilities

The principal facility that is associated with a transaction started by ATI can be:

- A local terminal
- A terminal owned by a remote system
- An intersystem or interregion session to a remote system.

Local terminals

A local terminal is a terminal that is owned by the same system that owns the transient data queue and the transaction.

For any local terminal other than an LUTYPE6.2 (APPC) terminal, you need to specify DESTFAC=(TERMINAL[,termid]). If you omit **termid**, the name of the queue (specified in DESTID) must be the same as the terminal name.

For a local LUTYPE6.2 (APPC) terminal, you need to specify DESTFAC=(SYSTEM,sysid), where **sysid** is the system name of the terminal. Note that the name of the **conversation** with the terminal is assigned when the session is allocated. You do not know it beforehand, and therefore cannot specify it. Once the transaction has started, the name of the conversation is available in EIBTRMID.

Remote terminals

A remote terminal is a terminal that is defined as remote on the system that owns the transient data queue and the associated transaction. Automatic transaction initiation with a remote terminal is a form of CICS transaction routing (see "Chapter 1.6. CICS transaction routing" on page 45), and the normal transaction routing rules apply.

Use DESTFAC=(TERMINAL,termid) to specify the name of the remote terminal. The terminal itself must be defined as a remote terminal, and the terminal-owning region must be connected to the local system either by an MRO link or an LUTYPE6.2 link.

Remote systems

You can name a remote system in the DESTFAC=(SYSTEM,sysid) operand. The remote system can be connected by any type of link; MRO, LUTYPE6.1, or LUTYPE6.2.

Part 4. Application programming

This part of the manual describes the application programming aspects of CICS intercommunication. It contains the following chapters:

Chapter 4.1. Application programming overview

Application programs that are designed to run in the CICS intercommunication environment can use one or more of the following facilities:

- Function shipping
- Asynchronous processing
- Transaction routing
- Distributed transaction processing.

The application programming requirements for each of these facilities are described separately in the remaining chapters of this part. If your application program uses more than one facility, the relevant chapter should be used as an aid to designing the corresponding part of the program. Similarly, if your program uses more than one intersystem session for distributed transaction processing, it must control each individual session according to the rules given for the appropriate session type.

Programming languages

In general, CICS application programs that use CICS intercommunication facilities can be written in COBOL, PL/I, or Assembler language, and must use the CICS command-level application programming interface. There are two exceptions to these rules:

1. Macro-level as well as command-level application programs can be invoked by transaction routing.
2. Application programs that hold LUTYPE6.2 **unmapped** conversations must be written in Assembler language.

Terminology

The following terms are sometimes used without further explanation in the remaining chapters of this part:

principal facility
the "terminal" that is associated with your transaction when the transaction is initiated. The more general term is used because the facility may be not a "real" terminal but an intersystem session. CICS commands, such as SEND or RECEIVE, that do not explicitly name a facility are taken to refer to the principal facility. Only one principal facility can be owned by a transaction.

alternate facility
in distributed transaction processing, a transaction can acquire the use of a session to a remote system. This session is called an alternate facility. It must be named explicitly on CICS commands that refer to it. A transaction can own more than one alternate facility.

Other intersystem sessions, such as those used for function shipping, are not owned by the transaction, and are not regarded as alternate facilities of the transaction.

front-end and back-end transactions
in distributed transaction processing, one of the pair of conversing transactions must be initiated first, acquire a session to the remote system, and cause the other transaction to be initiated. This is the front-end transaction. The transaction that the front-end transaction causes to be initiated is the back-end transaction.

Note that a transaction can at the same time be the back-end transaction on one conversation and the front-end transaction on one or more other conversations.

syncpoint initiator
the transaction that initially issues a SYNCPOINT or SYNCPOINT ROLLBACK request. The term is commonly abbreviated to initiator.

syncpoint agent
the transaction that receives a SYNCPOINT or SYNCPOINT ROLLBACK request from a partner. The term is commonly abbreviated to agent.

Chapter 4.2. Application programming for CICS function shipping

If you are writing a program to access resources in a remote system, you code it in much the same way as if the resources were on the local system. Your program can be written in PL/I, COBOL, or assembler language. Function shipping is available only through the CICS command-level interface or through DL/I calls or EXEC DLI commands.

The commands that you can use to access remote resources are:

1. File control commands
2. DL/I calls or EXEC DLI commands
3. Temporary storage commands
4. Transient data commands.

Interval control commands are deliberately left out of this list. For information on this subject, see "Chapter 4.3. Application programming for asynchronous processing" on page 167.

Your application can run in the CICS intercommunication environment and make use of the intercommunication facilities without being aware of the location of the resource being accessed. The location of the resource is defined by the system programmer in the appropriate CICS table. Optionally, you can use the SYSID option on EXEC commands to select the system on which the commana is to be executed. In this case, the resource definition tables on the local system are not referenced, unless the SYSID option names the local system.

When your application issues a command against a remote resource, CICS ships the request to the remote system, where a mirror transaction is initiated. The mirror transaction executes the request on your behalf, and returns any output to your application program. The mirror transaction is thus effectively a remote extension of your application program. If you would like more information on this mechanism, read "Chapter 1.4. CICS function shipping" on page 21.

Although the same commands are used to access both local and remote resources, there are a number of restrictions that apply when the resource is remote. Also, some errors that do not occur in single systems can arise when function shipping is being used. For these reasons, you should always know whether resources that your program accesses can possibly be remote.

File control

Function shipping allows you to access VSAM or DAM files located on a remote system.

If you use the SYSID option to access a remote system directly, you must observe the following rules:

1. For a file referencing a keyed data set, KEYLENGTH must be specified if RIDFLD is specified, unless you are using relative byte addresses (RBA) or relative record numbers (RRN).

For a remote DAM file, where the DEBKEY or DEBREC options have been specified, KEYLENGTH must be the total length of the key.

2. If the file has fixed length records, you must specify the record length (LENGTH).

These rules also apply if the file control table entry for the file does not define the appropriate values.

DL/I

Function shipping allows you to access DL/I DOS/VS or IMS DB data associated with a remote CICS system through the DL/I CALL interface or by using EXEC DLI commands.

Definitions of remote DL/I databases are provided by the system programmer. There is no facility for selecting specific systems in CICS application programs.

Temporary storage

Function shipping allows you to send data to or receive data from temporary storage queues located on remote systems. Definitions of remote temporary storage queues can be made by the system programmer. You can, however, use the SYSID option on the WRITEQ TS, READQ TS, and DELETEQ TS commands to specify the system on which the request is to be executed.

For MRO sessions, the MAIN and AUXILIARY options of the WRITEQ TS command can be used to select the required type of storage. If the queue is to be recoverable, AUXILIARY should be selected.

For LUTYPE6.1 or LUTYPE6.2 sessions, the MAIN and AUXILIARY options are ignored; auxiliary storage is always used in the remote system.

Transient data

Function shipping allows you to access intrapartition or extrapartition transient data queues located on remote systems. Definitions of remote transient data queues can be made by the system programmer. You can, however, use the SYSID option on the WRITEQ TD, READQ TD, and DELETEQ TD commands to specify the system on which the request is to be executed.

If the remote transient data queue has fixed length record, you must supply the record length in the LENGTH option if it is not specified in the DFHDCT TYPE=REMOTE macro, or if you use the SYSID option.

Function shipping exceptional conditions

Requests that are shipped to a remote system can raise any of the exceptional conditions for the command that can occur if the resource is local. In addition, there are some conditions that apply only when the resource is remote.

Remote system not available

At the time that a function shipping request is issued, a link to the remote system may not be available. If this is the case, the SYSIDERR condition is raised in the application program.

This condition is also raised if the named system is not defined, but this error should not occur in a production system unless the application is designed to obtain the name of the remote system from a terminal operator.

The default action for the SYSIDERR condition is to terminate the task abnormally.

Invalid request

The ISCINVREQ condition occurs when the remote system indicates a failure that does not correspond to a known condition. The default action is to terminate the task abnormally.

Mirror transaction abend

An application request against a remote resource may cause an abend in the mirror transaction (for example, the requested Transient Data destination may have been disabled by the remote CICS master terminal operator).

In these situations, the application program will also be abended, but with an abend code of ATNI (for ISC connections) or AZI6 (for MRO connections). The actual error condition will be logged by CICS in an error message sent to the CSMT destination. Any HANDLE ABEND command issued by the application will not be able to identify the original cause of the condition and take explicit corrective action (which may have been possible if the resource was local). An exception occurs in MRO function shipping if the mirror transaction abends with a DL/I program isolation deadlock; in this case, the application will abend with the normal deadlock abend code.

Note that the ATNI abend caused by a mirror transaction abend is not related to a terminal control command, and the TERMERR condition is therefore not raised.

Chapter 4.3. Application programming for asynchronous processing

This chapter discusses the application programming requirements for CICS-to-CICS asynchronous processing. The general information given for CICS transactions that use the START or RETRIEVE commands is also applicable to CICS-IMS communication. For details of the requirements of CICS-IMS communication, however, you should refer to "Chapter 4.8. CICS-to-IMS applications" on page 263.

A description of the concepts of asynchronous processing is given in "Chapter 1.5. Asynchronous processing" on page 33. It is assumed that you are familiar with concepts of CICS interval control, as described in the *CICS/MVS Application Programmer's Reference* manual.

Starting a transaction on a remote system

You can start a transaction on a remote system by issuing an EXEC CICS START command just as though the transaction were a local one. Remote transactions cannot be initiated by means of macro-level interval control requests.

Generally, the transaction will have been defined as being remote by the system programmer. You can, however, name a remote system explicitly in the SYSID option. This use of the START command is thus essentially a special case of CICS function shipping.

If your application requires you to specify the time at which the remote transaction is to be initiated, remember that the remote system may be in a different time zone. The use of the INTERVAL form of control is preferable under these circumstances.

Exceptional conditions for the START command

The exceptional conditions that can occur as a result of issuing a START request for a remote transaction depend on whether or not the NOCHECK performance option is specified on the START command.

If NOCHECK is not specified, the raising of conditions follows the normal rules for function shipping (see "Function shipping exceptional conditions" on page 165).

If NOCHECK is specified, no conditions will be raised as a result of the remote execution of the START command. SYSIDERR, however, will still occur if no link to the remote system is available, unless the system programmer has arranged for local queuing of start requests. Also, the local transaction will be abended if the remote mirror transaction associated with the START command abends.

Retrieving data associated with a remotely-issued start request

The RETRIEVE command is used to retrieve data that has been stored for a task as a result of a remotely-issued start request. This is the only available method for accessing such data.

As far as your transaction is concerned, there is no distinction between data stored by a remote start request and data stored by a local start request, and the normal considerations for use of the RETRIEVE command apply.

Chapter 4.4. Application programming for CICS transaction routing

In general, if you are writing a transaction that may be used in a transaction routing environment, you can design and code it just as you would for a single CICS system. There are, however, a number of restrictions that you must be aware of, and these are described in this chapter. The same considerations apply if you are migrating an existing transaction to the transaction routing environment.

The program can use either command-level or macro-level, and can be written in PL/I, COBOL, or assembler language.

Note: Information on macro-level programs is intended primarily for the migration of existing programs to a transaction routing environment. It is strongly recommended that command level be used for new applications.

Basic mapping support

Any BMS maps or partition sets that your program uses must reside in the same CICS system.

In a BMS routing application, a route request that specifies an operator or an operator class will direct output only to the operators signed on at terminals that are owned by the system in which the transaction is executing.

Pseudoconversational transactions

A routed transaction requires the use of an interregion or intersystem (LUTYPE6.2) session for as long as it is running. For this reason, long-running conversational transactions are best duplicated in the two systems, or alternatively designed as pseudoconversational transactions.

Care is needed in the naming and definition of the individual transactions that make up a pseudoconversational transaction, because a TRANSID specified in a CICS RETURN command is returned to the terminal-owning region, where it may be a local transaction.

There is, however, no reason why a pseudoconversational transaction cannot be made up of both local and remote transactions.

The terminal

The "terminal" with which your transaction will run is represented by a terminal control table entry (TCTTE) which is in many respects a copy of the "real" terminal TCTTE in the terminal-owning region. This copy is known as the **surrogate** TCTTE.

The surrogate TCTTE is released when the transaction terminates. Subsequent tasks will run using a new copy of the real terminal TCTTE. You should note that, if the new task is started via ATI, certain fields (for example, TCTTEAID (EIBAID) and TCTTECAD (EIBSCON)) will be meaningless and may contain either zeros or residual data.

Using the EXEC CICS ASSIGN command in the AOR

Three of the options to the EXEC CICS ASSIGN command may cause an unexpected reaction or return unexpected values. A closer look at these will help you to understand why:

PRINSYSID
This option returns the *sysid* of the principal facility to the transaction. It requires that this facility be an MRO, or an LUTYPE6.1 or LUTYPE6.2 session. The principal facility for a routed transaction is represented by the surrogate TCTTE, which does not meet the requirement. Therefore the INVREQ condition is raised (as stated in the *CICS/MVS Application Programmer's Reference* manual).

Note: An EXEC CICS ASSIGN PRINSYSID command cannot be used to find the name of the terminal-owning region.

USERID
For a routed transaction, CICS takes the *userid* from one of two sources, depending on how you specified your security requirements (see "Transaction routing security" on page 326).

If you specified preset security, by including OPERSECURITY, OPERRSL, or both, on the definition for the remote terminal, whatever was specified for the USERID option on the same DEFINE TERMINAL command is returned under this option. This appears as *blanks* if you let the USERID option default.

CICS goes to the same source for the *userid* if you did not specify preset security for the remote terminal definition but you did specify ATTACHSEC(Local) on the DEFINE CONNECTION command for the link.

If, instead of ATTACHSEC(Local), you requested automatic signon for remote users by specifying ATTACHSEC(Identify) or ATTACHSEC(Verify) on the DEFINE CONNECTION command for the link, the *userid* returned is the one that was sent over the link with the attach request for the transaction.

OPERKEYS
This option returns a 64-bit mask that represents the CICS transaction security profile of the remote user **in the local system**.

If preset security was defined in the remote terminal definition, the preset value is returned. This selection of this possibility precludes all others.

If the remote user is signed on locally as described in the explanation to USERID above, the returned mask is the value that was defined for the user in the signon table.

In all other cases, the user transaction security profile takes the default value of 1.

This option cannot give any information about the user's security status in a remote system.

Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations

"Chapter 1.7. Distributed transaction processing." introduces the concepts of distributed transaction processing (DTP). This chapter tells you how to code CICS application programs that hold high-level (mapped) conversations on LUTYPE6.2 sessions. The session partner can be another CICS system, or any other LUTYPE6.2 system that supports mapped conversations.

There is another type of LU6.2 conversation called a ***basic*** or ***unmapped*** conversation, and this is documented in "Chapter 4.6. CICS applications for logical unit type 6.2 unmapped conversations."

Application design

The starting point for the design of an LUTYPE6.2 application is provided by the application requirements developed during the planning activity.

In general, a program that is required to hold LUTYPE6.2 conversations must be designed as one of a pair of conversing applications, not as an isolated entity.

At the same time, you should ensure that your CICS application program follows a well-defined set of protocols, both to achieve correct operation and to deal with unexpected situations. This means that you must at all times be aware of the state of the session, as indicated by the settings of fields in the EXEC interface block (EIB), and use that state information to determine what operations are currently valid, or even mandatory.

A guide to the correct use of EIB fields and command sequences is given in "Command sequences on LUTYPE6.2 mapped conversations" on page 203.

Warning

Assumptions should not be made about the state of a conversation. The application must test the EIB flags after each LUTYPE6.2 command to determine the current state. If you follow this rule, your distributed application can converse with, or be migrated to, other releases of CICS without encountering significant problems.

Sessions and conversations

In the LUTYPE6.2 architecture, a distinction is made between a session, which represents a path between two logical units, and a conversation, which refers to exchanges between the end users of the session.

Your transaction can acquire the use of a session on which to conduct a conversation by using the ALLOCATE command. CICS may have to acquire a new session in order to satisfy your request, but this is incidental. Sessions are usually long-lived, and are used in turn by many different transactions.

When you have acquired a session, you can begin a conversation by using the CONNECT PROCESS command to initiate a back-end transaction. The conversation lasts from the sending of begin bracket with the first message to the receipt of end-bracket with the final message. In CICS transactions, you must give up your use of the session, by issuing a FREE command, at the end of the conversation. It is not possible to start another conversation on the same session.

Conversation state

LUTYPE6.2 conversations embody the concept of a **state**. Each end of the conversation is in a particular state, and what you can do on the conversation is governed by its state.

For example, only one end of the conversation is permitted to have the ability to send data at any time. This side is in SEND state; this forces its partner to be in RECEIVE state. The partner in SEND state can give up this ability (for example, by issuing a SEND INVITE command), thereby placing itself in RECEIVE state and its partner in SEND state.

Effective use of LUTYPE6.2 DTP depends on the good management of these states. See "Command sequences on LUTYPE6.2 mapped conversations" on page 203 for full details.

Synchronization levels

Three levels of synchronization are available for LUTYPE6.2 conversations:

- Level 0 – none

  No synchronization can take place between the conversation partners.

- Level 1 – confirm

  The conversation partners handle synchronization exchanges themselves. The CICS syncpointing mechanisms are not involved.

- Level 2 – syncpoint

  The equivalent of normal CICS syncpointing.

Synchronization must be at a level that both the conversation partners can accept. The maximum level that can be used is negotiated when the session that is to carry the conversation is bound. The level that is to be used for a particular conversation is requested by the front-end transaction. It will be accepted provided that:

- It is not greater than the maximum allowed for the session
- It is acceptable to the back-end transaction.

Abbreviations and assumptions

The rest of this chapter refers to synchronization levels 0, 1, and 2 as SL(0), SL(1), and SL(2) respectively. Unless otherwise stated, operation at SL(2) should be assumed.

CICS commands for LUTYPE6.2 conversations

The commands that can be used to establish and hold LUTYPE6.2 conversations are:

- ALLOCATE — used by the front-end transaction to acquire a session to the remote system.
- CONNECT PROCESS — used by the front-end transaction to initiate a conversation with a named process (or, in CICS terms, a transaction) on the remote system.
- EXTRACT PROCESS — used by the back-end transaction to access session-related information (for example, the requested synchronization level) in the LUTYPE6.2 attach header that caused it to be initiated.
- SEND, RECEIVE, and CONVERSE — used by the conversing transactions to send or receive data on the conversation.
- SEND INVITE — used to change the state of the issuing side from SEND to RECEIVE, and that of the partner from RECEIVE to SEND.
- WAIT or SEND WAIT — used to ensure that all existing data and control indicators have been sent to a partner before further processing is performed.
- SEND CONFIRM and ISSUE CONFIRMATION — used primarily at SL(1), but also available at SL(2), for the exchange of private requests to show that data has arrived and been processed correctly.
- SYNCPOINT and SYNCPOINT ROLLBACK — operates only at SL(2), and governs synchronization of all active conversations.
- ISSUE PREPARE — operates only at SL(2), and is used by a syncpoint initiator to ensure that its agents are ready to take a SYNCPOINT.
- ISSUE ERROR — used by either transaction to inform its conversation partner that a program-detected error has occurred.
- ISSUE ABEND — used by either transaction to inform its conversation partner that it is necessary to abend the conversation.
- ISSUE SIGNAL — used by the receiving transaction to request a change-direction from the sending transaction.
- FREE — used by either transaction to free the session.

Considerations for the front-end transaction

The front-end transaction is responsible for requesting a session to the remote system and initiating the remote process with which it is to converse. Thereafter, the conversation partners become equals.

Session allocation

An Application can get an LUTYPE6.2 session to a remote system by means of the ALLOCATE command, which has the following format:

```
ALLOCATE SYSID(name)
  [PROFILE(name)]
  [NOQUEUE]

  SYSIDERR,SYSBUSY,CBIDERR
```

The name specified in the SYSID option must be the name of an LUTYPE6.2 intersystem link. CICS will raise the SYSIDERR condition if:

- it cannot find the named system,
- the connection is in the INSERVICE RELEASED status (that is, the SNASVCMG sessions are not bound), or
- the connection is OUTSERVICE.

You can use the PROFILE option to select a specified communication profile for the session and the ensuing conversation. The profile, which is set up during resource definition, can contain the name of the group of LUTYPE6.2 sessions from which the session is to be acquired, thereby enabling a particular class of service to be selected. It also contains a set of terminal control processing options that are to be used for the conversation.

If you omit the PROFILE option, CICS will use the default profile DFHCICSA. Note that although DFHCICSA specifies INBFMH=ALL, this operand is ignored for LUTYPE6.2 conversations. LUTYPE6.2 function management headers will never be passed to your application program.

You can use the NOQUEUE (or NOSUSPEND) option to tell CICS to return control to you immediately if it finds it has no session available for allocation. For the purposes of this option, a session is available for allocation only if it meets **all** these conditions:

- It is a contention-winner
- It is already bound
- It is not already allocated.

The SYSBUSY code (X'D3') is set in the EIBRCODE field of the EXEC interface block if no session is available for allocation.

If your application has executed a HANDLE command for the SYSBUSY condition and no session can fulfil the above conditions, control is returned to the label specified in the HANDLE command. For this, you do not need to specify NOQUEUE and, if you do, it is overridden.

If you neither select NOQUEUE nor handle the SYSBUSY option, CICS tries to allocate a contention-winning session that is also available and bound. If it fails, it suspends your transaction and attempts to obtain an alternative session, working through the following categories in order and taking the action shown, where appropriate:

- Unbound contention winners. If one is *available*, it is bound.

- Unbound indeterminate sessions. These may be present if the combined number of contention winners defined for both sides is less than the total number of *available* sessions. If one is available, it is declared to be a contention winner and is bound.
- Bound contention losers. If one is available, a bid is issued to the partner system for the session.

Control is passed back to your transaction as soon as a session can be allocated. If you specify DTIMOUT on the DEFINE command for the transaction, CICS causes an ABEND if the time-out expires before a session is allocated. The HANDLE ABEND command lets you regain control in this event.

Both your resource configuration and your transaction design should be aimed at avoiding long or, in the worst case, irresolvable ALLOCATE suspensions. Single-session connections are particularly vulnerable. For more guidance, refer to "The AUTOCONNECT operand" on page 125 and "Chapter 6.2. Master terminal operations for LUTYPE6.2 connections" on page 329.

The conversation identifier

When a session has been allocated, the name by which the conversation will be known is available in the EIBRSRCE in the EIB. You should obtain this name immediately. It is the name that you must use in the CONVID option of all subsequent commands that relate to this conversation.

You should not make any assumptions about the name of the conversation. It will be different for different invocations of your transaction, and bears no defined relationship to the name that CICS uses to refer to the session that carries the conversation.

Attaching the remote process

When a session has been acquired, the next step is to initiate the conversation with the remote process (in CICS terms, a transaction) by means of a CONNECT PROCESS command. This command has the following format:

```
CONNECT PROCESS
   PROCNAME(data-area)
   PROCLENGTH(data-value)
   CONVID(name)
   SYNCLEVEL(data-value)
  [PIPLIST(data-area)
   PIPLENGTH(data-value)]

   INVREQ, NOTALLOC, LENGERR
```

The process, or transaction, that is to be connected to the other end of the conversation is named in the PROCNAME option, and the length of the name is specified as a halfword binary value in the PROCLENGTH option.

Four bytes are sufficient to identify a CICS transaction. However, the LUTYPE6.2 architecture allows a range of 1 to 64 bytes, leaving each product free to set its own maximum. CICS complies by allowing 32 bytes, but this need only concerr you if you are linked to a non-CICS system which demands longer transaction identifiers. To attach a transaction in the remote system, you need only supply the operands as set out above. If you expect a remote system to send attach requests with names longer than four bytes, you have a choice. Because CICS always interprets the first four bytes, you can make sure that these always represent a transaction identifier within your system. Alternatively, you can examine the full identifier by coding the user exit XZCATT as described in the *CICS/MVS Customization Guide*.

Note: Transaction names that begin with X'00' through X'3F' are reserved for use as SNA service transaction names, and cannot be used for user transactions. For details of LUTYPE6.2 symbol string conventions, see the SNA publication *Transaction Programmer's Reference Manual for LU Type 6.2*.

The conversation on which the connect process command is to be issued must be named in the CONVID option. The name that you use must be the name acquired from EIBRSRCE after session allocation.

The SYNCLEVEL option enables you to specify the synchronization level that is required for the conversation. The synchronization level required is usually determined when the overall application is designed.

You should remember these points when you specify the SYNCLEVEL option:

- Every remote set of sessions has a maximum synchronization level associated with it. A value of 2 is implied for a CICS system. If you want to converse with a non-CICS system, you cannot specify a level that is greater than the remote system can support; the CONNECT PROCESS will fail if you do.
- If you include commands in your transaction that are inconsistent with the synchronization level you have specified for a session, they will not involve the remote transaction connected by that session in any way.
- You should specify SYNCLEVEL(2) only if you want both conversation partners to participate in a syncpoint initiated by either partner. Remember that transaction termination initiates a syncpoint without express command.
- Distributed syncpointing requires complex coding, and adds flow to the application, increasing the network load. Do not use SL(2) unless it is required to protect data integrity. A discussion of SL(2) operation follows at "Syncpoint exchanges" on page 186.

The PIPLIST option specifies the process initialization parameter (PIP) data that is to be sent to the remote system. For a description of the format and use of PIP data, see "Process initialization parameter data" on page 17. The PIPLENGTH option must specify the total length of the PIP list.

Automatic transaction initiation

If a transaction is to be started by automatic transaction initiation (ATI) from a transient data trigger level with an LUTYPE6.2 session as its principal facility, the definition of the transient data queue must specify DESTFAC=(SYSTEM,sysid), where sysid is the sysid of the remote LUTYPE6.2 system (see "Intrapartition transient data queues and remote transactions" on page 157). The transaction can acquire the name of the queue that caused its initiation by means of the ASSIGN QNAME command.

Execution of an EXEC CICS START command that names a remote LUTYPE6.2 system in the TERMID option causes the transaction to be initiated with a session to the named system as its principal facility. The session is selected from the modeset specified for the transaction, if there is one. Otherwise, the modeset is selected by CICS.

The transaction starts in state 2 (Session Allocated), and continues by issuing a CONNECT PROCESS command.

Considerations for the back-end transaction

The back-end transaction in a conversation is initiated by the LUTYPE6.2 attach FMH received from the remote system and is started with the conversation as its principal facility.

As the back-end transaction is conversing with its principal facility (that is, the partner that did the initial ALLOCATE), the CONVID can be omitted from all LUTYPE6.2 commands. If the back-end transaction initiates its own conversation, however, CONVID must be supplied when it is conversing with this new partner (but CONVID can still be omitted when the transaction is conversing with its principal facility).

Acquiring conversation-related information

You can use the EXTRACT PROCESS command to recover conversation-related information from the attach FMH if required, but the use of this command is not mandatory. The command has the following format:

```
EXTRACT PROCESS
  [PROCNAME(data-area)
   PROCLENGTH(data-area)]
  [CONVID(name)]
  [SYNCLEVEL(data-area)]
  [PIPLIST(pointer-ref)
   PIPLENGTH(data-area)]

  INVREQ, NOTALLOC
```

The process name (PROCNAME) from the LUTYPE6.2 attach header, the length of the process name (PROCLENGTH), and the requested synchronization level (SYNCLEVEL) are returned in the specified data areas.

If you are issuing the EXTRACT PROCESS command on the principal facility, CONVID need not be specified.

The PIPLIST option specifies a pointer reference that is set to the address of a CICS-provided data area containing a PIPLIST (see "Process initialization parameter data" on page 17). The pointer value is null if no PIPLIST has been received.

The PIPLENGTH option returns the total length of the PIPLIST as a halfword binary value.

Initial state of back-end transaction

The back-end transaction is always initiated in RECEIVE state. However, to properly initiate the conversation with the front-end transaction, you ***must*** issue an EXEC CICS RECEIVE command before you do anything else that could effect the link (for example, a SYNCPOINT or a SYNCPOINT ROLLBACK command).

The only exception to this rule is that you can issue an EXTRACT PROCESS command, before EXEC CICS RECEIVE, to obtain the synchronization level and other conversation-related information.

Using the EXEC CICS ASSIGN command

You may find that two of the options to the EXEC CICS ASSIGN command return unexpected values. A closer look at these will help you to understand why:

USERID

CICS takes the *userid* from one of two sources, depending on how you specified your security requirements.

If you specified ATTACHSEC(Local) on the DEFINE CONNECTION command for the link, whatever was specified for the USERID option on the same DEFINE SESSIONS command is returned under this option. This appears as *blanks* if you let the USERID option default.

If, instead of ATTACHSEC(Local), you requested automatic signon for remote users by specifying ATTACHSEC(Identify) or ATTACHSEC(Verify) on the DEFINE CONNECTION command for the link, the *userid* returned is the one that was sent over the link with the attach request for the transaction.

OPERKEYS

This option returns a 64-bit mask that represents the CICS transaction security profile of the remote user **in the local system**. If the remote user is signed on locally as described in the explanation to USERID above, the returned mask is the value that was defined for the user in the signon table.

If no signon takes place, the user's security profile defaults to that of the link. The link itself may be signed on, in which case the mask will be taken from the signon table entry for the link. The other possibility is that OPERSECURITY, OPERRSL, or both were specified on the DEFINE SESSIONS command for the link. This *preset* security then determines the value returned under this option. In all cases of default, a value of 1 is returned.

This option cannot give any information about the user's security status in a remote system.

The conversation

The conversation between the front-end and the back-end transactions is held using the SEND, RECEIVE, and CONVERSE commands. Details of these commands for LUTYPE6.2 programs are given in the *CICS/MVS Application Programmer's Reference* manual.

Note that, for LUTYPE6.2 conversations, the use of the FROM option on the SEND command is optional. This means that you can use the SEND command to send the change-direction indicator (INVITE option) or the end-bracket indicator (LAST option) without naming any data for transmission.

When LUTYPE6.2 commands are being used, the CONVID parameter tells CICS which partner to talk to. If you are conversing with the principal facility, you can ***omit*** the CONVID parameter. In all other cases, include it. To show the options, consider this example:

Use of CONVID parameter

$$\begin{tikzcd}\boxed{\text{TASK A}} \arrow[r] & \boxed{\text{TASK B}} \arrow[r] & \boxed{\text{TASK C}}\end{tikzcd}$$

A→B uses SEND CONVID(b)

B→A uses SEND
B→C uses SEND CONVID(c)

C→B uses SEND

The principal facility of task A is a terminal, so it needs a CONVID when conversing with task B.

Task B has been initiated by an ALLOCATE in task A, so task A is its principal facility, and no CONVID is needed by task B when conversing with task A.

Task C has been initiated by an ALLOCATE in task B, so task B is its principal facility, and no CONVID is needed by task C when conversing with task B. However, one is required by task B when conversing with task C.

Deferred transmission

When you issue a SEND command, CICS normally defers sending the data until it becomes clear what your further intentions are. This mechanism enables CICS to avoid unnecessary flows by adding control indicators, such as end-bracket and syncpoint-request, to the data that is awaiting transmission.

For LUTYPE6.2, CICS reduces the number of flows still further by accumulating the data from successive SEND commands in an internal buffer. The contents of the buffer are not transmitted until the buffer becomes full or until it is flushed by the CONFIRM option on a SEND command or by an explicit or implicit WAIT. However, although the data is "bunched" for transmission, each RECEIVE command issued by the receiving transaction recovers only the data associated with a single SEND command.

The way in which LUTYPE6.2 deferred transmission operates is shown by the following command sequence:

```
SEND CONVID(CONV1)  data1 is placed in the send buffer.
     FROM(data1)    Transmission is deferred.
     LENGTH(251)

SEND CONVID(CONV1)  data2 is added to the send buffer.
     FROM(data2)    Transmission is still deferred.
     LENGTH(251)

SEND CONVID(CONV1)  the change-direction indicator is
     INVITE         added.  Transmission is still deferred.

WAIT CONVID(CONV1)  data1 and data2 are transmitted.
                    data2 carries the change-direction
                    indicator.

RECEIVE CONVID(CONV1)
        .
        .
```

The WAIT option can, of course, be added to a SEND command to cause immediate transmission:

```
SEND CONVID(CONV1)
     FROM(data2)
     LENGTH(251)
     INVITE
     WAIT

RECEIVE CONVID(CONV1)
        .
        .
```

Please remember that SEND WAIT alone is an incomplete command and is not the equivalent of WAIT.

A further short-cut is possible by omitting the INVITE and the WAIT options. If your transaction is in send state, and you issue a RECEIVE command, CICS will supply any implied options and send the deferred data with the change-direction indicator set. However, better program documentation will be achieved if you specify the options explicitly.

The CONVERSE command always implies the sequence:

```
SEND INVITE
WAIT
RECEIVE
```

Using the LAST option

The LAST option on the SEND command indicates the end of the conversation. No further data flows can occur on the conversation, and the next action must be to free the session. However, the session can still carry CICS syncpointing flows before it is freed.

The LAST option and syncpoint flows

If the conversation is using normal CICS syncpointing (level 2), it is eligible to take part in CICS syncpointing activity. A syncpoint is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command.

If your conversation has been terminated by a SEND LAST command, without the WAIT option, transmission will have been deferred, and the syncpointing activity will cause the final transmission to occur with a piggy-backed syncpoint request. The conversation will thus be automatically involved in the syncpoint.

If you do not want the conversation to be involved in the syncpoint (for example, because you know that the remote transaction does not access any recoverable resources) you must issue a SEND LAST WAIT command, or use the WAIT CONVID or FREE command, to force the transmission before using a command that causes a syncpoint.

Note: It is recommended that you initiate the last syncpoint for a conversation explicitly (by an EXEC CICS SYNCPOINT command) rather than implicitly (by an EXEC CICS RETURN command).

If an explicit syncpoint request is issued and rejected by one partner, the application is able to take the appropriate actions and issue notification of the rejection.

However, if an implicit syncpoint request is issued and rejected by the partner, the application has already returned control to CICS and no further action can be taken.

Sending and receiving error indications

Two commands are provided to enable either transaction to inform the other that an error has occurred:

- ISSUE ERROR
- ISSUE ABEND.

In general, you can use these commands at any point in your program, irrespective of whether the conversation is in send or receive state. To understand exactly when they can be used, you should use the state diagrams given under "Command sequences on LUTYPE6.2 mapped conversations" on page 203.

The use of these commands on a particular conversation always involves both of the conversing transactions. If one of the transactions is designed to use ISSUE ERROR or ISSUE ABEND, the other transaction must be designed to recognize them and take the appropriate action.

The ISSUE ERROR command

The ISSUE ERROR command has the following format:

```
ISSUE ERROR
  [CONVID(name)]

  INVREQ,NOTALLOC
```

You can use the ISSUE ERROR command to inform the conversation partner that something is wrong. Typically, you would use it as a negative response to a PREPARE, SYNCPOINT, or CONFIRM request (see "Synchronization points" on page 183). However, you can use ISSUE ERROR for any purpose required by your application design.

After issuing an ISSUE ERROR command, you should test EIBRECV to find the state of the conversation. If your transaction has not **received** ISSUE ERROR (see "ISSUE ERROR races"), it will be in **send state** on the conversation. You can use this opportunity to send information about the error to the other transaction. The usual continuation is to issue SEND INVITE followed by RECEIVE, and then check the EIB in the normal way.

From Figure 66 on page 214 you will see that, after issuing ISSUE ERROR in **receive** state, you should also test EIBFREE. This is necessary because the other transaction may already have ended the conversation by a SEND LAST WAIT command. Normally, of course, you would not design a pair of transactions in which this sequence is possible.

ISSUE ERROR races: If your application design allows either transaction to use ISSUE ERROR whenever they detect an error, there is a possibility that two ISSUE ERROR commands can cross one another. If this happens, the transaction that wins the race goes into send state. The transaction that loses the race goes into receive state, and receives an **ISSUE ERROR received** indication.

The **receipt** of an ISSUE ERROR command is indicated by EIBERR, with an error code of X'0889' in the first two bytes of EIBERRCD. These fields are set on return from the first command you issue on the conversation after the incoming error indication has been received by CICS.

| EIBERR and EIBERRCD are visible only once. If you do not test the EIB flags after each LUTYPE6.2 command, you will not be aware of these settings. This will almost certainly result in the application abending.

More information on the use of ISSUE ERROR is given under "Synchronization points" on page 183.

The ISSUE ABEND command

The ISSUE ABEND command has the following format:

```
ISSUE ABEND [CONVID(name)]

  INVREQ,NOTALLOC
```

The ISSUE ABEND command provides a means for you to abend the **conversation**. (If the remote process is a CICS transaction, the conversation abend will cause the TERMERR condition to be raised.) ISSUE ABEND can be issued by either transaction, irrespective of its send/receive state, at any time after the conversation has started. For a send-state transaction, any deferred data that is waiting for transmission is transmitted before the abend command is transmitted.

The transaction that issues the ISSUE ABEND command is not itself abended. It must, however, issue a FREE command for the conversation unless it is designed to terminate immediately.

If you issue an ISSUE ABEND command while your transaction is in receive state, CICS will purge all incoming data until a change-direction, syncpoint-request, or end-bracket indicator is received. If end-bracket is received, no error indication is sent to the remote process, and EIBFREE is set in your transaction, indicating that you must free the conversation.

The receipt of an ISSUE ABEND is indicated in EIBERR, with an error code of X'0864' in the first 2 bytes of EIBERRCD. EIBERR and EIBERRCD are visible only once. If you do not test the EIB flags after each LUTYPE6.2 command, you will not be aware of these settings. This will almost certainly result in the application abending.

Synchronization points

As described in "Synchronization levels" on page 172, there are three possible synchronization levels for LUTYPE6.2 sessions. You must distinguish between the maximum synchronization level possible between the communicating systems, which is determined at the time that the session is established, and the level at which your application is designed to operate. For a front-end transaction, you can specify the synchronization level on the CONNECT PROCESS command:

```
CONNECT PROCESS
  .
  SYNCLEVEL(data-value)
```

Do not specify a level that is greater than the remote system can support; if you do, the CONNECT PROCESS fails.

For a back-end transaction, you can find what synchronization level has been requested by **extracting** it:

```
EXTRACT PROCESS
  .
  SYNCLEVEL(data-area)
```

The possible synchronization levels for a conversation are:

Level 0
This means that no synchronization exchanges at all are possible on the conversation. If your transaction issues a CICS SYNCPOINT command, your SL(0) conversations are not involved in any way.

You can still use ISSUE ERROR commands on SL(0) conversations provided that the other transaction is designed to receive them.

Level 1
This means that the conversation supports the LUTYPE6.2 **confirmation** protocols, which enable a pair of conversing transactions to mutually confirm that they are in synchronism by means of the SEND CONFIRM and ISSUE CONFIRMATION commands. CICS syncpoint requests cannot be transmitted over SL(1) conversations; if your transaction issues a CICS SYNCPOINT command, your SL(1) conversations are not involved in any way.

Level 2
This means that the conversation supports both confirmation protocols and CICS syncpoint requests. If your transaction issues a CICS SYNCPOINT command, **all** SL(2) conversations are involved in the subsequent syncpointing activity.

Confirmation exchanges

| Confirmation exchanges affect only a single conversation, and are available at SL(1) and SL(2). A confirmation exchange involves only two commands:

1. The transaction that is in **send** state issues a SEND CONFIRM command.
2. The transaction that is in **receive** state responds, if all is well, by issuing an ISSUE CONFIRMATION command.

Negative responses to a confirmation request can be made by means of the ISSUE ERROR or ISSUE ABEND commands.

The following sections describe these commands in more detail. The descriptions refer to the **state diagrams** given under "Command sequences on LUTYPE6.2 mapped conversations" on page 203.

Sending SEND CONFIRM

The syntax of the SEND COMMAND for LUTYPE6.2 conversations is:

```
EXEC CICS SEND
[FROM(data-area) {LENGTH(data-value)|FLENGTH(data-value)}]
CONVID(name)
[INVITE|LAST]
[CONFIRM|WAIT]
```

You will see that the CONFIRM and WAIT options are mutually exclusive. CONFIRM, like WAIT, flushes the conversation send buffer; that is, it causes a real transmission to occur.

You can send data with the SEND CONFIRM command, and you can also specify either the INVITE or the LAST option.

The **state 3 — send state** diagram for LUTYPE6.2 mapped conversations (see Figure 64 on page 212) shows what happens for the possible combinations of the CONFIRM, INVITE, and LAST options. After a SEND CONFIRM command, without the INVITE or LAST options, the conversation is still in **state 3 — send state**. If the INVITE option is used, the conversation will switch to **state 5 — receive state**. If the LAST option is used, the conversation will switch to **state 10 — free session**.

These state changes assume that the other transaction has responded **positively** (ISSUE CONFIRMATION) to the CONFIRM request. You must always test EIBERR after a SEND CONFIRM command (see "Checking the response to SEND CONFIRM" on page 186).

You can achieve an effect similar to SEND LAST CONFIRM by using the command sequence:

```
SEND LAST
SEND CONFIRM
```

You can check this sequence in the state diagrams. Note that you cannot send data with a SEND CONFIRM command used in this way.

The form of command that you use depends on how you expect the conversation to continue if you receive the required confirmation. You must, however, always test for a negative response to your SEND CONFIRM command (see "Checking the response to SEND CONFIRM" on page 186).

Receiving and replying to a confirmation request

If your partner transaction issues a SEND CONFIRM, EIBCONF is set in your EIB. This is visible only once. If you do not test the EIB flags after each LUTYPE6.2 command, you will not be aware of these settings. This will almost certainly result in the application abending. Save the EIB before replying so that you can determine the conversation state following your reply.

If EIBCONF is set, you have a number of ways of replying (see Figure 67 on page 215):

1. You can reply positively by means of ISSUE CONFIRMATION, meaning that all is well. After sending ISSUE CONFIRMATION, you must check the saved values of EIBFREE and EIBRECV. This enables you to find out whether the sender sent SEND CONFIRM, SEND INVITE CONFIRM, or SEND LAST CONFIRM, and determines your next action.
2. You can reply negatively by means of ISSUE ERROR.

   This reply puts the conversation into **state 3 — send state**. What happens afterward is determined by your application design. A typical continuation is to use SEND INVITE to pass information about the error to the other transaction, followed by a RECEIVE and an EIB test to determine the next action.
3. You can abend the conversation by means of ISSUE ABEND.

   This makes the conversation unusable, and you must immediately issue a FREE for the session, or, alternatively, issue EXEC CICS RETURN.

Checking the response to SEND CONFIRM

After issuing SEND CONFIRM (or SEND CONFIRM LAST), you must check the response by testing EIBERR.

If EIBERR is not set, the other transaction has replied ISSUE CONFIRMATION, and you can continue normally.

If EIBERR is set, and the first two bytes of EIBERRCD contain X'0889', the other transaction has replied ISSUE ERROR. In this case, your conversation is in RECEIVE state, so you should issue a RECEIVE command.

Note: If you receive an error response (EIBERRCD=X'0889') in response to SEND LAST CONFIRM, the LAST option has been ignored and the conversation has **not** been terminated.

If the other transaction has replied ISSUE ABEND, the TERMERR condition will be raised in your transaction. You can choose to handle this condition (or the default condition ERROR), or you can use IGNORE or NOHANDLE to disable the condition, and check what has happened by testing EIBERRCD (see Figure 61 on page 209).

Syncpoint exchanges

This section describes the use of the SYNCPOINT, SYNCPOINT ROLLBACK, and ISSUE PREPARE commands. These commands do not affect LUTYPE6.2 conversations operating at SL(0) or SL(1).

The SYNCPOINT and SYNCPOINT ROLLBACK commands do not have the CONVID option; that is, they cannot be issued to a single conversation. They always involve all the SL(2) conversations that a transaction is holding. The ISSUE PREPARE command enables individual conversations to be prepared for syncpointing.

The SYNCPOINT command

No matter how many transactions are mutually connected by SL(2) conversations, your application design should arrange for just one of them to initiate syncpointing activity for the distributed unit of work. This **syncpoint initiator** must be in send state on all its conversations when it issues the SYNCPOINT command. Any transaction that receives the syncpoint request becomes a **syncpoint agent**.

A syncpoint agent is in receive state on its conversation with the syncpoint initiator, and becomes aware of the syncpoint request when it tests EIBSYNC after issuing a RECEIVE command (see Figure 66 on page 214). If it decides to respond positively by issuing SYNCPOINT, it must be in send state on all the conversations with its own partners, for whom it has become a syncpoint initiator. If a transaction acting as a syncpoint agent responds negatively to a syncpoint request by issuing SYNCPOINT ROLLBACK the initiator will see this in the EIB, which must be tested on return from the SYNCPOINT command.

Your transaction design ensures that all participating transactions are in the correct conversation state when a syncpoint is taken. It should not be necessary for one transaction program to force a state change on another before issuing a SYNCPOINT command.

Syncpoint examples

CICS transaction syncpointing over an LUTYPE6.2 link involves a two-phase commit process. This means that the flows that go to partner transactions are not all the same.

In phase 1, resources are prepared for commitment; in phase 2 they are committed. In conjunction with this, one partner is selected to start the phase 2 commitment.

In this section, you should be aware of the specialized terminology used for syncpoint exchanges. These are defined below. (The 'flows' are defined according to the LUTYPE6.2 architecture. For full details, see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*).

Partner
Generally means the other end of an LUTYPE6.2 conversation. However, here we use the term to mean all LUTYPE6.2 conversations **except** the one that has received a flow from an initiator.

Initiator
The end of an LUTYPE6.2 conversation that initiates the syncpoint by sending either a 'Prepare to commit' or a 'Request commit' flow to a partner.

Agent
The end of an LUTYPE6.2 conversation that receives either a 'Prepare to commit' or a 'Request commit' sent by an initiator. If the conversation has associated partners, it becomes the initiator for these conversations.

Last agent
One of the partners selected by an initiator to initiate the second phase of commitment. The partner selected is random, and depends on processing conditions.

'Prepare to commit'
A flow sent from an initiator to an agent requesting the agent to ready its resources for commitment.

'Request commit'
Sent from an agent to an initiator to say that the agent has readied all its resources for commitment, or sent by an initiator to an agent to get the agent to commit all the resources it has previously readied.

'Committed'
Sent from an agent to an initiator in response to 'Request commit', to indicate that the agent has committed all its resources; or sent from an initiator to an agent to say that the initiator has committed its resources, and that the agent should undergo the second phase of commitment and commit its resources.

'Forget'
Sent from an agent to an initiator in response to a 'Committed' to show that it has committed all its resources.

'Rollback'
Sent from an agent to an initiator to signify the rejection of a 'Prepare to commit' or 'Request commit' request, and shows that the agent has instead rolled back its resources. When sent from an initiator to an agent, it tells the agent to roll back its resources.

A successful two-phase syncpoint works like this :

1. The initiator sends 'Prepare to commit' to all its partners except one (the last agent).
2. The not-last agents become local initiators for their partners, and send 'Prepare to commit' to ***all*** of them. In this way, the syncpoint request cascades to all distributed members of the conversations.
3. All these agents return 'Request commit' to their initiators.
4. The initiator now sends 'Request commit' to its last agent, so starting the second phase.
5. The last agent may act as a syncpoint initiator, and so sends out 'Prepare to commit' and/or 'Request commit' to its partners.
6. All these agents respond positively.
7. The last agent commits its resources, and replies 'Committed'.
8. The initiator now commits its own resources, and sends out 'Committed' to all its not-last agents.
9. These agents now commit their resources, and reply with a 'Forget' flow (this 'Committed/Forget' sequence is similarly cascaded).
10. When the initiator has received all the Forget flows, the syncpoint has completed.

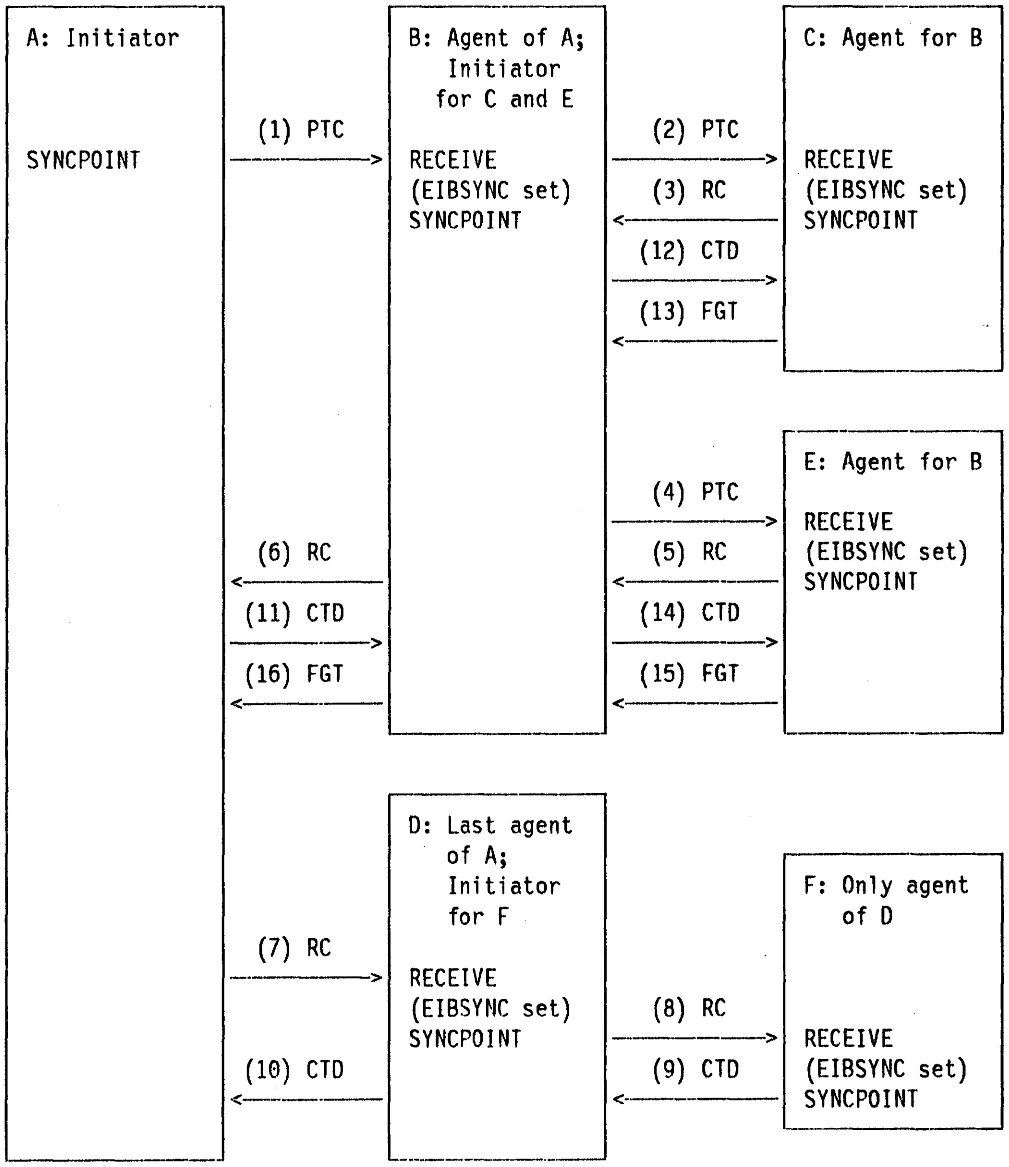
Figure 57 on page 189 shows the flows that result from a simple distributed transaction executing a successful two-phase syncpoint such as the one that has just been described.

An unsuccessful syncpoint scenario occurs when an agent, instead of returning 'Request commit' or 'Committed', sends a 'Rollback' flow. This then forces each level of initiator to send a 'Rollback' request to its partners, so backing out the commitment process. Because the commitment process is two-phased, this means that the local transactions either abandon the phase 1 ready-state and back out instead, or do a direct backout.

Figure 58 on page 193 shows the flows that result from a simple distributed transaction executing an unsuccessful syncpoint and triggering a syncpoint rollback.

In general, you do not need to be aware of the details of these flows — just the concepts. See "Part 5. Recovery and restart" on page 287 to obtain information on what happens when a syncpoint exchange fails, and on how to deal with this situation.

When you follow the rule about testing the EIB flags after each LUTYPE6.2 operation, you test EIBSYNC, EIBSYNRB, and EIBRLDBK to see what has happened throughout your distributed transaction. You do not need to know where the syncpoint has originated; just follow the rules, and reply positively or negatively to the request.



PTC - Prepare to Commit	CTD - Committed
RC - Request Commit	FGT - Forget

Figure 57. A successful distributed syncpoint

Note: In the following explanation to Figure 57 on page 189, the numbered references in italics refer to the flow of control indicators across the links and other action initiated by CICS on behalf of the transactions. The programmer is concerned only with issuing SYNCPOINT in response to finding EIBSYNC set.

Transaction A issues a SYNCPOINT command. It is in *send state* on its conversations with transactions B and D.

(1) Transaction A has more than one partner, so it must prepare all its partners but one (the 'last agent') for syncpointing by sending a 'prepare to commit' flow. This is part 1 of the two-phase commit process.

Transaction B sees that EIBSYNC is set, so it issues a SYNCPOINT command. This transaction is responding to a request from transaction A, but it also becomes the syncpoint initiator for transactions C and E, and must ensure that its conversations with these transactions are in send state.

(2) As a syncpoint initiator, transaction B sends 'prepare to commit' to transaction C.

Transaction C sees that EIBSYNC is set, so it issues a SYNCPOINT command.

(3) Transaction C returns 'request commit' to transaction B.

(4) Transaction B has received a 'prepare to commit' from its syncpoint initiator A, therefore the concept of a 'last agent' does not apply. It sends the 'prepare to commit' to all its other partners (only transaction E in this case).

Transaction E sees that EIBSYNC is set, so it issues a SYNCPOINT command.

(5) Transaction E returns 'request commit' to transaction B.

(6) Transaction B has now received 'request commit' from all its partners — they have all responded positively to the 'prepare to commit' flow. Therefore, transaction B returns the positive response of 'request commit' to its syncpoint initiator (transaction A).

(7) Transaction A has now sent 'prepare to commit' to, and received 'request commit' from, all its partners except the 'last agent'. It is time to get a commitment of all its distributed resources. Thus, it sends 'request commit' to its 'last agent', transaction D.

Transaction D sees that EIBSYNC is set, so it issues a SYNCPOINT command. This transaction is responding to a request from transaction A, but it also becomes the syncpoint initiator for transaction F, and must ensure that its conversation with this transaction is in send state.

(8) Because transaction D has only one partner, it sends a 'request commit' flow to transaction F.

Transaction F sees that EIBSYNC is set, so it issues a SYNCPOINT command.

All the transactions have now indicated, by issuing SYNCPOINT commands, that they are ready to commit their changes. This process begins with transaction F, which has no agents and has responded to 'request commit' by issuing a SYNCPOINT command.

(9) Transaction F commits its resources and returns 'committed' to transaction D.

(10) Transaction D has received a positive response to its 'request commit' request, so it commits its own resources and responds positively to its syncpoint initiator by issuing a 'committed' flow to transaction A.

(11) Transaction A has received a positive response from its 'last agent'. It can now commit its own resources. Therefore, it has to signal to all its other partners that the second part of the two-phase syncpoint can occur, so getting resource commitment. It does this by sending a 'committed' flow to transaction B.

(12, 14) B receives the 'committed' flow from its initiator. Because there is no 'last agent' in this case, it sends a 'committed' flow to its partners to request them to commit their resources.

(13, 15) The agents receive a 'committed' flow from their initiator. They commit their local resources and return a 'forget' flow to the initiator.

(16) B has successfully completed the commitment of its own resources and those held by its partners. It signals this success by returning 'forget' to its initiator, transaction A.

The distributed syncpoint is complete and control returns to transaction A following the SYNCPOINT command.

Negative responses to syncpoint requests

The previous discussion of the SYNCPOINT command assumed that all the agent transactions were ready to take a syncpoint by issuing SYNCPOINT in response to the syncpoint request (EIBSYNC set).

If, however, an agent has detected an error, it can reject the syncpoint request by means of one of the following commands (see Figure 68 on page 215):

- SYNCPOINT ROLLBACK (preferred response)
- ISSUE ERROR
- ISSUE ABEND.

The SYNCPOINT ROLLBACK command (see "The SYNCPOINT ROLLBACK command" on page 195) enables a transaction to initiate a back-out operation across the whole distributed unit of work. When it is issued in response to a syncpoint request, it has the following effects:

1. Any changes made to recoverable resources by the transaction that issues the rollback request are backed out.
2. The syncpoint initiator is also backed out (EIBRLDBK set).

This will cause the syncpoint initiator to initiate a back-out operation across the distributed unit of work.

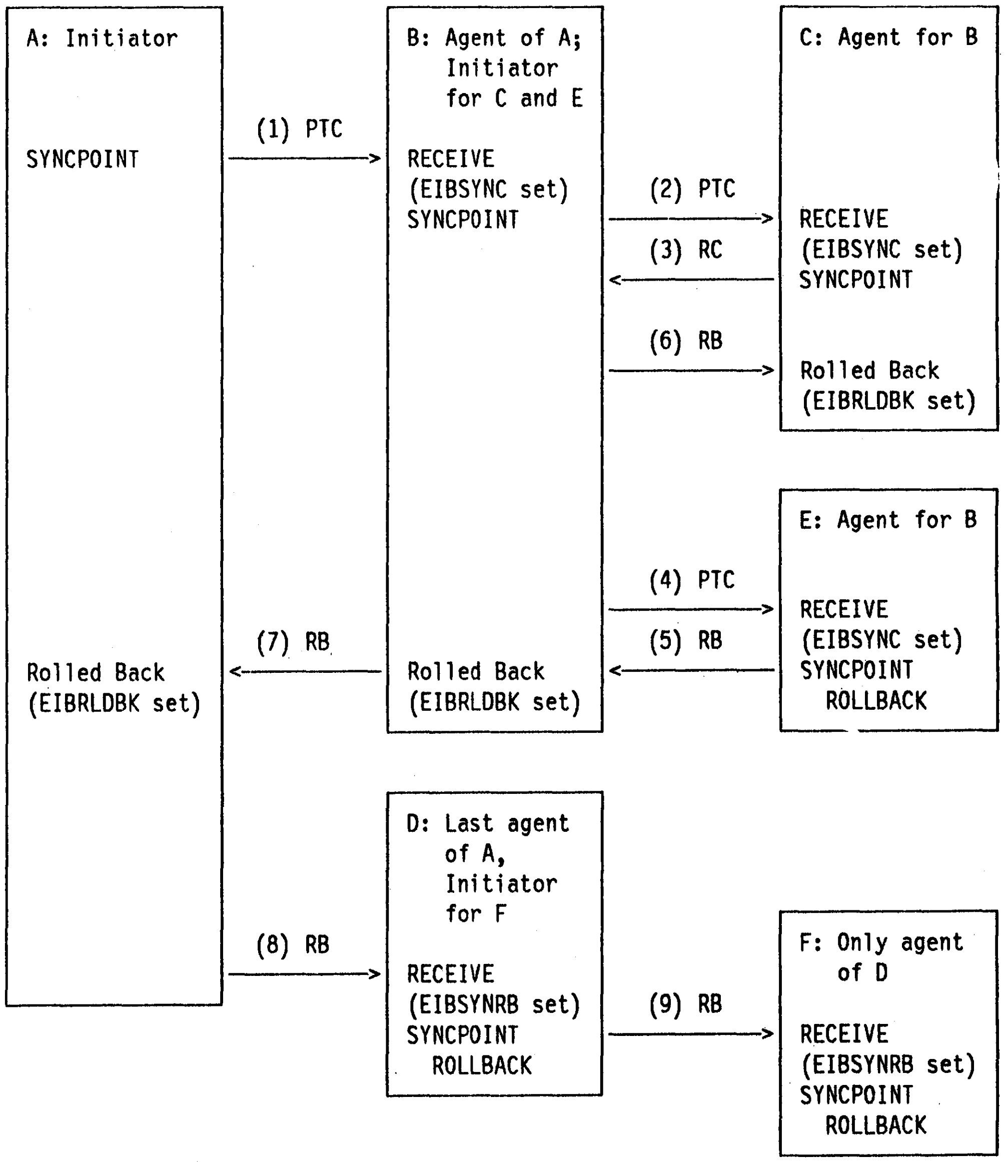
Implicit syncpoint at transaction termination

If a transaction terminates normally, CICS takes a syncpoint. If a connected transaction rejects the syncpoint with a negative response, the terminating transaction is correctly backed out, but, because control has already returned to CICS, it can no longer test whether the syncpoint was successful or not.

To avoid this situation, you should code EXEC CICS SYNCPOINT and test EIB flags before terminating.

This situation is equivalent to the one described in "The LAST option and syncpoint flows" on page 181, which explains the consequences of not coding an explicit SYNCPOINT (by an EXEC CICS SYNCPOINT command) before the transaction terminates.

Syncpoint rollback example



PTC - Prepare to Commit RC - Request Commit RB - Roll Back

Note: A positive response (+RSP) is returned after each RB. For the sake of clarity, these have been omitted.

Figure 58. Rollback during distributed syncpointing

Figure 58 on page 193 shows the same distributed syncpoint as Figure 57 on page 189. In this case, however, transaction E has detected an error that makes it unable to commit, and it issues SYNCPOINT ROLLBACK when it sees that EIBSYNC is set. This causes any changes that transaction E has made to be backed out, and initiates a distributed rollback.

(5) E has detected an error whilst readying its local resources during the phase 1 commit process. It therefore wants to rollback its own resources by issuing an EXEC CICS SYNCPOINT ROLLBACK command. This means that it has responded negatively to the 'prepare to commit' request as a 'rollback' response is issued by CICS to its initiator, B.

(6) B now must send the 'rollback' request to all its other partners — only C in this case. C reacts by rolling back its own local resources (and would cascade the 'rollback' to its partners if it had any). C returns a positive response, which has been omitted from the diagram for clarity.

(7) B has now rolled back its own resources, and ensured that all its partners have done the same. It replies to its initiator's 'prepare to commit' request negatively with the 'rollback' flow.

(8) Transaction A now receives the 'rollback' response, and so rolls back its own local resources. Transaction A then sends the 'rollback' request to all its partners except the one that returned the 'rollback' response. In this case, it only sends 'rollback' to transaction D.

(9) Transaction D sees EIBSYNRB, so it issues its own SYNCPOINT ROLLBACK, which causes the 'rollback' flow to its partners (transaction F in this case).

Transaction F sees that EIBSYNRB is set, and issues a SYNCPOINT ROLLBACK command. The distributed rollback is now complete.

This example shows how a SYNCPOINT ROLLBACK flows through the distributed transaction, and how the EIBRLDBK (rollback **has** occurred) and EIBSYNRB (take a rollback **now**) flags show what has happened and what should happen.

The SYNCPOINT ROLLBACK command is the recommended negative response to a syncpoint request (EIBSYNC set). However, ISSUE ERROR and ISSUE ABEND are also allowed.

The ISSUE ERROR command, when it is issued in response to a syncpoint request, causes the transaction that has issued the SYNCPOINT command to abend. The abend is propagated to the other transactions in the distributed unit of work, and dynamic backout occurs.

The transaction that issued the ISSUE ERROR command is not abended because this conversation is placed in SEND state. However, because the partner is no longer present, the conversation is in an unusual state. The state diagrams in figures Figure 62 on page 210 through Figure 71 on page 216 show what you should do next. Here is the sequence of commands to achieve an orderly termination of the conversation on your side:

```
(1) EXEC CICS SEND INVITE WAIT      –to go into Receive state
(2) EXEC CICS RECEIVE               –to set the EIB flags
(3) Examine the EIB                 –EIBSYNRB & EIBFREE are set
(4) EXEC CICS SYNCPOINT ROLLBACK
(5) EXEC CICS FREE
```

The ISSUE ABEND response causes the initiator to see a conversation abend. Only consider using ISSUE ABEND in response to a syncpoint request when you have not updated any recoverable resources in your transaction and its partners.

SYNCPOINT ROLLBACK is the preferred negative response to a SYNCPOINT request, with ISSUE ERROR as an alternative. Do not use ISSUE ABEND unless you have to.

The SYNCPOINT ROLLBACK command

A transaction can initiate a rollback at any time, not only in response to a syncpoint request. If a transaction issues a SYNCPOINT ROLLBACK command, the current logical unit of work is backed out unconditionally. In addition, the rollback request is transmitted to all the transaction's SL(2) conversation partners.

You can issue a SYNCPOINT ROLLBACK command irrespective of the send/receive state of your conversations. If the rollback command is issued when you have a conversation in receive state, incoming data on that conversation is purged in the way described for the ISSUE ERROR and ISSUE ABEND command.

If you receive a rollback request (EIBSYNRB set) you must respond with a SYNCPOINT ROLLBACK command. On return from the command, you will be in *receive state*.

If you are the initiator of rollback, your conversation is in *send state* after the rollback has completed. Your conversation partner, after responding with SYNCPOINT ROLLBACK, is in *receive state*. If, however, you and your partner issue SYNCPOINT ROLLBACK at the same time, CICS leaves you in the opposite state to the one you were in when you gave the command (see "CICS deviations from LUTYPE6.2 architecture" on page 440). Because this puts you in an ambiguous conversation state, you should design your transactions to avoid such *race* situations.

The ISSUE PREPARE command

The use of the EXEC CICS SYNCPOINT command sends either a 'prepare to commit' or a 'request commit' flow to all the partners; it affects the whole of the distributed task. However, you can send the first part of the two-phase commit process to a given partner by means of the ISSUE PREPARE command. This enables you to ready a portion of the distributed task, or handle a negative response sent from the partner on a conversation-by-conversation basis.

There are several reasons why you might want to do this:

1. In complex distributed transaction processing involving several conversing transactions, an ISSUE ERROR command issued by one of the transactions may not reach the syncpoint initiator in time to prevent it from issuing a SYNCPOINT command and initiating some syncpoint processing for itself and other partners. This can lead to the need for complex backout procedures for the distributed unit of work.

   You can use ISSUE PREPARE as a way of flushing any error responses from the network.

2. You may have one or more syncpoint partners that are not completely 'reliable'. You can use ISSUE PREPARE to check that all is well with these partners before proceeding with a general distributed syncpoint.

The format of the ISSUE PREPARE command is:

```
ISSUE PREPARE CONVID(conversation-name)
```

From this you will see that, unlike the SYNCPOINT and SYNCPOINT ROLLBACK commands, the ISSUE PREPARE command is directed to a specific conversation. Your transaction must be in send state on any conversation to which it issues ISSUE PREPARE.

Receiving and replying to ISSUE PREPARE

If a partner transaction issues an ISSUE PREPARE command, EIBSYNC will appear in the EIB.

EIBSYNC is the same flag that is used to notify the receipt of a distributed SYNCPOINT command. CICS does not tell you which command was issued, and you should not make any assumptions.

Consequently, the responses to an ISSUE PREPARE are exactly the same as for a SYNCPOINT: SYNCPOINT, SYNCPOINT ROLLBACK, ISSUE ERROR, or ISSUE ABEND. You can therefore use subsequent ISSUE PREPAREs for partner conversations before taking the decision as to which response is appropriate.

Checking the response to ISSUE PREPARE

After issuing ISSUE PREPARE you must check the response by testing EIBERR.

If EIBERR is set, and the first two bytes of EIBERRCD contain X'0889', the other transaction has sent ISSUE ERROR. In this case, your conversation is in RECEIVE state, so you should issue a RECEIVE command and test the EIB in the usual way. Your application design will determine what happens next.

If the partner transaction issued a SEND LAST WAIT command (as shown by EIBFREE), the conversation is ended. It cannot participate in further syncpoint activity, and you should end it by issuing a FREE command. However, if instead your partner continued via a SEND INVITE WAIT command (the recommended action after ISSUE ERROR), you can either free the conversation if it is not important for syncpointing, or use SYNCPOINT ROLLBACK to initiate a distributed rollback.

If EIBERR is set, and the first two bytes of EIBERRCD contain X'0824', the other transaction has replied SYNCPOINT ROLLBACK. (EIBSYNRB will also be set.) In this case, you must propagate the rollback by issuing SYNCPOINT ROLLBACK in the usual way.

If the other transaction has replied ISSUE ABEND, the TERMERR condition will be raised in your transaction. You can choose to handle this condition (or the default condition ERROR), or you can use IGNORE or NOHANDLE to disable the condition, and check what has happened by testing EIBERRCD for X'0864'. In this case, the conversation is no longer usable, and the other transaction can take no part in subsequent syncpointing activity.

If EIBERR is not set, the other transaction has issued SYNCPOINT, indicating that it is ready for syncpointing.

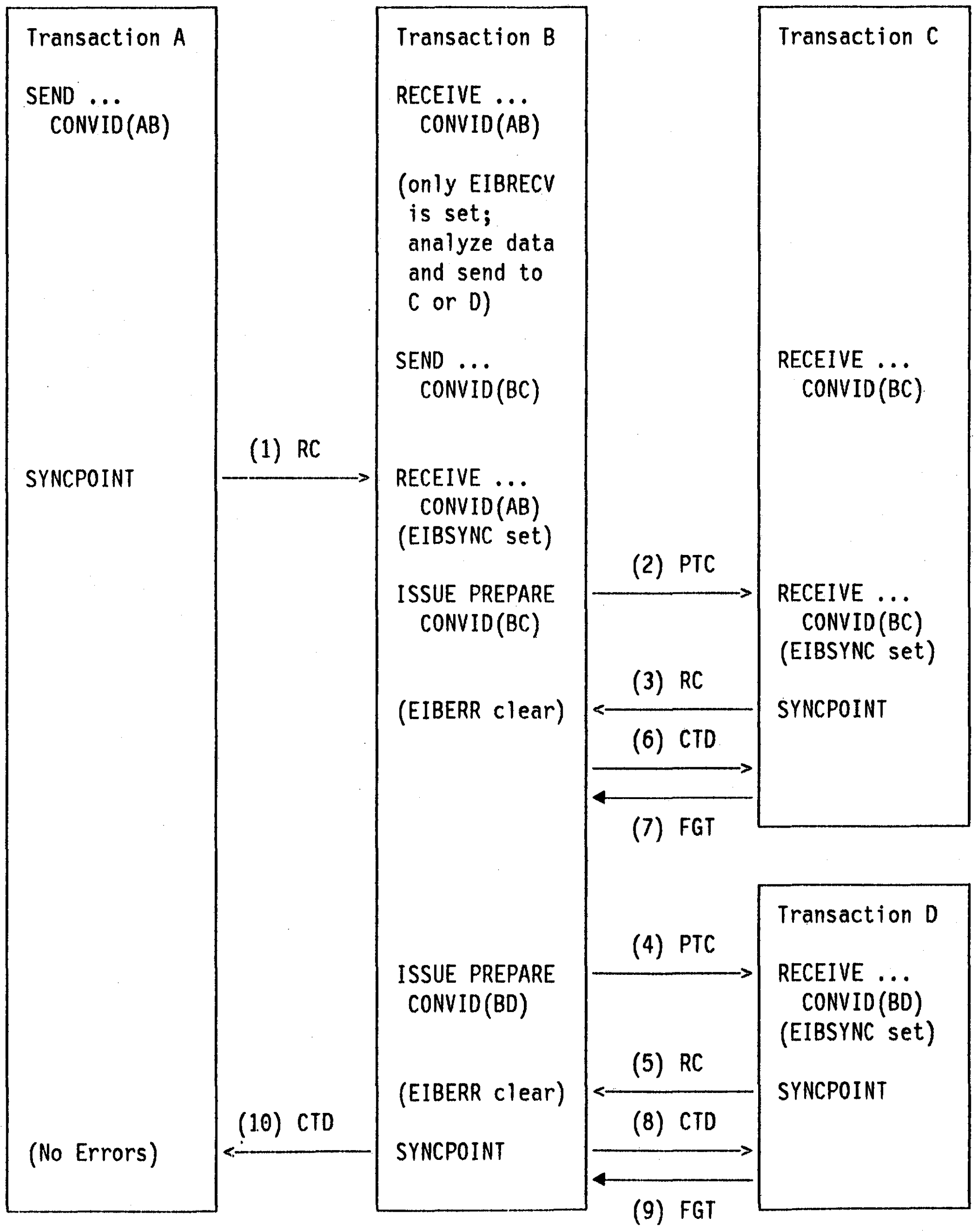
If any partner has responded positively (EIBERR not set) to an ISSUE PREPARE command, you must eventually issue either SYNCPOINT or SYNCPOINT ROLLBACK to cause that partner to commit or backout.

ISSUE PREPARE examples

Figure 59 on page 198 shows four conversing transactions A, B, C, and D.

It is assumed that transaction A is designed to send records to transaction B and issue a SYNCPOINT when it has sent a certain number of records. Transaction B examines each record and decides whether to send it to transaction C or transaction D for processing. Transactions C and D are both designed to issue ISSUE ERROR immediately if they find an error in one of the records.

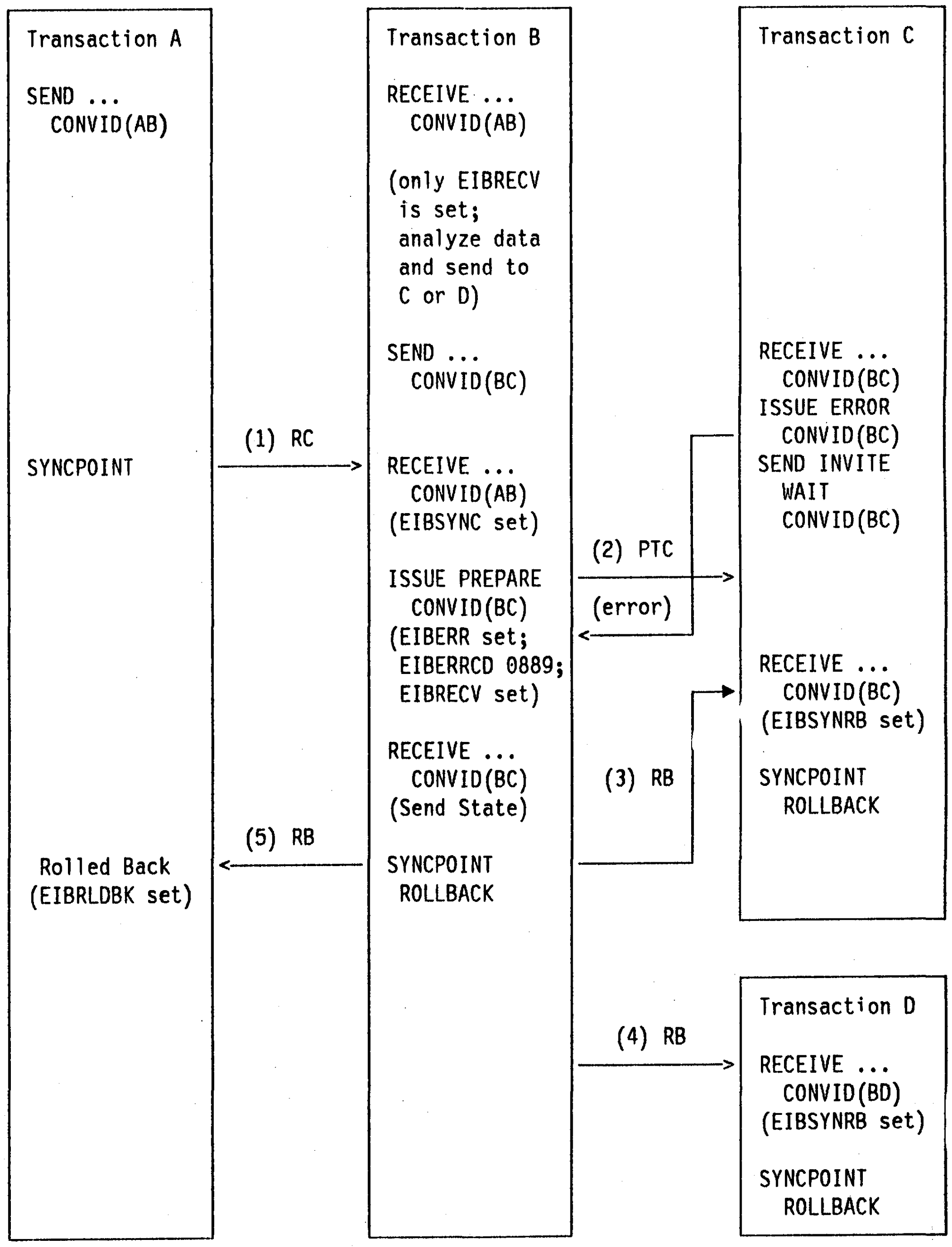
Because transactions C and D can send ISSUE ERROR at any time, transaction B is designed to use ISSUE PREPARE to find out whether any incoming error responses have not yet been received.



PTC - Prepare to Commit   RC - Request Commit   CTD - Committed   FGT - Forget

Figure 59. ISSUE PREPARE with positive responses

In Figure 59, there are no error responses outstanding, and both transactions respond positively to EIBSYNC. Transaction B can therefore go ahead and issue the SYNCPOINT command. If you compare this example with Figure 57 on page 189, you will see that the flows generated by CICS are the same. In Figure 60 on page 199, transaction C has issued ISSUE ERROR after receiving a record, and transaction B sees the error on return from the ISSUE PREPARE command. After the normal continuation after receiving ISSUE ERROR, transaction B issues SYNCPOINT ROLLBACK. The rollback is propagated in a similar manner to that shown in Figure 58 on page 193.


Transaction A
SEND ... CONVID(AB)
SYNCPOINT
Rolled Back (EIBRLDBK set)
Transaction B
RECEIVE ... CONVID(AB)
(only EIBRECV is set; analyze data and send to C or D)
SEND ... CONVID(BC)
RECEIVE ... CONVID(AB) (EIBSYNC set)
ISSUE PREPARE CONVID(BC) (EIBERR set; EIBERRCD 0889; EIBRECV set)
RECEIVE ... CONVID(BC) (Send State)
SYNCPOINT ROLLBACK
Transaction C
RECEIVE ... CONVID(BC)
ISSUE ERROR CONVID(BC)
SEND INVITE WAIT CONVID(BC)
RECEIVE ... CONVID(BC) (EIBSYNRB set)
SYNCPOINT ROLLBACK
Transaction D
RECEIVE ... CONVID(BD) (EIBSYNRB set)
SYNCPOINT ROLLBACK
(1) RC
(2) PTC
(error)
(3) RB
(4) RB
(5) RB

PTC - Prepare to Commit

Note: The positive response (+RSP) after each RB
RB - Rollback has been omitted for the sake of clarity.

Figure 60. ISSUE PREPARE with error response

This example shows just one of many possible application designs using ISSUE PREPARE.

You are advised to restrict the use of ISSUE PREPARE to complex syncpointing situations. In most cases, the use of SYNCPOINT and SYNCPOINT ROLLBACK will be sufficient.

Sending and receiving signals

The ISSUE SIGNAL command sends an SNA signal as an expedited flow command. This can be used by an application in receive state to notify its partner that it needs to get into SEND state. For example, this would be useful if too much data was currently being sent.

If an application receives an incoming signal then the SIGNAL condition will be raised and the field EIBSIG will be set. Note that EIBSIG is sent only once. If you miss it, or choose not to take any action in response to it, nothing happens — the partner's request is never satisfied.

Freeing the session

The command used to free the session has the following format:

```
FREE CONVID(name)

NOTALLOC, INVREQ
```

where "name" is the name of the conversation. The FREE command is normally used to free the session after the conversation has been terminated (for example, after SEND LAST WAIT has been issued). However, you can issue the FREE command at any time that your transaction is in send state. CICS determines whether the end-bracket indicator has already been transmitted, and transmits it if necessary before freeing the session. If there is also deferred data to transmit, the end-bracket indicator is transmitted with the data. Otherwise, the indicator is transmitted by itself.

If you terminate an SL(2) conversation with one of these command sequences:

- SEND LAST followed by FREE
- SEND LAST WAIT
- FREE in *send state*.

you divide the distributed transaction into two parts which continue to execute independently. Consequently, a subsequent SYNCPOINT or SYNCPOINT ROLLBACK command in either part is not propagated to the other.

This means that protected resources in one part o' the divided transaction could be committed, while those in the other part could ʟ backed out.

To safeguard against this, you should always terminate sync SL(2) conversations using the sequence:

```
SEND LAST
SYNCPOINT
FREE
```

The EXEC interface block (EIB)

Full details of the EIB are given in the *CICS/MVS Application Programmer's Reference* manual. This section highlights the fields that are of particular significance in LUTYPE6.2 applications. For further details of how and when these fields should be tested, or saved, refer to "Command sequences on LUTYPE6.2 mapped conversations" on page 203.

Conversation identifier fields

The following EIB fields enable you to obtain the name of the LUTYPE6.2 conversation.

EIBTRMID
contains the name of the principal facility. For a back-end transaction it is the conversation identifier (CONVID). You must acquire this name if you want to state the CONVID for the principal facility explicitly.

EIBRSRCE
contains the conversation identifier (CONVID) for the session obtained by means of an ALLOCATE statement. You must acquire this name immediately after issuing the ALLOCATE statement.

Procedural and error fields

These fields contain information on the state of the conversation and on the various indicators that are transmitted by the conversing transaction. In most cases, the settings relate to the session named in the last-executed RECEIVE or CONVERSE command, and should be tested, or saved for later testing, after the command has been issued. Further information on the use of these fields is given in "Command sequences on LUTYPE6.2 mapped conversations" on page 203.

EIBRECV
indicates the conversation state following RECEIVE or CONVERSE. If it is *off* (= X'00'), your conversation partner is inviting you to send, otherwise you would normally issue a further RECEIVE command. It does not necessarily reflect *receive state* at any other time.

EIBCOMPL
This field is used in conjunction with the RECEIVE NOTRUNCATE command; it is set to indicate that the data is complete.

EIBSYNC
indicates that CICS syncpointing is in progress and that the application should issue a SYNCPOINT command.

EIBSYNRB
indicates that CICS syncpointing is in progress and that the application should issue a SYNCPOINT ROLLBACK command.

EIBRLDBK
indicates that the remote transaction has sent SYNCPOINT ROLLBACK in response to a SYNCPOINT request. The transaction that issued the SYNCPOINT command has been rolled back.

EIBCONF
indicates that the conversation partner has issued a SEND CONFIRM command, and that a response is required.

EIBSIG
indicates that the conversation partner has issued an ISSUE SIGNAL command.

EIBFREE
indicates that the receiver must issue a FREE command for the session.

EIBERR
indicates that an abnormal condition has occurred. The reason is in EIBERRCD.

EIBERRCD
contains the reason for EIBERR.

The meanings of the various EIBERRCD values are given in Figure 61 on page 209. Three values that can arise as part of designed error-signaling in the conversing transactions are:

- X'08240000' — the conversation partner has issued a SYNCPOINT ROLLBACK command.
- X'08640000' — the conversation partner has issued an ISSUE ABEND command.
- X'08890000' — the conversation partner has issued an ISSUE ERROR command.

EIBNODAT
indicates that no application data has been received. This means that the remote system has generated a null request unit to convey conversation control information.

Fields that are not applicable

The following EIB fields are not applicable to LUTYPE6.2 mapped conversations:

EIBEOC
The EIBEOC is intended for use when the user program is doing its own chain assembly. Because chain assembly is not supported for LUTYPE6.2 conversations, EIBEOC is set on every RECEIVE command, and can be ignored.

EIBFMH
Because function management headers (FMH) are never passed to LUTYPE6.2 application programs, EIBFMH is never set and can be ignored.

Fields and synchronization levels

This table shows you how different fields are relevant at different levels of synchronization.

Table 6. EIB fields and synclevels at which they are relevant

Field name	SL(0)	SL(1)	SL(2)
EIBCOMPL	Y	Y	Y
EIBCONF		Y	Y
EIBERR & EIBERRCD	Y	Y	Y
EIBFREE	Y	Y	Y
EIBNODAT	Y	Y	Y
EIBRECV	Y	Y	Y
EIBRLDBK			Y
EIBSIG	Y	Y	Y
EIBSYNC			Y
EIBSYNRB			Y

Command sequences on LUTYPE6.2 mapped conversations

The command sequences that you use to communicate between the front-end and the back-end transactions are governed both by the requirements of your application and by a set of high-level protocols designed to ensure that commands are not issued in inappropriate circumstances.

The protocols presented in this section do not cover all possible command sequences. However, by following them, you will ensure that each transaction takes account of the requirements of the other, and hence reduce the error rate during program development.

The protocols are based on the concept of a number of states. These states apply only to the particular conversation, not to your application program as a whole. In each state, there are a number of commands that might most reasonably be issued. After the command has been issued, fields in the EIB must be tested in the order shown in the state diagrams, figures 62 through 71, to check on the current requirements of the conversation. The results of these tests, together with the command that has been issued, may cause a transition to another state, in which another set of commands becomes appropriate.

The states that are defined for the purposes of this section are:

- State 1 – session not allocated
- State 2 – session allocated
- State 3 – send state
- State 4 – receive pending after INVITE
- State 5 – receive state
- State 6 – receiver issue confirmation
- State 7 – receiver take syncpoint
- State 8 – receiver take rollback
- State 9 – free pending after SEND LAST
- State 10 – free session.

Initial states

The front-end transaction in a conversation will initially be in state 1 – session not allocated – and must issue an ALLOCATE command to acquire a session.

An exception to this occurs when the front-end transaction is started by automatic transaction initiation (ATI), in the local system, with an LUTYPE6.2 session as its principal facility. In this case, the session is already allocated, and the transaction is in state 2. For transactions of this type, you must immediately obtain the conversation name from EIBTRMID if you want to be able to name the conversation explicitly on subsequent commands.

The back-end transaction is initiated in RECEIVE state as a result of the CONNECT PROCESS command issued by the front-end transaction. However, to initiate the conversation properly, you must issue an EXEC CICS RECEIVE command before you do anything else that could affect the link (for example, a SYNCPOINT ROLLBACK command). The only exception to this rule is that you can issue an EXTRACT PROCESS command, before EXEC CICS RECEIVE, to obtain the synchronization level and other conversation-related information.

State diagrams

Figure 61 on page 209 through Figure 71 on page 216 are intended to enable you to construct valid command sequences. Each diagram relates to one particular state, as previously defined, and shows the commands that you might reasonably issue and the tests that you must make after issuing the command. Where more than one test is shown, they must be made in the order indicated.

The combination of the command issued and a particular positive test result lead to a resultant state, shown in the final column.

Note that an ISSUE SIGNAL command is always valid within an allocated LUTYPE6.2 session.

Using the state diagrams

As a guide to using the state diagrams, consider a front-end transaction that is designed to invoke a remote back-end transaction and send it some sort of file-search criteria. The back-end transaction is expected to send each record that matches the search criteria, and then issue a SYNCPOINT and free the session.

Note: The use of SYNCPOINT in an inquiry-only application is normally not required. It is included in this example only to illustrate the use of the state diagrams.

Allocating a session

Initially, the front-end transaction has no session to the remote system. It is therefore in **state 1 – session not allocated**.

The diagram for state 1 shows that you need to issue an ALLOCATE command:

```
EXEC CICS ALLOCATE SYSID(RSYS)
```

Where RSYS is the CONNECTION name (or SYSIDNT) of the remote system.

You should now check for SYSIDERR and SYSBUSY to make sure that a session has been allocated. If all is well, you must immediately get the conversation identifier from EIBRSRCE, for example:

```
MVC CNAME,EIBRSRCE
```

You must name the conversation explicitly in the commands that follow. The diagram shows that the conversation is now in **state 2 – session allocated**.

Connecting the remote process

The diagram for state 2 shows that the next command to issue is CONNECT PROCESS:

```
EXEC CICS CONNECT PROCESS CONVID(CNAME) SYNCLEVEL(2)
                  PROCNAME('REMT') PROCLENGTH(4)
```

Here the name of the back-end transaction is REMT and level 2 synchronization is specified. The state 2 diagram shows that, after this command, the conversation is in **state 3 – send state**.

The SEND command

The next thing to do is to send the file-search criteria to the back-end transaction. This is to be a single transmission, after which you expect to receive the matching records from the back-end transaction. Your aim, therefore, is to get into **state 5 – receive state**, so that you can issue RECEIVE commands.

The diagram for state 3 shows that you now have several ways of continuing after the CONNECT PROCESS command.

1. The first possibility is to use a simple SEND command. The diagram shows that, after a SEND command, the conversation is still in **state 3 — send state**, not in the required receive state.

   However, looking further down the table, you will see that you can still go right ahead and issue your RECEIVE command. CICS will then INVITE the back-end transaction to send (remember that the back-end transaction is initialized in **state 5 — receive state**) and WAIT to make the actual transmission occur.

   The table shows that CICS will supply the INVITE and the WAIT when you issue a SEND followed immediately by a RECEIVE. However, if you look at the send-state table for **unmapped** LUTYPE6.2 conversations (Figure 78 on page 239) you will see that in that case you have to supply the INVITE and WAIT options yourself.

2. The next possibility is to use SEND INVITE; that is, to specify the INVITE explicitly. The table shows that after SEND INVITE the conversation is in **state 4 — receive pending after invite**. If you look at the diagram for state 4, you will see that you can now issue an explicit WAIT command to get to state 5, or, as before, just issue the RECEIVE command.

3. The third possibility is to specify INVITE and WAIT explicitly on the SEND command. The table shows that, after SEND INVITE WAIT, the conversation is in **state 5 — receive state**.

Suppose that you decide to use the third option:

```
EXEC CICS SEND INVITE WAIT CONVID(CNAME) FROM(OUTAREA) LENGTH(OUTLEN)
```

The RECEIVE commands

You are now ready to issue RECEIVE commands to get the records from the back-end transaction:

```
EXEC CICS RECEIVE CONVID(CNAME) INTO(INREC) LENGTH(RECLEN)
```

The diagram for state 5 shows the tests you must make after issuing a RECEIVE command. As you will see, there are up to six tests that you may have to make, depending on the synchronization level established for the conversation (level 2 in this example). As well as showing which fields must be tested, the state table also shows the order in which you must make the tests.

Because the RECEIVE command in this example did not use the NOTRUNCATE option, you do not need to test EIBCOMPL. We are assuming that the two transactions know the maximum length of data that they will be handling, and that overlength data would be a logic error that would cause the LENGERR condition to be raised. The use of the NOTRUNCATE option, together with EIBCOMPL tests, would be appropriate in programs that are communicating with systems or devices that can send data of no defined maximum length.

After issuing the RECEIVE command (and checking for EIBCOMPL if this is appropriate) the first thing to do is to save the EIB values. (You need the values that were set by the RECEIVE command, and you may have to use other commands in the meantime that can change the EIB.) The following tests should then be made on the saved values of the EIB fields, not on the EIB itself.

The first test for level 2 synchronization is EIBCONF. If you know that the other transaction will never issue the SEND CONFIRM command, as in this example, you can omit this test. You are advised not to make any similar assumptions about EIBSYNC and EIBSYNRB, but to test them after every RECEIVE command on a synchronization-level 2 conversation. However, to keep things simple, the syncpoint rollback test (EIBSYNRB) is left out of the following description. Your tests can then proceed as follows:

1. Test EIBSYNC

   a. **Set**

      The state 5 diagram shows that the conversation is now in **state 7 – receiver take syncpoint**, so go to the state 7 diagram. Unless you want to inform the back-end transaction about some problem, the appropriate command is:

      ```
      EXEC CICS SYNCPOINT
      ```

      Using the state 7 diagram, you see that the next thing to do is to test the **saved value** of EIBFREE. If this is set, the conversation is in **state 10 – free session**, and you must:

      ```
      EXEC CICS FREE CONVID(CNAME)
      ```

      If EIBFREE is not set, you must test the saved value of EIBRECV. If EIBRECV is set, the conversation is back in **state 5 – receive state**, and you can issue your next RECEIVE command.

      If neither EIBFREE nor EIBRECV is set, the conversation is in **state 3 – send state**. This is not expected in this example, but you should test for it and take some action; for example, ISSUE ERROR.

      Note: You would expect to be in send state if, for example, the back-end transaction was designed to use:

      ```
      EXEC CICS SEND INVITE
      EXEC CICS SYNCPOINT
      ```

      You can check that sequence in the state diagrams to see how it works.

   b. **Not set**

      Make test 2.

2. Test EIBFREE

   a. **Set**

      The conversation is now in **state 10 — free session**, and you must:

      ```
      EXEC CICS FREE CONVID(CNAME)
      ```

   b. **Not set**

      Make test 3.

3. Test EIBRECV

   a. **Set**

      The conversation is still in receive state, so you can issue your next RECEIVE command.

   b. **Not set**

      As described earlier, this is not expected in this example.

Testing EIBERR and EIBSYNRB

The state diagrams in this section do not show the EIBERR tests for individual commands.

EIBERR can be set at any time that your transaction is in receive state, and also following any command that causes a transmission to the remote system. It is safest to test EIBERR after every command. If EIBERR is set, there will be an associated error code in EIBERRCD. EIBERRCD values are listed in Figure 61 on page 209.

If EIBERR is set with an EIBERRCD of X'0889' (ISSUE ERROR received), your transaction is in receive state, and you should issue a RECEIVE command.

If your application design includes the use of SYNCPOINT ROLLBACK, similar considerations apply. When the transaction is required to execute a SYNCPOINT ROLLBACK command, EIBSYNRB is set as well as EIBERR, with an EIBERRCD of X'0824'.

Other tests

Tests for other conditions that may possibly arise, for example, INVREQ or NOTALLOC, should be made in the normal way. Further information on these errors, if any, is available in EIBRCODE.

Also, if your transaction is expected to receive an incoming SIGNAL command, you should either execute a HANDLE command for the SIGNAL condition or test EIBSIG after each command.

ALL STATES	MAPPED LUTYPE6.2 CONVERSATIONS	ERROR CHECKING

Errors associated with commands issued on mapped LUTYPE6.2 conversations can occur either on the command that causes the error or on a later command issued on the same conversation.

Errors cause EIBERR to be set, with an associated return code in EIBERRCD. Some errors cause a CICS condition to be raised, which you may decide to HANDLE or to check for in EIBRCODE. Some error indications can arise in a planned-for manner; for example, because the other transaction is designed to send ISSUE ERROR under certain conditions.

In general, you are advised to check EIBERR after every command, unless you are prepared to allow your transactions to ABEND when errors occur.

Errors Associated with Connecting and Conversing with the Remote Process

EIBERRCD	CICS Condition	Meaning and Notes
080F6051	TERMERR	The link and/or the user failed to pass the remote system's security checks.
084B6031	TERMERR	The specified PROCESS is not available.
084C0000	TERMERR	The specified PROCESS is not available.
10086021	TERMERR	The specified PROCESS name was not recognized.
10086031	TERMERR	PIP data was specified but the remote process does not support it.
10086032	TERMERR	The PIP data was incorrectly specified.
10086034	TERMERR	The conversation types do not match (the remote conversation partner is using unmapped commands).
10086041	TERMERR	The specified SYNC_LEVEL is not supported by the remote process.

Figure 61 (Part 1 of 2). Checking EIBERRCD

General Errors and State Indications		
EIBERRCD	CICS Condition	Meaning and Notes
08240000		A ROLLBACK command has been received, and EIBSYNRB is set. The conversation is in STATE 8.
08640000	TERMERR	An ISSUE ABEND command has been received.
08890000		An ISSUE ERROR command has been received.
A000	TERMERR	The conversation has been prematurely terminated.
A001	TERMERR	Deadlock timeout or terminal read timeout. This code is returned instead of an AKCS or AKCT abend occurring.

Figure 61 (Part 2 of 2). Checking EIBERRCD

STATE 1 MAPPED LUTYPE6.2 CONVERSATIONS		SESSION NOT ALLOCATED
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
ALLOCATE [NOQUEUE] *	SYSIDERR	1
	SYSBUSY *	1
	Otherwise (obtain conversation identifier from EIBRSRCE)	2
* If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command and do not code a HANDLE command for the SYSBUSY condition. If you want control to be returned to your program if a session is not immediately available, either specify NOQUEUE on the ALLOCATE command and test EIBRCODE for SYSBUSY (X'D3'), or code a HANDLE CONDITION SYSBUSY command.		

Figure 62. State 1 – session not allocated

STATE 2 MAPPED LUTYPE6.2 CONVERSATIONS		SESSION ALLOCATED
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
CONNECT PROCESS	TERMERR *	3
FREE	-	1
* The failure of a CONNECT PROCESS command (caused by such things as the unavailability of the remote process or mismatched sync levels) is usually indicated on a later command on the same conversation.		

Figure 63. State 2 — session allocated

STATE 3 MAPPED LUTYPE6.2 CONVERSATIONS		SEND STATE
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
SEND	–	3
SEND INVITE	–	4
SEND INVITE WAIT	–	5
SEND LAST	–	9
SEND LAST WAIT	–	10
SEND CONFIRM (SYNCLEVEL 1 or 2 only)	See "Checking the Response to SEND CONFIRM" earlier in this chapter. New states assume that EIBERR is not set.	3
SEND INVITE CONFIRM (SYNCLEVEL 1 or 2 only)		5
SEND LAST CONFIRM (SYNCLEVEL 1 or 2 only)		10
CONVERSE Equivalent to: SEND INVITE WAIT RECEIVE	Go to the STATE 5 table and make the tests shown for the RECEIVE command	–
RECEIVE (INVITE is sent by CICS)	Go to the STATE 5 table and make the tests shown for the RECEIVE command	–
ISSUE PREPARE (SYNCLEVEL 2 only) Note: If a negative response is received, EIBERR and EIBERRCD will also be set	EIBSYNRB	8
	EIBFREE	10
	Otherwise	3
SYNCPOINT (SYNCLEVEL 2 only)	EIBRLDBK (or ROLLEDBACK condition)	5
	Otherwise (transaction will ABEND if SYNCPOINT fails)	3
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3
WAIT CONVID	–	3

Figure 64 (Part 1 of 2). State 3 – send state

ISSUE ERROR	EIBRECV	5
	Otherwise	3
ISSUE ABEND	–	10
FREE Equivalent to: SEND LAST WAIT FREE	–	1

Figure 64 (Part 2 of 2). State 3 – send state

STATE 4 MAPPED LUTYPE6.2 CONVERSATIONS	RECEIVE PENDING AFTER INVITE	
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
WAIT	–	5
RECEIVE Equivalent to: WAIT RECEIVE	Go to the STATE 5 table and make the tests shown for the RECEIVE command	–
ISSUE ERROR	EIBFREE	10
	Otherwise	3
ISSUE ABEND	–	10
SYNCPOINT (SYNCLEVEL 2 only)	EIBRLDBK (or ROLLEDBACK condition)	5
	Otherwise (transaction will ABEND if SYNCPOINT fails)	5
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3

Figure 65. State 4 – receive pending after INVITE

STATE 5 MAPPED LUTYPE6.2 CONVERSATIONS		RECEIVE STATE
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
RECEIVE [NOTRUNCATE] *	EIBCOMPL *	-
	EIBCONF (SYNCLEVEL 1 or 2 only)	6
	EIBSYNC (SYNCLEVEL 2 only)	7
	EIBSYNRB (SYNCLEVEL 2 only)	8
	EIBFREE	10
	EIBRECV **	5
	Otherwise	3
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3
ISSUE ERROR	EIBFREE	10
	Otherwise	3
ISSUE ABEND	-	10

* If NOTRUNCATE is specified, a zero value in EIBCOMPL indicates that the data passed to the application by CICS is incomplete (because, for example, the data-area specified in the RECEIVE command is too small). CICS will save the remaining data for retrieval by subsequent RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back. If the NOTRUNCATE option is not specified, overlength data is indicated by the LENGERR condition, and the remaining data is discarded by CICS.

** If a receive command completes with 'EIBRECV' and 'EIBNODAT', it is implied that another RECEIVE must be issued in order to receive additional application-level data.

Figure 66. State 5 — receive state

STATE 6 MAPPED LUTYPE6.2 CONVERSATIONS		RECEIVER ISSUE CONFIRMATION
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
ISSUE CONFIRMATION	EIBFREE (saved value)	10
	EIBRECV (saved value)	5
	Otherwise	3
ISSUE ERROR	EIBFREE (saved value)	3
	Otherwise	3
ISSUE ABEND	–	10

Figure 67. State 6 – receiver issue confirmation

STATE 7 MAPPED LUTYPE6.2 CONVERSATIONS		RECEIVER TAKE SYNCPOINT
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
SYNCPOINT	EIBFREE (saved value)	10
	EIBRECV (saved value)	5
	Otherwise	3
SYNCPOINT ROLLBACK	-	3
ISSUE ERROR (This will cause the other transaction to abend if it issued SYNCPOINT, but not if it issued ISSUE PREPARE.)	(Now issue SEND INVITE WAIT followed by RECEIVE)	(3) then 5
ISSUE ABEND	–	10

Figure 68. State 7 – receiver take syncpoint

STATE 8 MAPPED LUTYPE6.2 CONVERSATIONS		RECEIVER TAKE ROLLBACK
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
SYNCPOINT ROLLBACK	–	5

Figure 69. State 8 – receiver take rollback

STATE 9 MAPPED LUTYPE6.2 CONVERSATIONS		FREE PENDING AFTER SEND LAST
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
WAIT	–	10
FREE	–	1
SEND CONFIRM (no data) (SYNCLEVEL 1 or 2 only)	–	10
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	–	10
SYNCPOINT (SYNCLEVEL 2 only)	–	10
ISSUE ABEND	–	10

Figure 70. State 9 – free pending after SEND LAST

STATE 10 MAPPED LUTYPE6.2 CONVERSATIONS		FREE SESSION
Commands You Can Issue	What To Test (For EIBERRCD tests, see above)	New State
FREE	–	1

Figure 71. State 10 – free session

Migration of LUTYPE6.1 applications to LUTYPE6.2 links

If your installation is changing its CICS-to-CICS ISC links from LUTYPE6.1 to LUTYPE6.2, you may want to redesign some of your existing ISC applications to take advantage of LUTYPE6.2 function. Alternatively, you can continue to run your existing applications in "migration" mode.

In migration mode, the front-end and back-end transactions use LUTYPE6.1 commands just as if the session was an LUTYPE6.1 session. CICS takes data from the transaction in the normal way, and formats it as an LUTYPE6.2 data stream for transmission over the link. At the receiving side, CICS analyses the LUTYPE6.2 data stream and presents the LUTYPE6.1 data and function management headers to the receiving transaction.

In general, you will not have to modify existing CICS-to-CICS LUTYPE6.2 applications to enable them to run in migration mode on LUTYPE6.2 links. A notable exception is the use of the ALLOCATE SESSION command. If your installation previously had individually-defined LUTYPE6.1 sessions, and your application used the ALLOCATE SESSION command to acquire a specific session, you must change this command to ALLOCATE SYSID. The ISSUE SIGNAL command is available for both LUTYPE6.1 and LUTYPE6.2, but the WAIT SIGNAL command is valid only for LUTYPE6.1.

Figure 72 on page 218 compares the commands that you can use for:

- LUTYPE6.1 applications on LUTYPE6.1 links
- LUTYPE6.1 applications on LUTYPE6.2 links (migration mode)
- LUTYPE6.2 applications on LUTYPE6.2 links.

As this figure shows, migration mode allows you to start adding new functions to an application (for example, using ISSUE ERROR or ISSUE ABEND) without converting it entirely to LUTYPE6.2. You can also implement different levels of synchronization if you modify the application to use the CONNECT PROCESS command. The migration of an application towards the "pure" LUTYPE6.2 level can thus be made stepwise, and halted at any time.

To facilitate migration, the keywords SESSION and CONVID can be used interchangeably.

If a migration-mode transaction abends, the data flows will be the architected LUTYPE6.2 flows. The effect on the connected transaction depends on the point at which the abend occurs and is often different from that which you would expect if the connection were native LUTYPE6.1.

Operation	Command	LUTYPE6.1	Migration	LUTYPE6.2
Obtain use of a session	ALLOCATE SESSION	yes	no	no
	ALLOCATE SYSID	yes	yes	yes
Build an LU6.1 attach FMH	BUILD ATTACHID	yes	yes	no
Start a remote transaction	SEND	yes(1)	yes(4)	no
	SEND ATTACHID	yes(2)	yes(5)	no
	SEND FMH	yes(3)	yes(6)	no
	CONNECT PROCESS	no	yes(7)	yes(7)
Retrieve information about how the transaction was initiated	EXTRACT ATTACH	yes	yes	no
	EXTRACT PROCESS	no	yes	yes
Send data	SEND	yes	yes	yes
Send further LU6.1 FMHs	SEND ATTACHID	yes	yes	no
	SEND FMH	yes	yes	no
Receive LU 6.1 FMHs	EXTRACT ATTACH	yes	yes	no
Receive data	RECEIVE	yes	yes	yes
Send and receive data	CONVERSE	yes	yes	yes
Program Error	ISSUE ERROR	no	yes	yes
Abend conversation	ISSUE ABEND	no	yes	yes
Request change of direction	ISSUE SIGNAL	yes	yes	yes
Await change of direction	WAIT SIGNAL	yes	no	no

Figure 72 (Part 1 of 2). Migration of LUTYPE6.1 programs to LUTYPE6.2 links

Synchronize	Level 0	no	yes(8)	yes
	Level 1			
	SEND CONFIRM	no	yes(8)	yes
	ISSUE CONFIRMATION	no	yes	yes
	Level 2			
	SEND CONFIRM	no	yes(8)	yes
	ISSUE CONFIRMATION	no	yes	yes
	SYNCPOINT	yes	yes	yes
	SYNCPOINT ROLLBACK	no	yes	yes

Figure 72 (Part 2 of 2). Migration of LUTYPE6.1 programs to LUTYPE6.2 links

Notes:

1. The CICS transaction identifier is included in the first four bytes of the data. No attach FMH generated.
2. An LUTYPE6.1 attach FMH is generated.
3. An LUTYPE6.1 FMH provided by the application program is sent.
4. An LUTYPE6.2 attach FMH is generated, but with no TPN (TPNL=0). The CICS transaction identifier is included in the first four bytes of the data.
5. An LUTYPE6.2 attach FMH and an LUTYPE6.1 attach FMH are generated.
6. An LUTYPE6.2 attach FMH and an LUTYPE6.1 FMH (provided by the application program) are sent.
7. An LUTYPE6.2 attach FMH is generated.
8. Synchronization levels 0 and 1 can be used if CONNECT PROCESS has been used to define the SYNCLEVEL in operation. If CONNECT PROCESS has not been used, SYNCLEVEL 2 is assumed.

User exits

Because LUTYPE6.2 uses different modules from LUTYPE6.1, the XZCIN and XZCOUT user exits are not taken for LUTYPE6.2 sessions. Any programs making use of these exits on LUTYPE6.1 will need consideration.

LUTYPE6.2 release considerations

The physical action of the LUTYPE6.2 commands is to generate LUTYPE6.2 architectured flows. However, there is a degree of latitude within the architecture as to exactly when these flows occur. Therefore, upgrading either of the partners in an LUTYPE6.2 conversation can affect the physical flows. This means that application programs could function differently, because they are responding to slightly different flows.

If you are conversing with an LUTYPE6.2 application that uses the SAA Common Programming Interface for Communications (CPI-C), you might get different flows from an equivalent CICS application program. Similarly, different CICS releases may do slightly different things under what seems to be the same circumstance.

The variations will probably show up in a differing number of EXEC CICS RECEIVEs that have to be done to collect both data and indicators. However, if you do not make any assumptions as to the conversation state of your LUTYPE6.2 application program and always test the EIB flags, as described in this chapter, you will automatically cater for these variations.

Chapter 4.6. CICS applications for logical unit type 6.2 unmapped conversations

This chapter contains details of the CICS low-level, "unmapped", interface to LUTYPE6.2 sessions (which are also known as ***basic*** LUTYPE6.2 conversations).

To use this interface, you need to understand the format of LUTYPE6.2 data entities, which contain both length and identifier fields, and to build the appropriate data in your application programs.

If you are new to CICS LUTYPE6.2 application programming, you may find it helpful to read "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171 before starting on this chapter. You should also have the *SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269 and the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084 available for reference. The mapping between LUTYPE6.2 verbs and CICS commands is described in Appendix C, "CICS mapping to the LUTYPE6.2 architecture" on page 413.

CICS applications that use the unmapped interface can be written only in assembler language.

If the device with which you are conversing supports LUTYPE6.2 mapped conversations, it is much simpler to write an application program using the mapped protocols than the unmapped.

CICS commands for unmapped LUTYPE6.2 conversations

The commands that can be used to establish and hold unmapped LUTYPE6.2 conversations are listed below. In these commands, GDS stands for Generalized Data Stream, indicating that you are using the LUTYPE6.2 architectured datastream with these commands.

- EXEC CICS GDS ALLOCATE – used to acquire a session to the remote system.
- EXEC CICS GDS CONNECT PROCESS – used to initiate a conversation with a named process on the remote system.
- EXEC CICS GDS EXTRACT PROCESS – used to access session-related information (for example, the requested synchronization level) in the LUTYPE6.2 attach header that caused the application to be initiated.
- EXEC CICS GDS ASSIGN – used to obtain the conversation identifier and the system identifier of the application program's principal facility.
- EXEC CICS GDS SEND and EXEC CICS GDS RECEIVE – used to send or receive data on the conversation. There is no GDS equivalent of the mapped CONVERSE command.
- EXEC CICS GDS WAIT – used to ensure that the transaction has transmitted any accumulated data or data flow control indicators before it continues with further processing.

- EXEC CICS GDS SEND CONFIRM and EXEC CICS GDS ISSUE CONFIRMATION — used to exchange private synchronization requests when SL(1) or SL(2) is being used.
- EXEC CICS GDS ISSUE PREPARE — used to issue the first flow of a syncpoint exchange (SL(2) only) under direct control of the transaction.
- EXEC CICS GDS ISSUE ERROR — used to inform the conversation partner that a program-detected error has occurred.
- EXEC CICS GDS ISSUE SIGNAL — used by the receiving transaction to request a change-direction from the sending transaction.
- EXEC CICS GDS ISSUE ABEND — used to inform the conversation partner that it is necessary to abend the conversation.
- EXEC CICS GDS FREE — used by a transaction to relinquish its use of a session. The session is then available for use by other transactions. If a session is not freed explicitly, it is freed by CICS when the transaction terminates.

The full syntax of these commands is given in "EXEC CICS GDS commands" on page 225.

The EXEC CICS SYNCPOINT and EXEC CICS SYNCPOINT ROLLBACK commands can also be used. Only conversations that can support SL(2) syncpointing are involved in CICS syncpointing activity.

Session data and return and error codes

The CICS exec interface block (EIB) is not affected by unmapped conversations on LUTYPE6.2 sessions, and no CICS conditions are raised when EXEC CICS GDS commands are executed. Instead, you must provide data areas in your application to receive return codes and session status information.

The data areas required are:

- A 6-byte area to receive RETCODE information
- A 24-byte area to receive CONVDATA information.

You can give these areas any identifiers you like. They must be named explicitly in most EXEC CICS GDS commands.

The RETCODE area is used to detect any errors that occur when an EXEC CICS GDS command is executed. These errors correspond to CICS exceptional conditions, such as NOTALLOC, that can be raised when EXEC CICS commands are executed.

The CONVDATA area contains information on the state of the conversation after certain EXEC GDS commands have been successfully executed, such as whether the transaction is in send or receive state on the conversation. Its structure corresponds to a subset of the CICS EIB.

The application program is responsible for managing this area — we recommend that you save it after each GDS command in a similar fashion to the way the EIB should be saved for mapped conversations. However, because you can specify a different CONVDATA area for each GDS command, you may choose to manage status information in a different fashion from that for mapped conversations.

RETCODE values

Return codes are returned in the 6-byte area that you name in the RETCODE option of the EXEC CICS GDS commands. Figure 73 shows the possible hexadecimal values for the first three bytes of RETCODE; the second three bytes are normally zeros.

```
00 .. ..   NORMAL RETURN CODE

01 .. ..   SYSIDERR ERROR

01 04 ..   ALLOCATE FAILURE
01 04 04       SYSBUSY (NO BOUND CONTENTION WINNER AVAILABLE)
01 04 08       MODENAME NOT KNOWN ON THIS SYSTEM
01 04 0C       ATTEMPT TO USE RESERVED MODENAME SNASVCMG
01 04 14       AVAILABLE COUNT ZERO FOR THIS MODEGROUP

01 08 ..     SYSID IS OUT OF SERVICE
01 08 00       LOCAL QUEUEING WAS NOT ATTEMPTED
01 08 04       LOCAL QUEUEING DID NOT SUCCEED

01 0C ..     SYSID IS NOT KNOWN IN TCT
01 0C 00       SYSID NAME IS NOT KNOWN
01 0C 04       SYSID NAME IS NOT THAT OF A TCTSE
01 0C 08       SYSID.MODENAME IS NOT KNOWN
01 0C 0C       SYSID.PROFILE IS NOT KNOWN

03 .. ..   INVREQ ERROR
03 00 ..     SESSION IS NOT DEFINED AS LUTYPE6.2
03 04 ..     CONVERSATION LEVEL IS WRONG
03 08 ..     STATE ERROR
03 0C ..     SYNCLEVEL CAN NOT BE SUPPORTED
03 10 ..     LLCOUNT ERROR
03 14 ..     INVALID REQUEST
03 18 ..     TPN SEND CHECK FAILED

04 .. ..   NOTALLOC ERROR

05 .. ..   LENGERR ERROR
```

Figure 73. RETCODE values

Testing RETCODE values

The error conditions that can occur when you issue a command for an LUTYPE6.2 unmapped conversation are shown under "EXEC CICS GDS commands" on page 225. These errors do not raise the equivalent CICS exceptional conditions in your application program, and you cannot write CICS HANDLE commands to "catch" them. Instead, you must test the value returned in your RETCODE data area after issuing each command. Possible values are shown in the previous section.

CONVDATA fields

The fields required in the CONVDATA area are:

```
CDBCOMPL  DS   C      X'FF' = data complete
CDBSYNC   DS   C      X'FF' = SYNCPOINT required
CDBFREE   DS   C      X'FF' = FREE required
CDBRECV   DS   C      X'FF' = RECEIVE required
CDBSIG    DS   C      X'FF' = SIGNAL received
CDBCONF   DS   C      X'FF' = CONFIRM received
CDBERR    DS   C      X'FF' = ERROR received
CDBERRCD  DS   CL4    error code (when CDBERR set)
CDBSYNRB  DS   C      X'FF' = SYNCPOINT ROLLBACK required
CDBRSVD   DS   CL12   reserved
```

These definitions are provided in copybook DFHCDBLK. The copybook does not contain a DSECT statement. This enables you to provide your own DSECT statements and provides the flexibility to enable you to manage more than one conversation at the same time.

Testing CONVDATA values

CONVDATA contains information on the state of the conversation and on the various indicators that are transmitted by the remote process (possibly a CICS transaction) with which your CICS transaction is conversing.

In general, CONVDATA fields are set only as a result of an input operation (such as RECEIVE), and reflect the state of the conversation after the command has been executed. The fields are not usually modified as a result of an output operation (such as SEND). For example, if the remote process issues a SEND INVITE command (or its non-CICS equivalent), execution of your RECEIVE command will ensure that CDBRECV is set to X'00', indicating that you are now expected to send. If you now issue a SEND INVITE command, you know that you can now receive; CDBRECV is not updated until you have issued the RECEIVE command.

If your application is one that expects to receive an SNA SIGNAL command, it should also check the CDBSIG field.

Details of the various conversation "states", the commands that you can issue, and the required CONVDATA tests are given in "Command sequences on LUTYPE6.2 unmapped conversations" on page 233.

Error code values

Most error conditions that occur on an unmapped LUTYPE6.2 conversation cause CDBERR to be set and an error code to be placed in CDBERRCD. Your transaction can thus remain in control in many circumstances that would cause a transaction holding a mapped conversation to abend.

It is normally necessary to test CDBERR after every EXEC CICS GDS command. Error code values are listed in Figure 75 on page 236.

EXEC CICS GDS commands

The following sections show the full syntax of the EXEC CICS GDS commands. The way in which these commands relate to LUTYPE6.2 verbs is explained in Appendix C, "CICS mapping to the LUTYPE6.2 architecture" on page 413.

```
EXEC CICS GDS ALLOCATE
SYSID(name)
CONVID(data-area)
RETCODE(data-area)
[MODENAME(name)]
[NOQUEUE]

Errors: SYSIDERR, SYSBUSY
```

```
EXEC CICS GDS ASSIGN
[PRINCONVID(data-area)]
[PRINSYSID(data-area)]
RETCODE(data-area)

Errors: INVREQ
```

```
EXEC CICS GDS CONNECT PROCESS
PROCNAME(name)
PROCLENGTH(data-value)
SYNCLEVEL(data-value)
CONVID(name)
CONVDATA(data-area)
[PIPLIST(data-area)
 PIPLENGTH(data-value)]
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS EXTRACT PROCESS
[PROCNAME(data-area)
 PROCLENGTH(data-area)]
[SYNCLEVEL(data-area)]
CONVID(name)
[PIPLIST(pointer-ref)
 PIPLENGTH(data-area)]
RETCODE(data-area)

Errors: INVREQ
```

```
EXEC CICS GDS FREE
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS ISSUE CONFIRMATION
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS ISSUE ERROR
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS ISSUE PREPARE
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS ISSUE SIGNAL
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS RECEIVE
{INTO(data-area)|SET(pointer-ref)}
FLENGTH(data-area)
MAXFLENGTH(data-value)
{BUFFER | LLID}
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, LENGERR, NOTALLOC
```

```
EXEC CICS GDS SEND
[FROM(data-area)
 FLENGTH(data-value)]
[INVITE|LAST]
[CONFIRM|WAIT]
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, LENGERR, NOTALLOC
```

Figure 74 (Part 1 of 2). EXEC CICS GDS commands

```
EXEC CICS GDS ISSUE ABEND
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

```
EXEC CICS GDS WAIT
CONVID(name)
CONVDATA(data-area)
RETCODE(data-area)

Errors: INVREQ, NOTALLOC
```

Figure 74 (Part 2 of 2). EXEC CICS GDS commands

Command options

In general, the arguments (such as **data-value**) can be replaced by absolute or relocatable assembler expressions. For full details of the argument allowed in assembler language programs, refer to the *CICS/MVS Application Programmer's Reference* manual.

BUFFER
specifies that the length of the data passed to the application program in response to the RECEIVE command is to be the length specified in the MAXFLENGTH option, irrespective of structured field boundaries. Control is returned to the application program when this length has been received, or when a synchronization request, change-direction, or end-bracket is received.

CONFIRM
allows an application working at SL(1) or SL(2) to synchronize its processing with that of a process in a remote system. The actions taken to synchronize processing are wholly defined by the application programs involved. The CONFIRM option causes RQD2 to be added to the data already sent and forces a WAIT. On receipt of the indicator, the remote process will take agreed actions and then send a response. When the WAIT completes, CDBERR will be set to X'00' if the appropriate response has been received.

CONVDATA(data-area)
specifies the application data area into which conversation related information is to be moved. A description of the format of the data area is given under "CONVDATA fields" on page 224. The data-area must be 24 bytes in length.

CONVID(data-area)
specifies the application data area that is to contain the token returned by an ALLOCATE command to identify the allocated conversation. The data-area must be 4 characters in length.

CONVID(name)
identifies the conversation to which the command relates. The name, which is 4 characters in length, identifies either the token returned by a previously executed GDS ALLOCATE command or the token representing the principal session (returned by a previously executed GDS ASSIGN command).

FLENGTH(data-area)
specifies a data area that is set to the length of the data made available to the application program. The data area must be fullword binary.

FLENGTH(data-value)
specifies the length (as a fullword binary value) of the data specified in the FROM option. The length must not exceed 32 767.

INTO(data-area)
specifies the application target data area into which data is to be received from the application program connected to the other end of the current conversation.

INVITE
allows an application program to add CD to data already sent to a process in a connected LUTYPE6.2 system. CD is not transmitted by CICS until the subsequent execution of a WAIT or a SYNCPOINT command, unless CONFIRM or WAIT are also coded on the SEND INVITE command.

LAST
allows an application program to add CEB to data already sent to a process in a connected LUTYPE6.2 system. CEB is not transmitted by CICS until the subsequent execution of a WAIT or a SYNCPOINT command, unless CONFIRM or WAIT are also coded on the SEND LAST command. Note that the LUTYPE6.2 architecture states that, should one of these commands fail because of a conversation related error, the conversation will remain in bracket. In such a case, the application program should execute a RECEIVE command. However, SEND LAST WAIT (with no data) always causes the conversation to be deallocated.

LLID
specifies that the delimiter to be used by CICS to terminate the passing of data to the application program is the end of a GDS structured field.

MAXFLENGTH(data-value)
specifies, as a fullword binary value, either the length of the target data area specified in the INTO option or the maximum length of data to be addressed by the pointer reference specified in the SET option. The length must not exceed 32 767 bytes.

MODENAME(name)
specifies the modename group from which the session is to be acquired. If MODENAME is not specified CICS will select a modename from the modenames defined for the system.

NOQUEUE
specifies that the request to allocate a session is not to be queued when a suitable LUTYPE6.2 session can not be acquired immediately. A session will only be acquired immediately if a session is already available as a contention winner.

If NOQUEUE is specified, control will be returned to the program whether or not a session has been acquired. SYSBUSY is set in RETCODE if the allocation is unsuccessful.

If the NOQUEUE option is not used, a delay may occur before control is passed back to the application program. A delay can occur for any of the following reasons:

- All sessions for the specified sysid and modename are in use

- The CICS allocation algorithm has selected a session that is not currently bound (in which case CICS has to bind)
- The CICS allocation algorithm has selected a contention loser (in which case CICS has to bid).

In the event of a delay, the program will wait until the session has been acquired.

PIPLENGTH(data-area)
specifies a halfword binary data area that is to receive the length of the PIPLIST received by an EXTRACT PROCESS command.

PIPLENGTH(data-value)
specifies the total length of the PIPLIST specified on a CONNECT PROCESS command.

PIPLIST(data-area)
specifies the PIP data that is to be sent to the remote process. See "Process initialization parameter data" on page 17.

PIPLIST(pointer-ref)
specifies the pointer reference that is to be set to the address of the PIPLIST received by an EXTRACT process. A zero setting indicates that no PIPLIST was received.

PRINCONVID(data-area)
specifies a 4-byte data area in which the conversation token (CONVID) of the principal facility is to be returned.

PRINSYSID(data-area)
specifies a 4-byte data area in which the SYSID of the principal facility is to be returned.

PROCLENGTH(data-area)
specifies a halfword binary data area that is set to the actual length of the process name.

PROCLENGTH(data-value)
specifies the length (as a halfword binary value) of the target process name. The value must be greater than zero and not greater than 32.

PROCNAME(data-area)
specifies the application target data area into which the process name, specified in the LUTYPE6.2 attach function management header, is to be moved. The data area must be 32 bytes long. The process name will be moved into this area and the area will be padded with blanks, if necessary.

PROCNAME(name)
specifies the target process (in CICS terms, the transaction) that is to be connected to the other end of the current conversation. Four bytes are sufficient to identify a CICS transaction. The LUTYPE6.2 architecture allows a range of 1 to 64 bytes but leaves each product free to set its own maximum. CICS complies by allowing 32 bytes, but this need only concern you if you are linked to a non-CICS system which demands longer transaction identifiers. To attach a transaction in the remote system, you need only supply the operands as set out above. If you expect a remote system to

send attach requests with names longer than four bytes, you have a choice. Because CICS always interprets the first four bytes, you can make sure that these always represent a transaction identifier within your system. Alternatively, you can examine the full identifier by coding the user exit XZCATT as described in the *CICS/MVS Customization Guide*.

For unmapped conversations, each character of a transaction identifier must be chosen from the EBCDIC range:

a through z
A through Z
0 through 9
. $ # @

If the remote system expects ASCII characters, you are restricted to those which share a binary value with one of the above characters. For full details, refer to the "AE" character set defined in the SNA publication *Transaction Programmer's Reference Manual for LU Type 6.2*.

RETCODE(data-area)
specifies the application data area into which return code information is to be moved. The data-area must be 6 bytes in length. (See "RETCODE values" on page 223.)

SET(pointer-ref)
specifies the pointer reference that is to be set to the address of data received from the application program connected to the other end of the current conversation.

SYNCLEVEL(data-area)
specifies a data area that is set to indicate the syncpoint level in effect for the current conversation. The values that may be returned are:-

0 none

1 confirm

2 syncpoint.

The data-area must be halfword binary.

SYNCLEVEL(data-value)
specifies (as a halfword binary value) the syncpoint level desired for the current conversation. The values that can be specified are:-

0 none

1 confirm

2 syncpoint.

SYSID(name)
specifies the remote system to which an LUTYPE6.2 session is to be allocated. The name, which is up to 4 characters in length, identifies an entry (defined as an LUTYPE6.2 system) in the CICS Terminal Control table.

WAIT

specifies that processing of the command must be completed before any subsequent processing is attempted. If the WAIT option is not specified, control is returned to the application program once processing of the command has started.

Note: If the WAIT option is not used, data from successive SEND commands, together with any indicators, is accumulated by CICS in an internal buffer. If the buffer becomes full, the accumulated data is transmitted to the remote system, but the accumulated indicators are not. Transmission of the accumulated data plus the indicators is forced by the WAIT or CONFIRM options of the SEND command, or by an EXEC CICS GDS WAIT command.

Exceptional conditions

This list lists the exceptional conditions that can arise as a result of executing an EXEC CICS GDS command. These conditions must be tested for in RETCODE. EXEC CICS HANDLE branches will not be taken, and there are no default actions.

INVREQ

occurs if the operation requested on the conversation is not valid. For possible reasons, see "RETCODE values" on page 223.

LENGERR

For a SEND or RECEIVE command, the LENGERR error code is returned if:

1. The value specified in the FLENGTH option is less than zero
2. The value specified in the FLENGTH option exceeds a CICS implementation-defined limit. CICS returns the implementation limit, which is currently 32 767, in bytes 2-5 of RETCODE.

For a CONNECT PROCESS or EXTRACT PROCESS command the LENGERR error code is returned if:

1. The application and CICS supplied values for the attach function management header together cause the length of the function management header to exceed 255.
2. The value specified in the PROCLENGTH option is less than or equal to zero.
3. The value specified by the PIPLENGTH option is less than zero or exceeds the CICS implementation limit (currently 32763).
4. The sum of the length elements in the PIPLIST is not equal to the value specified in the PIPLENGTH option.
5. Any length element in the PIPLIST has a value less than four.

NOTALLOC

occurs if the conversation named in the CONVID option is not allocated to the application program.

SYSBUSY

occurs on an ALLOCATE command when the specified system is busy.

SYSIDERR

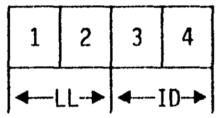
occurs on an ALLOCATE command when the specified SYSID is in error. For possible reasons, see "RETCODE values" on page 223.

Comparisons between LUTYPE6.2 mapped and unmapped conversations

At the lowest level, an LUTYPE6.2 unmapped conversation performs flows that are equivalent to those of an LUTYPE6.2 mapped conversation. The concepts involved in unmapped conversations are similar to those for mapped conversations. See "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171 for an overview of LUTYPE6.2 mapped concepts if you are unsure about LUTYPE6.2 unmapped operation. The rest of this chapter assumes that you are familiar with these concepts. See the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084 for a full description of LUTYPE6.2 unmapped programming.

The unmapped conversation

The datastream used for both mapped and unmapped LUTYPE6.2 conversations is the SNA generalized datastream (GDS). In GDS, each chunk of information is preceded by a 4-byte header field with the following format:



The first 2 bytes, LL, specify the overall length of the information (including the length field). The second 2 bytes, ID, specify the type of data. For example, the ID for user data is X'12FF'. These datastreams are defined in the *SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269.

For an unmapped conversation, you will normally be responsible for building GDS structured fields with the appropriate length (LL) and identifier (ID) fields. You must always ensure that the value in the LL field matches the actual length of the structured field. Note that the LL value must not be zero or one.

You need not observe structured field boundaries in the output data area named in an EXEC CICS GDS SEND command; you can send several structured fields, or part of a structured field. You must, however, be on a structured field boundary when you issue a SEND CONFIRM, SEND LAST, SEND INVITE, or SYNCPOINT command. This restriction does not apply to ISSUE ERROR or ISSUE ABEND; the error code in the receiving process will indicate if a structured field was truncated.

Data from successive SEND commands, together with any indicators, is accumulated by CICS in an internal buffer. If the buffer becomes full, the accumulated data is transmitted to the remote system, but the accumulated indicators are not. Transmission of the accumulated data plus the indicators is forced by the WAIT or CONFIRM options of the SEND command, or by the WAIT, ISSUE ERROR, ISSUE ABEND, or SYNCPOINT commands.

For unmapped conversations, you are required to exercise full control over the state of the conversation. CICS does not detect implicit state changes, and therefore cannot supply the implied indicators.

For example, in a mapped conversation, the command sequence

```
EXEC CICS SEND
EXEC CICS RECEIVE
```

is valid because CICS supplies the implied INVITE and WAIT option. In an unmapped conversation, you must specify

```
EXEC CICS GDS SEND INVITE WAIT
EXEC CICS GDS RECEIVE
```

explicitly, otherwise a state error will occur. You can avoid errors of this kind by following the rules given in the state tables, Figure 75 on page 236 through Figure 85 on page 243.

The LLID and BUFFER options of the RECEIVE command enable you to specify how data is to be received by your application program.

A RECEIVE command with the LLID option will recover at most a single structured field. If the length exceeds the value specified in the MAXFLENGTH option, two or more RECEIVE commands are required to recover the whole field. CDBCOMPL is set to X'FF' to indicate that the end of the structured field has been received.

A RECEIVE command with the BUFFER option will recover the length of data specified in the MAXFLENGTH option, ignoring structured field boundaries. Control is not returned to the application program until this length of data, or until an indicator such as change-direction that signifies the end of the incoming data, has been received. CDBCOMPL is set to X'FF' after every RECEIVE command with the BUFFER option.

If you issue an ISSUE ERROR, ISSUE ABEND, or SYNCPOINT ROLLBACK command while your transaction is in receive state, CICS will purge all incoming data until a change-direction, syncpoint-request, or end-bracket indicator is received. If end-bracket is received, no indication is sent to the remote process, and CDBFREE is set in your transaction, indicating that you must free the conversation. Otherwise, the appropriate indication is sent.

If your transaction is conversing with a remote CICS transaction, and you use ISSUE ERROR to respond negatively to syncpoint request (CDBSYNC set), the remote transaction will be abended. The remote transaction should be designed to back-out under these circumstances. You should continue by issuing a SEND INVITE WAIT command followed by a RECEIVE command, and then test the CDB. CDBSYNRB will be set, indicating that you must issue a SYNCPOINT ROLLBACK command to back-out as well. This sequence is shown in the state tables, Figure 75 on page 236 through Figure 85 on page 243. Note that the CDBSYNRB test is required even if the remote transaction does not issue any explicit syncpoint rollback requests.

If a transaction receives ISSUE ERROR while CICS syncpointing (SL(2) synchronization) is taking place on the session, the transaction is abended.

If the remote transaction is designed to use ISSUE ERROR while it is in receive state, you can avoid CICS syncpoint failures caused by an incoming error indication in the following way:

1. Before issuing the CICS SYNCPOINT command, issue SEND INVITE WAIT followed by RECEIVE. This will flush any deferred data and put your transaction in receive state.

   The remote transaction must be designed to recognize this pre-syncpointing flow and (unless it has issued ISSUE ERROR) to return SEND INVITE WAIT to put the conversation in the correct state for syncpointing.
2. In your transaction, test CDBERR and, if it is set, CDBERRCD, to determine whether the remote transaction has issued ISSUE ERROR. If it has, take your application-defined action for this condition. If it has not, test the CDB to ensure that the transaction is in the correct state, and if all is well issue the SYNCPOINT command.

Command sequences on LUTYPE6.2 unmapped conversations

The command sequences that you can use on unmapped LUTYPE6.2 conversations are governed by protocols designed to ensure that commands are not issued in inappropriate circumstances.

The protocols are based on the concept of a number of states. These states apply only to the particular conversation, not to your application program as a whole. In each state, there are a number of commands that might most reasonably be issued. After the command has been issued, fields in CONVDATA or RETCODE must be tested in the order shown in the state diagrams, Figure 76 on page 238 through Figure 85 on page 243, to check on the current requirements of the conversation. The results of these tests, together with the command that has been issued, may cause a transition to another state, in which another set of commands becomes appropriate.

The states that are defined for the purposes of this section are:

- State 1 — session not allocated
- State 2 — session allocated
- State 3 — send state
- State 4 — receive pending after INVITE
- State 5 — receive state
- State 6 — receiver issue confirmation
- State 7 — receiver take syncpoint
- State 8 — receiver take rollback
- State 9 — free pending after SEND LAST
- State 10 — free session.

Initial states

The front-end transaction in a conversation will initially be in state 1 — session not allocated — and must issue an ALLOCATE command to acquire a session.

A back-end transaction, initiated by an attach FMH, is initially in state 5 — receive state. Because the CONVID option is mandatory on EXEC CICS GDS commands, even for the principal facility, you must issue an EXEC CICS GDS ASSIGN command to obtain the CONVID immediately. After this, you should issue an EXEC CICS GDS RECEIVE to set up the back-end conversation correctly, and so obtain any indicators and data that were sent on the initial flow.

State diagrams

The following diagrams are intended to enable you to construct valid command sequences. Each diagram relates to one particular state, as previously defined, and shows the commands that you might reasonably issue and the tests that you should make after issuing the command. Where more than one test is shown, they must be made in the order indicated.

The combination of the command issued and a particular positive test result lead to a resultant state, shown in the final column.

Note that a GDS ISSUE SIGNAL command is always valid within an allocated LUTYPE6.2 session.

See "Using the state diagrams" on page 205 for an example of how to use state diagrams.

Testing CDBERR and CDBSYNRB

The state diagrams in this section do not show CDBERR tests. It is normally necessary to test CDBERR after every EXEC CICS GDS command. Some of the most common CDBERRCD values are shown in Figure 75 on page 236.

You will have to include tests for "issue error received" and "issue abend received" if your overall application design makes use of these facilities. For SL(1) conversations, this is hardly avoidable, because ISSUE ERROR is the usual negative response to SEND CONFIRM. You can also test CDBERRCD for "syncpoint rollback received", although this is also indicated in CDBSYNRB.

You can receive issue error, issue abend, or syncpoint rollback at any time that your transaction is in receive state, and also following any command that causes a transmission to the remote system. It is safest to test for these CDBERRCD values after every command.

If ISSUE ERROR is received, your transaction is in receive state, and you should issue a RECEIVE command. If ISSUE ABEND is received, the conversation has been abended, and you should free the session.

If both the sending and the receiving transaction perform ISSUE ERROR at the same time, the transaction that is in send state is put into receive state, and will find CDBERR set. The transaction that is in receive state is put into send state, and should issue a SEND command.

If your application design includes the use of SYNCPOINT ROLLBACK, similar considerations apply. When the transaction is required to execute a SYNCPOINT ROLLBACK command, CDBSYNRB is set as well as CDBERR, with a CDBERRCD of X'0824'.

Other CDBERRCD values can be tested for as required. You can, of course, choose simply to abend the transaction by means of an EXEC CICS ABEND command.

Other tests

Not all possible RETCODE tests are shown in the state diagrams. A RETCODE test should normally be made as a matter of course after each command, before any CONVDATA tests.

Also, if your transaction expects to receive an SNA SIGNAL command, you should test CDBSIG after every command.

ALL STATES	UNMAPPED LUTYPE6.2 CONVERSATIONS	ERROR CHECKING

Errors associated with commands issued on unmapped LUTYPE6.2 conversations can occur either on the command that causes the error or on a later command issued on the same conversation.

Errors cause CDBERR to be set, with an associated return code in CDBERRCD. Some error indications can arise in a planned-for manner; for example, because the other transaction is designed to send ISSUE ERROR under certain conditions.

In general, you are advised to check CDBERR after every command, unless you are prepared to allow your transactions to ABEND when errors occur.

Errors Associated with ALLOCATE and CONNECT PROCESS

CDBERRCD	Meaning and Notes
080F6051	The link and/or the user failed to pass the remote system's security checks.
084B6031	The specified PROCESS is not available.
084C0000	The specified PROCESS is not available.
10086021	The specified PROCESS name was not recognized.
10086031	PIP data specified but the remote process does not support it.
10086032	The PIP data was incorrectly specified.
10086034	The conversation types do not match (the remote conversation partner is using mapped commands).
10086041	The specified SYNC_LEVEL is not supported by the remote process.

Figure 75 (Part 1 of 2). Checking CDBERRCD

General Errors and State Indications	
CDBERRCD	Meaning and Notes
A000....	The conversation has been prematurely terminated. Session failure is handled by the node abnormal condition program, including invocation of any node-error program. However, the task is not normally abended, regardless of the setting of the action flags. The ABEND error action flag is overridden and control is returned to the application program.
A001....	Deadlock timeout or terminal read timeout. This code is returned instead of an AKCS or AKCT abend occurring.
08240000	A ROLLBACK command has been received, and CDBSYNRB is set. The conversation is in STATE 8.
08640000	The conversation is terminated abnormally. Either the remote process has issued ISSUE ABEND or CICS had to force ABEND.
08640001	A system logic error has been detected. No retry of the conversation should be attempted.
08890000	An ISSUE ERROR command has been received. The GDS field has not been truncated.
08890001	An ISSUE ERROR command has been received. The GDS field has been truncated.

Figure 75 (Part 2 of 2). Checking CDBERRCD

STATE 1 UNMAPPED LUTYPE6.2 CONVERSATIONS		SESSION NOT ALLOCATED
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
ALLOCATE [NOQUEUE] *	RETCODE for SYSIDERR (01 .. ..) or SYSBUSY (01 04 04) *	1
	Otherwise (conversation identifier returned in CONVID data area)	2

* If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command.

If you want control to be returned to your program if a session is not immediately available, specify NOQUEUE on the ALLOCATE command and test RETCODE for SYSBUSY.

Figure 76. State 1 — session not allocated

STATE 2 UNMAPPED LUTYPE6.2 CONVERSATIONS		SESSION ALLOCATED
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
CONNECT PROCESS	RETCODE for INVREQ, NOTALLOC, or LENGERR	2
	–	3
FREE	RETCODE for INVREQ or NOTALLOC	2
	–	1

Figure 77. State 2 — session allocated

STATE 3 UNMAPPED LUTYPE6.2 CONVERSATIONS		SEND STATE
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
SEND	–	3
SEND INVITE	–	4
SEND INVITE WAIT	–	5
SEND LAST	–	9
SEND LAST WAIT	–	10
SEND CONFIRM (SYNCLEVEL 1 or 2 only)	–	3
SEND INVITE CONFIRM (SYNCLEVEL 1 or 2 only)	–	5
SEND LAST CONFIRM (SYNCLEVEL 1 or 2 only)	–	10
ISSUE PREPARE (SYNCLEVEL 2 only) Note: If a negative response is received, CDBERR and CDBERRCD will also be set	CDBSYNRB	8
	CDBFREE	10
	Otherwise	3
SYNCPOINT (SYNCLEVEL 2 only) Note: This is not a GDS command.	EIBRLDBK (or ROLLEDBACK condition)	5
	Otherwise (transaction will ABEND if SYNCPOINT fails)	3
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3
WAIT	–	3
ISSUE ERROR	–	3
ISSUE ABEND	–	10

Figure 78. State 3 – send state

STATE 4 UNMAPPED LUTYPE6.2 CONVERSATIONS RECEIVE PENDING AFTER INVITE		
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
WAIT	-	5
ISSUE ERROR	CDBFREE	10
	Otherwise	3
ISSUE ABEND	-	10
SYNCPOINT (SYNCLEVEL 2 only)	EIBRLDBK (or ROLLEDBACK condition)	5
	Otherwise (transaction will ABEND if SYNCPOINT fails)	5
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3

Figure 79. State 4 — receive pending after INVITE

STATE 5 UNMAPPED LUTYPE6.2 CONVERSATIONS		RECEIVE STATE
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
RECEIVE	CDBCOMPL *	–
	CDBCONF (SYNCLEVEL 1 or 2 only)	6
	CDBSYNC (SYNCLEVEL 2 only)	7
	CDBSYNRB (SYNCLEVEL 2 only)	8
	CDBFREE	10
	CDBRECV	5
	Otherwise	3
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	(transaction will ABEND if ROLLBACK fails)	3
ISSUE ERROR	CDBFREE	10
	Otherwise	3
ISSUE ABEND	–	10
* A zero value in CDBCOMPL indicates incomplete data. CICS saves the remaining data for retrieval by subsequent RECEIVE commands. CDBCOMPL is set when the last part of the data is passed back.		

Figure 80. State 5 – receive state

STATE 6 UNMAPPED LUTYPE6.2 CONVERSATIONS		RECEIVER ISSUE CONFIRMATION
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
ISSUE CONFIRMATION	CDBFREE (saved value)	10
	CDBRECV (saved value)	5
	Otherwise	3
ISSUE ERROR	CDBFREE (saved value)	3
	Otherwise	3
ISSUE ABEND	–	10

Figure 81. State 6 – receiver issue confirmation

STATE 7 UNMAPPED LUTYPE6.2 CONVERSATIONS RECEIVER TAKE SYNCPOINT		
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
SYNCPOINT	CDBFREE (saved value)	10
	CDBRECV (saved value)	5
	Otherwise	3
SYNCPOINT ROLLBACK	-	3
ISSUE ERROR (This will cause the other transaction to abend if it issued SYNCPOINT, but not if it issued ISSUE PREPARE.)	(Now issue SEND INVITE WAIT followed by RECEIVE)	(3) then 5
ISSUE ABEND	-	10

Figure 82. State 7 — receiver take syncpoint

STATE 8 UNMAPPED LUTYPE6.2 CONVERSATIONS RECEIVER TAKE ROLLBACK		
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
SYNCPOINT ROLLBACK	-	5

Figure 83. State 8 — receiver take rollback

STATE 9 UNMAPPED LUTYPE6.2 CONVERSATIONS FREE PENDING AFTER SEND LAST		
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
WAIT	–	10
SEND CONFIRM (no data) (SYNCLEVEL 1 or 2 only)	–	10
SYNCPOINT (SYNCLEVEL 2 only)	–	10
SYNCPOINT ROLLBACK (SYNCLEVEL 2 only)	–	10
ISSUE ABEND	–	10

Figure 84. State 9 – free pending after SEND LAST

STATE 10 UNMAPPED LUTYPE6.2 CONVERSATIONS FREE SESSION		
Commands You Can Issue	What To Test (For CDBERRCD tests, see above)	New State
FREE	–	1

Figure 85. State 10 – free session

Chapter 4.7. CICS-to-CICS distributed transaction processing for MRO and LUTYPE6.1

This chapter tells you how to code CICS transactions that use distributed transaction processing to communicate with other CICS transactions via MRO or LUTYPE6.1 links. For information on distributed transaction processing via LUTYPE6.2 links, see "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171.

There are some differences between DTP under MRO and DTP using an LUTYPE6.1 link. These are pointed out in the text of this chapter. A summary of the differences and restrictions for MRO is given under "Restrictions for multiregion operation" on page 261.

Application design

The starting point for the design of a CICS application is provided by the application requirements developed during the planning activity.

In general, a program that is required to communicate with another CICS program must be designed as one of a pair of conversing applications, not as an isolated entity.

The exchanges between the two applications will be governed primarily by the total requirements of your application. At the same time, however, you must ensure that your CICS transactions follow a well-defined set of protocols, both to achieve correct operation and to deal with unexpected situations. This means that you must at all times be aware of the state of the session, as indicated by the settings of fields in the EXEC interface block (EIB), and use that state information to determine what operations are currently valid, or even mandatory.

A guide to the correct use of EIB fields and command sequences is given in "Command sequences for CICS-to-CICS sessions" on page 256.

CICS commands for MRO and LUTYPE6.1 sessions

The application programming commands that you can use for MRO and LUTYPE6.1 are basically the same, but MRO has a number of restrictions that do not apply to LUTYPE6.1. In general, programs that are written for MRO will operate correctly over LUTYPE6.1 links. The converse is not necessarily true. Differences between MRO and LUTYPE6.1 are pointed out in the relevant places in this chapter and are summarized in "Restrictions for multiregion operation" on page 261.

The commands that can be used to acquire and use CICS-to-CICS sessions are:

- ALLOCATE — used by the front-end transaction to acquire a session to the remote CICS system.

- BUILD ATTACH — used by the front-end transaction to build an LUTYPE6.1 attach header that will be used to initiate the back-end transaction on the remote system.

  The BUILD ATTACH command is seldom used in CICS-CICS communication.

- EXTRACT ATTACH — used by the back-end transaction to recover information from the LUTYPE6.1 attach header that caused it to be initiated.

  The EXTRACT ATTACH command is seldom used in CICS-CICS communication.

- SEND, RECEIVE, and CONVERSE — used by the conversing transactions to send or receive data on the sessions. The first SEND or CONVERSE command issued by the front-end transaction can name the attach header if one is to be sent.

- WAIT TERMINAL SESSION(name) — used by either transaction to ensure that CICS has transmitted any accumulated data or data flow control indicators before it continues with further processing. This command has no effect on MRO sessions.

- ISSUE SIGNAL — used by the receiving transaction to request a change-direction from the sending transaction.

- FREE — used by both transactions to relinquish their use of the session.

Considerations for the front-end transaction

The front-end transaction is responsible for acquiring a session to the remote CICS system and initiating the remote transaction.

| Thereafter, the two transactions become equals. However, the front-end transaction is usually designed as the driving transaction.

Session allocation

You acquire an MRO or LUTYPE6.1 session to a remote system by means of the ALLOCATE command, which has the following format:

```
ALLOCATE {SYSID(name)
    |SESSION(name)}
  [PROFILE(name)]
  [NOQUEUE]

  SYSIDERR,SYSBUSY,CBIDERR
```

You can use the SESSION option to request the use of a specific session only if the system programmer has defined and named individual sessions to the remote system. The definition of individual sessions is not normally necessary for CICS-CICS LUTYPE6.1 links, and cannot be done for MRO links. In most cases, therefore, you will use the SYSID option to name the system with which the session is required.

The name specified in the SYSID option must be the name of an LUTYPE6.1 or MRO link. CICS will raise the SYSIDERR condition if it cannot find the named system.

The PROFILE option allows you to select a specified communication profile for an LUTYPE6.1 session. The profile, which is set up during resource definition, contains a set of terminal control processing options that are to be used for the session.

If you omit the PROFILE option, CICS will use the default profile DFHCICSA. This profile specifies INBFMH(ALL), which means that inbound function management headers will be passed to your program and will cause the INBFMH condition to be raised.

For LUTYPE6.1 sessions, CICS will raise the CBIDERR condition if it cannot find the named (or defaulted) profile.

Depending on the circumstances, CICS sometimes ignores profile specifications. For example, INBFMH is always used for MRO sessions.

The NOQUEUE option allows you to specify explicitly that you do not want your request for a session to be queued if a session is not available immediately. A session is "not immediately available" in any of the following situations:

- All the sessions to the specified system are in use.
- The only available sessions are not bound (in which case CICS would have to bind a session).
- The only available sessions are contention losers (in which case CICS would have to bid to begin a bracket).

The action taken by CICS if a session is not immediately available depends on whether you specify NOQUEUE and also on whether your application has executed a HANDLE command for the SYSBUSY condition. The possible combinations are shown below:

- HANDLE for SYSBUSY condition
  - Control is returned immediately to the label specified in the HANDLE command, whether or not you have specified NOQUEUE.
- No HANDLE for SYSBUSY condition
  - If you have specified NOQUEUE, control is returned immediately to your application program. The SYSBUSY code (X'D3') is set in the EIBRCODE field of the EXEC interface block. You should test this field immediately after issuing the ALLOCATE command.
  - If you have omitted the NOQUEUE option, CICS queues the request until a session is available.

Whether or not a delay in acquiring a session is acceptable will depend on your application.

The session identifier

When a session has been allocated, the name by which it is known is available in the EIBRSRCE field in the EIB. Because EIBRSRCE will probably be overwritten by the next EXEC CICS command, you should acquire the session name immediately. It is the name that you must use in the SESSION option of all subsequent commands that relate to this session.

Attaching the remote transaction

When a session has been acquired, the next step is to cause the remote transaction to be initiated. There are three ways in which this can be done:

1. By using CICS commands to build an attach function management header which can then be sent to the remote system.
2. By building an attach header directly in your application program.
3. By sending a CICS transaction identifier as ordinary data.

These three methods are described in the following sections. The formality of the first two methods (which are for generalized communication between dissimilar LUTYPE6.1 systems) is not normally necessary for CICS-CICS communication, for which you are recommended to use method 3, which initiates the transaction without user-defined attach headers.

Using the BUILD ATTACH command

The LUTYPE6.1 architecture defines a special function management header, called an attach header, which carries the name of the remote process (in CICS terms, the transaction) that is to be initiated, and also contains further session-related information. CICS provides the BUILD ATTACH command to enable the front-end transaction to build an attach header, and the EXTRACT ATTACH command to enable the back-end transaction to obtain information from it. Because these commands are available, you do not need to know the format of an LUTYPE6.1 attach header. In some cases, however, you will need to know the meaning of the information that it carries.

For CICS-CICS communication, the only necessary field in the attach header is the name of the remote transaction. The simplified form of the BUILD ATTACH command that can be used in CICS-to-CICS communication is:

```
BUILD ATTACH
   ATTACHID(name)
  [PROCESS(name)]
```

The ATTACHID option enables you to assign a name to the attach header so that you can refer to it in a subsequent SEND or CONVERSE command. (The BUILD ATTACH command builds an attach header; it does not transmit it.) The process, or transaction, that is to be initiated at the remote system is named in the PROCESS option.

Although they are not mandatory for CICS-to-CICS communication, other fields in the attach header can be useful for passing information from the attaching to the attached transaction. You could, for example, use the QUEUE option to pass the name of a CICS temporary storage queue.

Having built the attach header, you must ensure that it is transmitted with the first data sent to the remote system by naming it in the ATTACHID option of the SEND or CONVERSE command.

Building your own attach header

CICS allows you to build an attach header, or any function management header, as part of your output data. You can therefore initiate the remote transaction by including a LUTYPE6.1 attach header in the output area referenced by the first SEND or CONVERSE command. You must specify the FMH option on the command to tell CICS that the data contains an FMH.

Transaction initiation without attach headers

For CICS-to-CICS communication, you can take a short cut by forgetting about attach headers and simply sending the name of the remote CICS transaction in the first four bytes of the first data that is sent. This will cause normal CICS transaction initiation in the remote system.

If you use this method, you must also ensure that the back-end transaction does not issue an EXTRACT ATTACH command.

Automatic transaction initiation

If the front-end transaction is designed to be started by automatic transaction initiation (ATI) in the local system, and is required to hold a conversation with an MRO or LUTYPE6.1 session as its principal facility, the session will have already been allocated when the transaction starts. You can omit the SESSION option from commands that relate to the principal facility. If, however, you wish to name the session explicitly in these commands, you should obtain the name from EIBTRMID.

Considerations for the back-end transaction

The back-end transaction is initiated either by an LUTYPE6.1 attach FMH received from the remote system or by a transaction name included in the incoming data, and is started with the session as its principal facility.

Acquiring session-related information

If the back-end transaction is designed to be initiated by an attach header, you can use the EXTRACT ATTACH command to recover session-related information from the attach FMH if required, but the use of this command is not mandatory.

The presence of an attach header is indicated by EIBATT, which is set after the first RECEIVE command has been issued.

For CICS-to-CICS communication, the only mandatory field in the attach header is the PROCESS name. It can be recovered by the following command:

```
EXTRACT ATTACH
  [SESSION(data-area)]
  [PROCESS(data-area)]
```

The process name (PROCESS) from the LUTYPE6.1 attach header is returned in the specified data area. If the front-end transaction is designed to pass application-related data in other fields in the attach header, you can acquire it by means of the appropriate options.

If you name the SESSION explicitly on an EXTRACT ATTACH command, you must use the name obtained from EIBTRMID.

Initial state of back-end transaction

| The back-end transaction is always initiated in **receive** state. However, to initiate
| the conversation properly with the front-end transaction, you must issue an EXEC
| CICS RECEIVE command before you do anything else that could affect the link
| (for example, a SYNCPOINT ROLLBACK command).

Using the EXEC CICS ASSIGN command

You may find that two of the options to the EXEC CICS ASSIGN command return unexpected values. A closer look at these will help you to understand why:

USERID

CICS takes the *userid* from one of two sources, depending on how you specified your security requirements.

If you specified ATTACHSEC(Local) on the DEFINE CONNECTION command for the link, whatever was specified for the USERID option on the same DEFINE SESSIONS command is returned under this option. This appears as *blanks* if you let the USERID option default.

For LUTYPE6.1 links, ATTACHSEC(Local) is the only form permitted.

If, however, you requested automatic signon for remote users by specifying ATTACHSEC(Identify) or ATTACHSEC(Verify) on the DEFINE CONNECTION command for the link, the *userid* returned is the one that was sent over the link with the attach request for the transaction.

OPERKEYS

This option returns a 64-bit mask that represents the CICS transaction security profile of the remote user **in the local system**. If the remote user is signed on locally as described in the explanation to USERID above, the returned mask is the value that was defined for the user in the signon table.

If no sign on takes place, the user's security profile defaults to that of the link. The link itself may be signed on, in which case the mask will be taken from the signon table entry for the link. The other possibility is that OPERSECURITY, OPERRSL, or both were specified on the DEFINE SESSIONS command for the link. This *preset* security then determines the value returned under this option. In all cases of default, a value of 1 is returned.

This option cannot give any information about the user's security status in a remote system.

The conversation

The conversation between the front-end and the back-end transactions is held using the SEND, RECEIVE, and CONVERSE commands. Details of these commands for MRO and LUTYPE6.1 are given in the *CICS/MVS Application Programmer's Reference* manual.

In all of these commands, you must name the session in the SESSION option unless the conversation is with the principal facility.

Deferred transmission

On LUTYPE6.1 sessions, when you issue a SEND command, CICS will normally defer sending the data until it becomes clear what your further intentions are. This mechanism enables CICS to avoid unnecessary flows by superimposing, or "piggy-backing", control indicators on the data that is awaiting transmission.

On MRO sessions, SEND data is not deferred, and the later addition of control indicators to the data is not possible. The same command sequence may therefore require more flows on an MRO session than it does on an LUTYPE6.1 session, but, provided that the receiving transaction is correctly designed to be driven by the EIB settings, the same effects will be achieved.

Some of the differences between LUTYPE6.1 and MRO are illustrated in the following command sequence:

Commands	**LUTYPE6.1**	**MRO**
EXEC CICS SEND SESSION(REM1) FROM(data1) LENGTH(251)	sending is deferred	data1 is sent
EXEC CICS SYNCPOINT	syncpoint request is added to data1, and data1 is sent	syncpoint request is sent with null data
EXEC CICS SEND SESSION(REM1) FROM(data2) LENGTH(251) INVITE	sending of data2, with change direction, is deferred	data2, with change direction, is sent
EXEC CICS WAIT SESSION(REM1)	data2, with change direction, is sent	(nothing to do)
EXEC CICS RECEIVE SESSION(REM1) . .		
(INVITE received)		
EXEC CICS SEND SESSION(REM1) FROM(data3) LENGTH(251) LAST	sending of data3, with end bracket indicator, is deferred	data3 is sent, but without end bracket indicator
EXEC CICS SYNCPOINT	syncpoint request is added to data3, and data3 is sent	syncpoint request and end bracket are sent with null data

The WAIT option can be added to the SEND command to cause immediate transmission on LUTYPE6.1 links; for example:

```
SEND SESSION(REM1)
     FROM(data2)
     LENGTH(251)
     INVITE
     WAIT

RECEIVE SESSION(REM1)
```

There are no significant differences between the MRO and LUTYPE6.1 implementations of this command sequence.

A further implementation difference arises between LUTYPE6.1 and MRO for command sequences that contain an implicit change of direction.

For MRO, you must not issue a RECEIVE command unless your transaction is in receive state. For LUTYPE6.1, if you issue a RECEIVE command while your transaction is in send state, CICS will supply the implied options and send the deferred data with the change-direction indicator set. However, better program documentation will be achieved if you specify the options explicitly, and compatibility with MRO links will be maintained.

Using the LAST option

The LAST option on the SEND command indicates the end of the conversation. No further data flows can occur on the session, and the next action must be to free the session. However, the session can still carry CICS syncpointing flows before it is freed.

The circumstances under which session syncpointing occurs, and the ways in which you can avoid syncpointing on the session, differ for LUTYPE6.1 and MRO.

The LAST option and syncpoint flows on LUTYPE6.1 sessions

A syncpoint on an LUTYPE6.1 session is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command.

If your conversation has been terminated by a SEND LAST command, without the WAIT option, transmission will have been deferred, and the syncpointing activity will cause the final transmission to occur with a piggy-backed syncpoint request. The conversation will thus be automatically involved in the syncpoint.

If you do not want the conversation to be involved in the syncpoint (for example, because you know that the remote transaction does not access any recoverable resources) you must issue a SEND LAST WAIT command, or a WAIT TERMINAL SESSION or FREE command, to force the transmission before using a command that causes a syncpoint.

The LAST option and syncpoint flows on MRO sessions

A syncpoint on an MRO session is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command.

If your conversation is terminated by a SEND LAST command, without the WAIT option, the WAIT implicit in all MRO commands will be applied, and the data will be transmitted. However, in anticipation of coming syncpoint flows, CICS will not send the end-bracket with this data.

If you do not want the conversation to be involved in the syncpoint (for example, because you know that the remote transaction does not access any recoverable resources) you must specify the WAIT option explicitly on the SEND LAST command to force the end-bracket to be sent with the data. Alternatively, you could follow the SEND LAST command by a FREE command; in this case CICS will send a null request unit with the end-bracket indicator set.

Freeing the session

The command used to free the session has the following format:

```
FREE SESSION(name)
```

where "name" is the name of the session. You can issue the FREE command at any time that your transaction is in send state. CICS determines whether the end-bracket indicator has already been transmitted, and transmits it if necessary before freeing the session. If there is also deferred data to transmit, the end-bracket indicator is transmitted with the data. Otherwise, the indicator is transmitted with a null RU.

The EXEC interface block (EIB)

Full details of the EIB are given in the *CICS/MVS Application Programmer's Reference* manual. This section highlights the fields that are of particular significance in LUTYPE6.1 and MRO applications. For further details of how and when these fields should be tested, refer to "Command sequences for CICS-to-CICS sessions" on page 256. The EIB should be saved after each MRO or LUTYPE6.1 operation.

Conversation identifier fields

The following EIB fields enable you to obtain the name of the ISC or MRO session.

EIBTRMID
contains the name of the principal facility. For a back-end transaction, or for a front-end transaction started by ATI, it is the conversation identifier (SESSION). You must acquire this name if you want to state the session name of the principal facility explicitly.

EIBRSRCE
contains the session identifier (SESSION) for the session obtained by means of an ALLOCATE statement. You must acquire this name immediately after issuing the ALLOCATE statement.

Procedural fields

These fields contain information on the state of the session. In most cases, the settings relate to the session named in the last-executed RECEIVE or CONVERSE command, and should be tested, or saved for later testing, after the command has been issued. Further information on the use of these fields is given in "Command sequences for CICS-to-CICS sessions" on page 256.

EIBRECV
indicates the conversation state following RECEIVE or CONVERSE. If it is *off* (= X'00'), your conversation partner is inviting you to send, otherwise you would normally issue a further RECEIVE command. It does not necessarily reflect *receive state* at any other time.

EIBCOMPL
This field is used in conjunction with the RECEIVE NOTRUNCATE command; it is set to indicate that the data is complete.

EIBSYNC
indicates that CICS syncpointing is in progress and that the application should issue a SYNCPOINT command.

EIBSYNRB
MRO only. Indicates that CICS syncpointing is in progress and that the application should issue a SYNCPOINT ROLLBACK command.

EIBRLDBK
MRO only. Indicates that the remote transaction has sent SYNCPOINT ROLLBACK in response to a SYNCPOINT request. The transaction that issued the SYNCPOINT command has been rolled back.

EIBSIG
indicates that the conversation partner has issued an ISSUE SIGNAL command.

Note that ISSUE SIGNAL is not a valid command for MRO sessions.

EIBFREE
indicates that the receiver must issue a FREE command for the session.

Informatory fields

EIBEOC
indicates that end-of-chain has been received. This field is applicable only if your program is doing its own chain assembly. Otherwise, EIBEOC is set for every RECEIVE command, and can be ignored.

The following fields contain information about FMHs received from the remote transaction:

EIBATT
indicates that the data received contained an attach header. The attach header is not passed to your application program; however, EIBATT indicates that an EXTRACT ATTACH command is appropriate.

EIBFMH
indicates that the data passed to your application program contains a concatenated FMH. This can result only from the remote CICS transaction building an FMH in the data and specifying the FMH option on a SEND command.

If you want to use these facilities, you must ensure that you use communication profiles that specify INBFMH(ALL). The default profile (DFHCICSA) for the session allocated by the front-end transaction has this specification. However, the default principal facility profile (DFHCICST) for the back-end transaction does not. Further information on this subject is given under "Defining communication profiles" on page 151.

Command sequences for CICS-to-CICS sessions

The command sequences that you use to communicate between the front-end and the back-end transactions are governed both by the requirements of your application and by a set of high-level protocols designed to ensure that commands are not issued in inappropriate circumstances.

The protocols presented in this section do not cover all possible command sequences. However, by following them, you will ensure that each transaction takes account of the requirements of the other, and hence reduce the error rate during program development.

The protocols are based on the concept of a number of states. These states apply only to the particular conversation, not to your application program as a whole. In each state, there are a number of commands that might most reasonably be issued. After the command has been issued, fields in the EIB must be tested in the order shown in the state diagrams, Figure 86 on page 257 through Figure 93 on page 261, to check on the current requirements of the conversation. The results of these tests, together with the command that has been issued, may cause a transition to another state, in which another set of commands become appropriate.

The states that are defined for the purposes of this section are:

- State 1 — Session not allocated
- State 2 — Send state
- State 3 — Receive pending after INVITE
- State 4 — Receive state
- State 5 — Receiver take syncpoint
- State 6 — Receiver rollback or free session (MRO only)
- State 7 — Free pending after SEND LAST
- State 8 — Free session

Initial states

Normally, the front-end transaction in a conversation will initially be in state 1 — session not allocated — and must issue an ALLOCATE command to acquire a session.

An exception to this occurs when the front-end transaction is started by automatic transaction initiation (ATI) in the local system, with an LUTYPE6.1 or MRO session as its principal facility. In this case the session is already allocated, and the transaction in is state 2. For transactions of this type, you must immediately obtain the session name from EIBTRMID if you want to be able to name the session explicitly on subsequent commands.

You must always assume that the back-end transaction is initially in state 4 (receive state). Even if it is designed only to send data to the front-end transaction, you must issue a RECEIVE to receive the SEND INVITE issued by the front-end transaction and get into send state.

State diagrams

The following diagrams are intended to enable you to construct valid command sequences. Each diagram relates to one particular state, as previously defined, and shows the commands that you might reasonably issue and the tests that you must make after issuing the command. Where more than one test is shown, they must be made in the order indicated.

The combination of the command issued and a particular positive test result lead to a resultant state, shown in the final column. In an MRO or LUTYPE6.1 conversation (just as in an LUTYPE6.2 mapped conversation), your program should not make assumptions as to the current or next state of the conversation. Always test the EIB to determine the current state of the conversation and what to do next.

See "Using the state diagrams" on page 205 for an example of how to use the state diagrams.

Other tests

The tests that are shown in the diagrams are those that are significant to the state of the conversation. Tests for other conditions that may possibly arise, for example, INVREQ or NOTALLOC, should be made in the normal way.

Also, if your transaction is expected to receive an incoming SIGNAL command, you should either execute a HANDLE command for the SIGNAL condition or test EIBSIG after each command.

STATE 1 LUTYPE6.1 and MRO CONVERSATIONS		SESSION NOT ALLOCATED
Commands You Can Issue	What To Test	New State
ALLOCATE [NOQUEUE] *	SYSIDERR	1
	SYSBUSY *	1
	Otherwise (obtain session name from EIBRSRCE)	2

* If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command and do not code a HANDLE command for the SYSBUSY condition.

If you want control to be returned to your program if a session is not immediately available, either specify NOQUEUE on the ALLOCATE command and test EIBRCODE for SYSBUSY (X'D3'), or code a HANDLE CONDITION SYSBUSY command.

Figure 86. State 1 — session not allocated

STATE 2 LUTYPE6.1 and MRO CONVERSATIONS		SEND STATE
Commands You Can Issue *	What To Test	New State
SEND		2
SEND INVITE	–	3
SEND INVITE WAIT	–	4
SEND LAST	–	7
SEND LAST WAIT	–	8
CONVERSE Equivalent to: SEND INVITE WAIT RECEIVE	Go to the STATE 4 table and make the tests shown for the RECEIVE command	–
RECEIVE (LUTYPE6.1 only)	Go to the STATE 4 table and make the tests shown for the RECEIVE command	–
SYNCPOINT	EIBRLDBK (or ROLLEDBACK condition) (MRO only)	4
	Otherwise (transaction will ABEND if SYNCPOINT fails)	2
SYNCPOINT ROLLBACK (MRO only)	(transaction will ABEND if ROLLBACK fails)	2
FREE Equivalent to: SEND LAST WAIT FREE	–	1
* For the front-end transaction, the first command used after the session has been allocated must be a SEND command or CONVERSE command that initiates the back-end transaction in one of the ways described under 'Attaching the Remote Transaction' above.		

Figure 87. State 2 — send state

STATE 3 LUTYPE6.1 and MRO CONVERSATIONS RECEIVE PENDING AFTER INVITE		
Commands You Can Issue	What To Test	New State
WAIT TERMINAL SESSION	-	4
SYNCPOINT (LUTYPE6.1 only)	(transaction will ABEND if SYNCPOINT fails)	4

Figure 88. State 3 — receive pending after INVITE

STATE 4 LUTYPE6.1 and MRO CONVERSATIONS RECEIVE STATE		
Commands You Can Issue	What To Test	New State
RECEIVE [NOTRUNCATE] *	EIBCOMPL *	-
	EIBSYNC	5
	EIBSYNRB (MRO only)	6
	EIBFREE	8
	EIBRECV	4
	Otherwise	2
SYNCPOINT ROLLBACK (MRO only)	(transaction will ABEND if ROLLBACK fails)	2
* If NOTRUNCATE is specified, a zero value in EIBCOMPL indicates that the data passed to the application by CICS is incomplete (because, for example, the data-area specified in the RECEIVE command is too small). CICS will save the remaining data for retrieval by subsequent RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back. If the NOTRUNCATE option is not specified, overlength data is indicated by the LENGERR condition, and the remaining data is discarded by CICS.		

Figure 89. State 4 — receive state

STATE 5 LUTYPE6.1 and MRO CONVERSATIONS RECEIVER TAKE SYNCPOINT		
Commands You Can Issue	What To Test	New State
SYNCPOINT	EIBFREE (saved value)	8
	EIBRECV (saved value)	4
	Otherwise	2
SYNCPOINT ROLLBACK (MRO Only)	EIBFREE (saved value)	8
	EIBRECV (saved value)	4
	Otherwise	2

Figure 90. State 5 – receiver take syncpoint

STATE 6 MRO CONVERSATIONS RECEIVER ROLLBACK OR FREE SESSION		
Commands You Can Issue	What To Test	New State
SYNCPOINT ROLLBACK	EIBFREE (saved value)	8
	EIBRECV (saved value)	4
	Otherwise	2
FREE	–	1

Figure 91. State 6 – receiver rollback or free session

STATE 7 LUTYPE6.1 and MRO CONVERSATIONS FREE PENDING AFTER SEND LAST		
Commands You Can Issue	What To Test	New State
WAIT TERMINAL SESSION (do not use for MRO sessions)	–	8
SYNCPOINT	–	8
FREE	–	1

Figure 92. State 7 – free pending after SEND LAST

STATE 8 LUTYPE6.1 and MRO CONVERSATIONS		FREE SESSION
Commands You Can Issue	What To Test	New State
FREE	-	1

Figure 93. State 8 – free session

Restrictions for multiregion operation

This section summarizes the restrictions that apply when you are using distributed transaction processing over MRO links.

1. You cannot use the following commands on the MRO session:

   ```
   EXTRACT TCT
   ISSUE SIGNAL
   WAIT SIGNAL
   ISSUE DISCONNECT
   ```

2. You should not use the WAIT command, because it is functionally a null operation on MRO sessions. In particular, the command sequence:

   ```
   SEND LAST
   WAIT
   SYNCPOINT (or RETURN)
   ```

   will cause a syncpoint flow on an MRO session but not on an LUTYPE6.1 session.

   To avoid syncpoint flows on an MRO session, use either

   ```
     SEND LAST WAIT
   or
     SEND LAST
     FREE
   ```

   The MRO and LUTYPE6.1 implementations of these sequences are functionally equivalent.

3. You cannot use the following command sequence on an MRO session:

   ```
   SEND INVITE
   SYNCPOINT
   RECEIVE
   ```

4. You cannot use the following command sequence on an MRO session:

   ```
   SEND
   RECEIVE
   ```

   You must always code the INVITE option explicitly.

5. You are required always to test the EIB settings after issuing a RECEIVE command on an MRO session, rather than making any assumptions about the state of the session. Failure to observe the EIB states can lead to errors.

   A program designed to be driven by the EIB states after issuing a RECEIVE command will operate correctly on both MRO and LUTYPE6.1 sessions.

Chapter 4.8. CICS-to-IMS applications

This chapter tells you how to code CICS transactions that communicate with an IMS system. For full details of IMS ISC, you should read the appropriate IMS publications. This chapter is intended to provide sufficient information about IMS to enable you to work with your IMS counterpart to implement a CICS-to-IMS ISC application.

The design of CICS-to-IMS ISC applications

There are many differences between CICS and IMS, both in their architecture and in their application and system programming requirements.

The design of CICS-to-IMS ISC applications involves principally CICS application programming and IMS system definition. This difference reflects where the control lies in each of the two systems.

CICS is a **direct control** system. Data entered at a terminal causes CICS to invoke the appropriate application program to process the incoming data. The data is stored, rather than queued, and the application "owns" the terminal until it completes its processing and terminates. In CICS ISC, the application program is involved with data flow protocols, with syncpointing, and, in general, with most system services.

In contrast, IMS is a **queued** system. All input and output messages are queued by the IMS control region on behalf of the related application programs and terminals. The queuing of messages and the processing of messages are therefore performed asynchronously. (Do not confuse this use of asynchronous with CICS asynchronous processing.) This is illustrated in Figure 94 on page 264.

As a result of this type of system design, IMS application programs do not have direct control over IMS system resources, nor do they become directly involved in the control of intersystem communication. IMS message switching is handled entirely in the IMS control region; the message processing region is not involved.

Data formats

Messages transmitted between CICS and IMS can have either of the following data formats:

- Variable length variable blocked (VLVB)
- Chain of RUs.

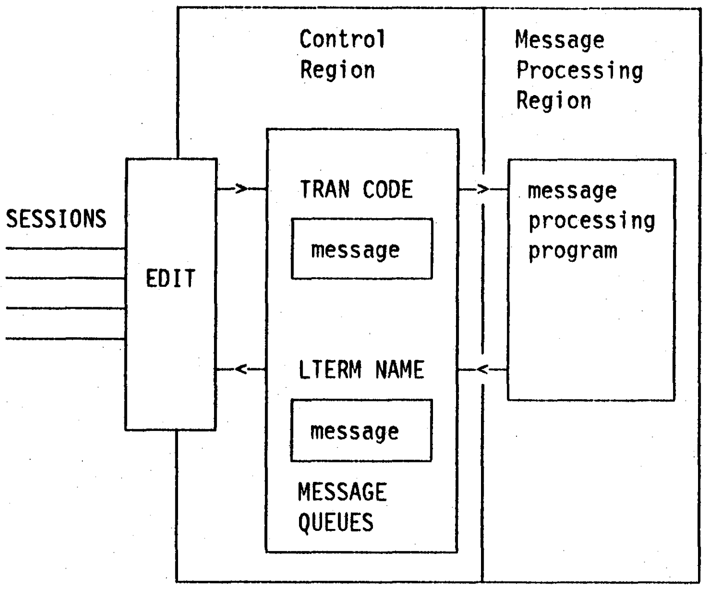
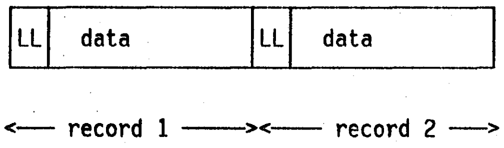


Figure 94. Basic IMS message queuing

In normal CICS communication with logical units, "chain of RUs" is the default data format. In IMS, VLVB is the default. In CICS-to-IMS communication, the format that is being used is specified in the LUTYPE6.1 attach headers that are sent with the initial data.

Variable length variable blocked

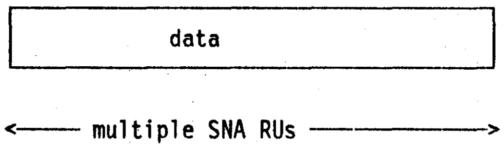
In VLVB format, a message can contain multiple records. Each record is prefixed by a 2-byte length field, as shown here.



In CICS, the I/O area contains a complete message, which can contain one or more records. The blocking of records for output, and the deblocking on input, must be done by your CICS application program.

Chain of RUs

In this format, which is the most common CICS format, a message is transmitted as multiple SNA RUs, as shown here.



In CICS, the I/O area contains a complete message.

Forms of intersystem communication with IMS

There are three forms of CICS-to-IMS that must be considered:

1. Asynchronous processing using CICS START and RETRIEVE commands.
2. Asynchronous processing using CICS SEND LAST and RECEIVE commands.
3. Distributed transaction processing (that is, synchronous processing) using CICS SEND and RECEIVE commands.

The basic differences between these forms of communication are described in "Chapter 1.5. Asynchronous processing" on page 33 and "Chapter 1.7. Distributed transaction processing" on page 55.

In any particular application that involves communication between CICS and IMS, the intersystem communication must be initiated by one or other of the two systems. For example, if a CICS terminal operator initiates a CICS transaction that is designed to obtain data from a remote IMS system, the intersystem communication for the purposes of this application is initiated by CICS.

The system that initiates intersystem communication for any particular application is the **front-end** system as far as that application is concerned. The other system is called the **back-end** system.

When CICS is the front-end, it supports all three types of intersystem communication listed above. The form of communication that can be used for any particular application depends on the IMS transaction type or on the IMS facility that is being initiated. Details of the forms of communication that IMS supports when it is the back-end system are given in the *IMS Version 2 Programming Guide for Remote SNA Systems*.

When IMS is the front-end system, it always uses asynchronous processing (corresponding to the CICS START/RETRIEVE interface) to initiate communication with CICS.

Asynchronous processing

In asynchronous processing, the intersystem session is used only to pass an initiation request, together with various items of data, from one system to the other. All other processing is independent of the session that is used to pass the request.

The two application programming interfaces available in CICS for asynchronous processing are:

- The START/RETRIEVE interface
- The SEND/RECEIVE interface.

The START/RETRIEVE interface

The full syntax of the CICS START and RETRIEVE "interval control" commands is given in the *CICS/MVS Application Programmer's Reference* manual. The relevant forms of these commands, together with the specific meanings of the command options in a CICS-to-IMS intersystem communication environment, are given in "The START command" on page 268 and "The RETRIEVE command" on page 269.

CICS front end

When CICS is the front-end system, you can use CICS START and RETRIEVE commands to process IMS nonresponse mode and nonconversational transactions, messages switches, and the IMS /DIS, /RDIS, and /FOR operator commands.

The general command sequence for your application program is shown in Figure 95.

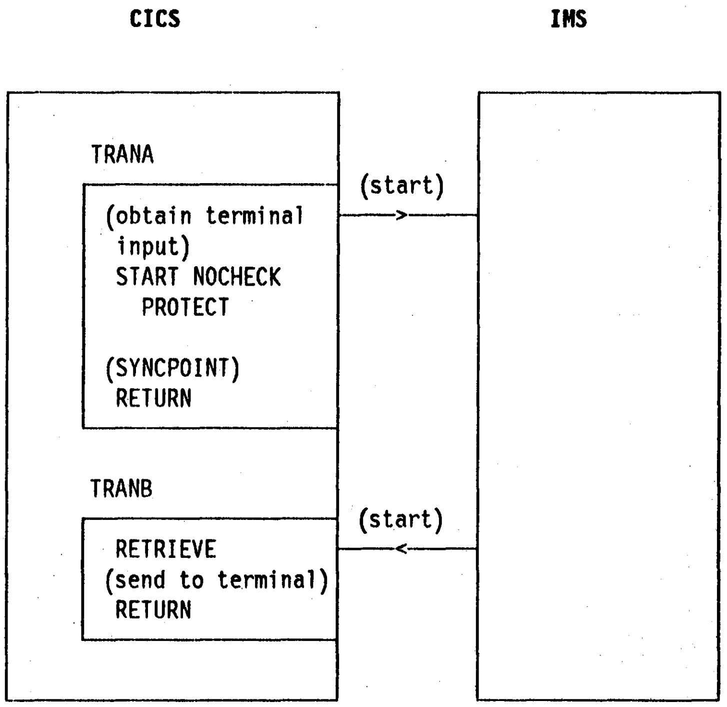


Figure 95. START/RETRIEVE asynchronous processing – CICS front-end

Note: Unless change direction is also sent, IMS expects the above commands to be sent requesting a definite response. If you code the PROTECT option on the START command, CICS sends the command requesting a definite response. For further information, see the *IMS Version 2 Programming Guide for Remote SNA Systems.*

After transaction TRANA has obtained an input message from the terminal, it issues a START NOCHECK command to initiate the remote IMS transaction. The START command specifies the name of the IMS editor that is to be initiated to process the message and the IMS transaction or logical terminal (LTERM) that is to receive the message. It also specifies the name of the CICS transaction that is to receive the reply and the name of the associated CICS terminal.

The PROTECT option can be specified on the START command to ensure delivery of the message to IMS.

The start request is not shipped until your application program either issues a SYNCPOINT command or terminates. However, the request will not carry the syncpoint-indicator unless PROTECT was specified on the START command.

Although CICS allows an application program to issue multiple START NOCHECK commands without intervening syncpoints (see "Deferred sending of START requests" on page 38), this technique is not recommended for CICS-to-IMS communication.

IMS sends the reply by issuing a "start" request which is handled in the normal way by the CICS mirror transaction. The request specifies the CICS transaction and terminal that you named in the original START command. The transaction that is started (TRANB) can then retrieve the reply by issuing a RETRIEVE command.

In the above example, it has been assumed that there are two separate CICS transactions; one to issue the START command and one to receive the reply and return it to the terminal. These two transactions can be combined, and there are two ways in which this can be done.

The first method is to write a transaction that contains both the START and the RETRIEVE processing, but which performs only one of these functions for a particular execution. The CICS ASSIGN STARTCODE command can be used to determine whether the transaction was initiated from the terminal, in which case the START processing is required, or by a start request, in which case the RETRIEVE processing is required.

The second method is to write a transaction that, having issued the START command, issues a SYNCPOINT command to "flush" the start request and then waits for the reply by issuing a RETRIEVE command with the WAIT option. The terminal will be held by the transaction during this time, and CICS will return control to the transaction when input directed to the same transaction and terminal is received.

In all cases, you should make no assumptions about the timing of the reply or its relationship to a particular, previous, request. A RETRIEVE command will retrieve any outstanding data destined for the same transaction and terminal. The correlation of requests and replies is the responsibility of your application program.

IMS front end

When IMS is the front-end system, the only supported flow is the asynchronous start request. Your application program must use the RETRIEVE command to obtain the request from IMS, followed by a START command to send the reply if one is required.

The general command sequence for your application program is shown in Figure 96.

If a reply to the retrieved data is required, your start command must specify the IMS editor and transaction or LTERM name obtained by the RETRIEVE command.

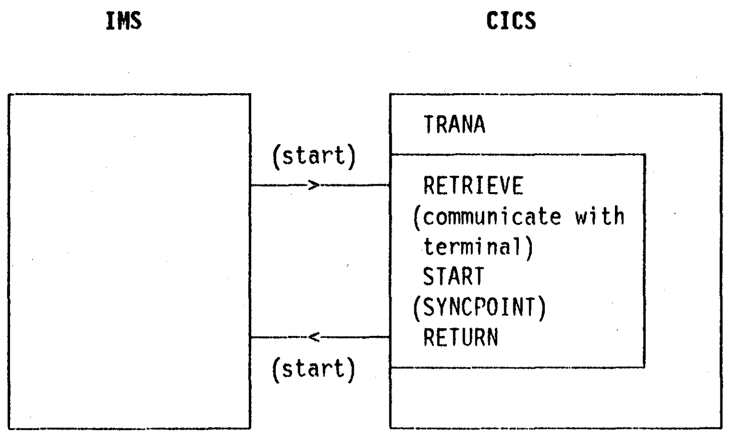


Figure 96. RETRIEVE/START asynchronous processing — IMS front-end

The START command

This section show the format of the START command that is used to schedule remote IMS transactions. Note that no interval control is possible (although it is not an error to specify INTERVAL(0)) and that the NOCHECK option must be specified.

```
EXEC CICS START
     [SYSID(name)]
     TRANSID(name)
     [FROM(data-area) LENGTH(parameter)]
     [TERMID(name)]
     [RTRANSID(name)]
     [RTERMID(name)]
      NOCHECK
     [PROTECT]
     [FMH]
```

SYSID(name)
specifies the name of the remote IMS system. This is the name that is specified by the system programmer in the CONNECTION operand of the DEFINE CONNECTION command (or the SYSIDNT operand of the DFHTCT TYPE=SYSTEM macro) that defines the link to the remote system. You need this option only if you are required to name the remote system explicitly.

TRANSID(name)
specifies the name of the IMS editor that is to be initiated to process the message. It must be an alias (not exceeding four characters) of ISCEDT, or an MFS MID name.

Alternatively, it can name a transaction that is defined as "remote" in the local CICS program control table. In this case, the SYSID option is not used.

The PCT entry must name the required IMS editor in the RMTNAME operand, which can be up to eight characters long.

FROM(data-area)
specifies the data that is to be sent. The format of the data (VLVB or chain of RUs) must match the format specified in the RECORDFORMAT option of the DEFINE CONNECTION command (or the RECFM operand of the DFHTCT TYPE=SYSTEM macro) that defines the remote IMS system (see "Chapter 3.1. Defining links to remote systems" on page 91).

LENGTH(parameter)
specifies, as a halfword binary value, the length of the data specified in the FROM option.

TERMID(name)
specifies the primary resource name that is to be assigned to the remote process. For IMS, it is a transaction code or an LTERM name.

If this option is omitted, you must specify the transaction code or the LTERM name in the first eight characters of the data named in the FROM option. You must use this method if the name exceeds four characters (the CICS limit for the TERMID option) or if IMS password processing is required.

RTRANSID(name)
specifies the name of the transaction that is to be invoked when IMS returns a reply to CICS. The name must not exceed four characters in length.

RTERMID(name)
specifies the name of the terminal that is to be attached to the transaction specified in the RTRANSID option when it is invoked. The name must not exceed four characters in length.

NOCHECK
This option is mandatory.

PROTECT
specifies that the remote IMS transaction must not be scheduled until the local CICS transaction has taken a syncpoint.

FMH
specifies that the user data to be passed to the started task contains FMHs. This option is not normally used.

The RETRIEVE command

This section show the format of the RETRIEVE command that is used retrieve data sent by IMS.

```
EXEC CICS RETRIEVE
     [{INTO(data-area)|SET(pointer-ref)}
       LENGTH(data-area)]
     [RTRANSID(data-area)]
     [RTERMID(data-area)]
     [WAIT]
```

INTO(data-area)
specifies the user data area into which the data retrieved from IMS is to be written.

LENGTH(parameter)
specifies the halfword binary length of the retrieved data.

For a RETRIEVE command with the INTO option, the parameter must be a data area that specifies the maximum length of data that the program is prepared to handle. If the value specified is less than zero, zero is assumed. If the length of the data exceeds the value specified, the data is truncated to that value and the LENGERR condition occurs. On completion of the retrieval operation, the data area is set to the original length of the data.

For a RETRIEVE command with the SET option, the parameter must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

RTERMID(data-area)
specifies an area to receive the return primary resource name sent by IMS. It is either a transaction name or an LTERM name.

Your application can use this name in the TERMID of the START command used to send the reply.

RTRANSID(data-area)
specifies an area to receive the return destination process name sent by IMS. It is either an MFS MID name chained from an output MOD, or is blank.

Your application can use this name in the TRANSID of a subsequent START command.

SET(pointer-ref)
specifies the pointer reference to be set to the address of the data retrieved from IMS.

WAIT
specifies that control is not to be returned to your application program until data is sent by IMS.

If WAIT is not specified, the ENDDATA condition will be raised if no data is available. If WAIT is specified, the ENDDATA condition will be raised only if CICS is shut down before any data becomes available.

The use of the WAIT option is not generally recommended, because it can cause intervening messages (not the expected reply) to be retrieved.

The asynchronous SEND/RECEIVE interface

This form of asynchronous processing is, in CICS, a special case of distributed transaction processing. A CICS transaction acquires the use of a session to a remote system, and uses the session for a single transmission (using a SEND command with the LAST option) to initiate a remote transaction and send data to it. The reply from the remote system causes a CICS transaction to be initiated just as if it were a back-end transaction in normal DTP. This transaction, however, can issue only a single RECEIVE command, and must then free the session.

Except for these additional restrictions, you can design your application according to the rules given for distributed transaction processing in "Distributed transaction processing" on page 271.

The general command sequence for asynchronous SEND/RECEIVE application programs is shown in Figure 97.

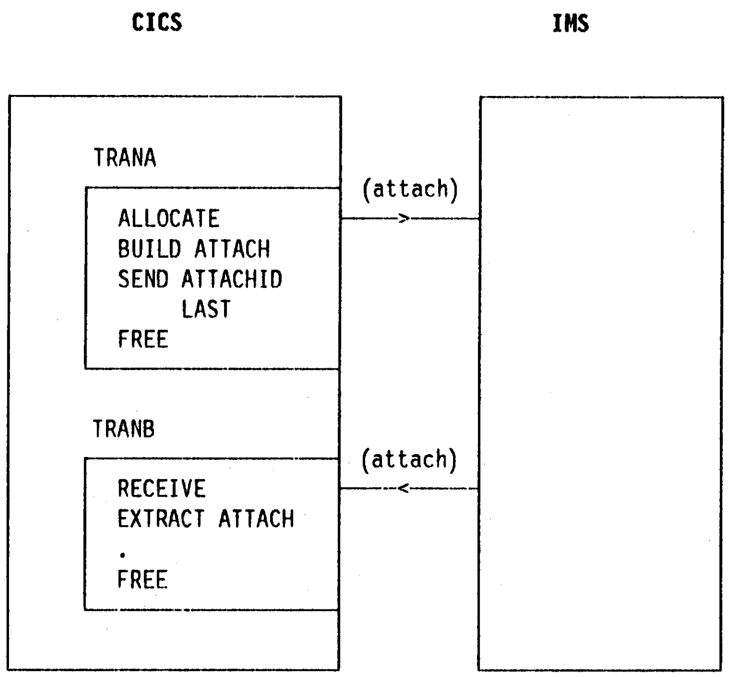


Figure 97. SEND/RECEIVE asynchronous processing – CICS front-end

Distributed transaction processing

This section describes application programming for CICS-to-IMS distributed transaction processing.

CICS commands for CICS-to-IMS sessions

The commands that can be used to acquire and use CICS-to-IMS sessions are:

- ALLOCATE – used to acquire a session to the remote IMS system.
- BUILD ATTACH – used to build an LUTYPE6.1 attach header that will be used to initiate a transaction on a remote IMS system.
- EXTRACT ATTACH – used by a CICS transaction to recover information from the LUTYPE6.1 attach header that caused it to be initiated. This command is required only for SEND/RECEIVE asynchronous processing.
- SEND, RECEIVE, and CONVERSE – used by the CICS transaction to send or receive data on the session. The first SEND or CONVERSE command issued by a front-end CICS transaction must name the attach header that has been defined by the BUILD ATTACH command.
- WAIT TERMINAL SESSION(name) – used to ensure that CICS has transmitted any accumulated data or data flow control indicators before it continues with further processing. This command is required only for certain highly-specialized applications.

- ISSUE SIGNAL SESSION(name) — used by a transaction that is in receive state to request an invitation to send (change-direction) from IMS. This command is required only for certain highly-specialized applications.
- FREE — used by a CICS transaction to relinquish its use of the session.

Considerations for the front-end transaction

Except in the special case of the receiving transaction in SEND/RECEIVE asynchronous processing, the CICS transaction is always the front-end transaction in CICS-to-IMS DTP.

The front-end transaction is responsible for acquiring a session to the remote IMS system and initiating the remote transaction.

Thereafter, the two transactions become equals. However, the front-end transaction is usually designed as the driving transaction.

Session allocation

You acquire an LUTYPE6.1 session to a remote IMS system by means of the ALLOCATE command, which has the following format:

```
ALLOCATE {SYSID(name)|SESSION(name)}
  [PROFILE(name)]
  [NOQUEUE]
```

You can use the SESSION option to request the use of a specific session to the remote IMS system, or you can use the SYSID option to name the remote system and allow CICS to select an available session. The use of the SESSION option is not normally recommended, because it can result in an application program queuing on a specific session when others are available. In most cases, therefore, you will use the SYSID option to name the system with which the session is required.

CICS will raise the SYSIDERR condition if it cannot find a named system, or the SESSIONERR condition if it cannot find a named session.

The PROFILE option allows you to select a specified communication profile for an LUTYPE6.1 session. The profile, which is set up during resource definition, contains a set of terminal control processing options that are to be used for the session.

If you omit the PROFILE option, CICS will use the default profile DFHCICSA. This profile specifies INBFMH(ALL), which means that inbound function management headers will be passed to your program and will cause the INBFMH condition to be raised.

The NOQUEUE option allows you to specify explicitly that you do not want your request for a session to be queued if a session is not available immediately. A session is "not immediately available" in any of the following situations:

- All the sessions to the specified system are in use.
- The only available sessions are not bound (in which case CICS would have to bind a session).

- The only available sessions are contention losers (in which case CICS would have to bid to begin a bracket).

The action taken by CICS if a session is not immediately available depends on whether you specify NOQUEUE and also on whether your application has executed a HANDLE command for the SYSBUSY condition. The possible combinations are shown below:

- HANDLE for SYSBUSY condition
  - Control is returned immediately to the label specified in the HANDLE command, whether or not you have specified NOQUEUE.
- No HANDLE for SYSBUSY condition
  - If you have specified NOQUEUE, control is returned immediately to your application program. The SYSBUSY code (X'D3') is set in the EIBRCODE field of the EXEC interface block. You should test this field immediately after issuing the ALLOCATE command.
  - If you have omitted the NOQUEUE option, CICS queues the request until a session is available.

Whether or not a delay in acquiring a session is acceptable will depend on your application.

Similar considerations apply to an ALLOCATE command that specifies SESSION rather than SYSID. The associated condition is SESSBUSY (EIBRCODE=X'D2').

The session identifier

When a session has been allocated, the name by which it is known is available in the EIBRSRCE field in the EIB. Because EIBRSRCE will probably be overwritten by the next EXEC CICS command, you must acquire the session name immediately. It is the name that you must use in the SESSION option of all subsequent commands that relate to this session.

Automatic transaction initiation

If the front-end transaction is designed to be started by automatic transaction initiation (ATI) in the local system, and is required to hold a conversation with an LUTYPE6.1 session as its principal facility, the session will have already been allocated when the transaction starts. You can omit the SESSION option from commands that relate to the principal facility. If, however, you wish to name the session explicitly in these commands, you should obtain the name from EIBTRMID.

Attaching the remote transaction

When a session has been acquired, the next step is to cause the remote IMS process to be initiated.

The LUTYPE6.1 architecture defines a special function management header, called an attach header, which carries the name of the remote process (in CICS terms, the transaction) that is to be initiated, and also contains further session-related information.

CICS provides the BUILD ATTACH command to enable a CICS application program to build an attach header to send to IMS, and the EXTRACT ATTACH command to enable information to be obtained from attach headers received from IMS.

Because these commands are available, you do not need to know the detailed format of an LUTYPE6.1 attach header. In most cases, however, you will need to know the meaning of the information that it carries. The format of the BUILD ATTACH command is:

```
BUILD ATTACH
   ATTACHID(name)
  [PROCESS(name)]
  [RESOURCE(name)]
  [RPROCESS(name)]
  [RRESOURCE(name)]
  [QUEUE(name)]
  [IUTYPE(data-value)]
  [DATASTR(data-value)]
  [RECFM(data-value)]
```

The options of the BUILD ATTACH command have the following meanings:

ATTACHID(name)
The ATTACHID option enables you to assign a name to the attach header so that you can refer to it in a subsequent SEND or CONVERSE command. (The BUILD ATTACH command builds an attach header; it does not transmit it.)

PROCESS(name)
This corresponds to the process name, ATTDPN, in an attach FMH. It specifies the remote process that is to be initiated.

In CICS-to-IMS communication, the remote process is always an editor. It can be ISCEDT (or its alias), BASICEDT, or an MFS MID name. The process name must not exceed 8 characters.

If the PROCESS option is omitted, IMS assumes ISCEDT.

RESOURCE(name)
This corresponds to the resource name, ATTPRN, in an attach FMH.

The RESOURCE option specifies the primary resource name that is to be assigned to the remote process that is being initiated.

In CICS-to-IMS communication, the primary resource name is either an IMS transaction code or a logical terminal name. You can omit the RESOURCE option if the IMS message destination is specified in the first eight bytes of the message or if the destination is preset by the IMS operator.

If a primary resource name is supplied to IMS, the data stream is not edited for destination and security information. You should therefore omit the RESOURCE option if IMS password processing is required.

The name in the RESOURCE option is ignored during conversational processing, or if the remote process is BASICEDT.

The name must not exceed eight characters.

RPROCESS(name)

This corresponds to the return process name, ATTRDPN, in an attach FMH.

The RPROCESS option specifies a suggested return destination process name. IMS returns this name as a destination process name (ATTDPN) when it sends a reply to CICS, although the name may be overridden by MFS.

CICS uses the returned destination process name to determine the transaction that is to be attached after a session restart. At any other time, it is ignored. The RPROCESS option should therefore name a transaction that will handle any queued messages when it is attached by CICS at session restart following a session failure.

RRESOURCE(name)

This corresponds to the return resource name, ATTRPRN, in an attach FMH.

The RRESOURCE option specifies a suggested primary resource name that is to be assigned to the return process. IMS returns this name as the resource name (ATTPRN) when it sends a reply to CICS.

Although CICS normally ignores this field, one use for it in ISC is to specify a CICS terminal to which output messages occurring after session restart should be sent.

QUEUE(name)

This corresponds to the queue name, ATTDQN, in an attach FMH.

The QUEUE option specifies a queue that can be associated with the remote process. In CICS-to-IMS communication, it is used only to send a paging request to IMS during demand paging. The name used must be the one obtained by a previous EXTRACT ATTACH QNAME command. The name must not exceed 8 characters.

IUTYPE(data-value)

This corresponds to the interchange unit field, ATTIU, in an attach FMH.

The IUTYPE option specifies SNA chaining information for the message. The value is halfword binary. The bits in the binary value are used as follows:

0-7	X'00'	– must be set to zero
8-15	X'00'	– multiple RU chains
	X'01'	– single RU chains

If the option is omitted, multiple RU chains are assumed. Because IMS will always accept this value, regardless of the actual message chain types, you are advised to omit the IUTYPE option.

DATASTR(data-value)

This corresponds to the data stream profile field, ATTDSP, in an attach FMH.

The DATASTR option is used to select an IMS component.

The value is halfword binary. The bits in the binary value are used as follows:

0-7	X'00'	– must be set to zero
8-11	0000	– (user-defined datastream)
12-15	0000	– IMS Component 1
	0001	– IMS Component 2
	0010	– IMS Component 3
	0011	– IMS Component 4

If the DATASTR option is omitted, IMS Component 1 is assumed.

RECFM(data-value)

This corresponds to the deblocking algorithm field, ATTDBA, in an attach FMH.

The RECFM option specifies the format of the user-data that will be sent to the remote process.

The name must represent a halfword binary value. The bits in the binary value are used as follows:

0-7	X'00'	– reserved – must be set to zero
8-15	X'01'	– variable length variable blocked (VLVB) format
	X'04'	– chain of RUs

If VLVB is specified, your application program must add a 2-byte binary length field in front of each record. If "chain of RUs" is specified, you can send your data in the usual way; no length fields are required.

A record is interpreted by IMS as either a segment of a message (without MFS) or an MFS record (with MFS).

The RECFM option indicates only the type of the message format. Multiple records can be sent by one SEND command. In this case, it is the responsibility of your application program to perform the blocking.

Having built the attach header, you must ensure that it is transmitted with the first data sent to the remote system by naming it in the ATTACHID option of the SEND or CONVERSE command.

Building your own attach header

CICS allows you to build an attach header, or any function management header, as part of your output data. You can therefore initiate the remote transaction by including a LUTYPE6.1 attach header in the output area referenced by the first SEND or CONVERSE command. You must specify the FMH option on the command to tell CICS that the data contains an FMH.

Considerations for the back-end transaction

A CICS transaction can be the back-end transaction in CICS-to-IMS communication only in the special case of SEND/RECEIVE asynchronous processing.

The transaction is initiated by an LUTYPE6.1 attach FMH received from the remote IMS system, and is allowed to issue only a single RECEIVE command, possibly followed by an EXTRACT ATTACH command.

Acquiring session-related information

You can use the EXTRACT ATTACH command to recover session-related information from the attach FMH if required, but the use of this command is not mandatory.

The presence of an attach header is indicated by EIBATT, which is set after the first RECEIVE command has been issued.

The format of the EXTRACT ATTACH command is:

```
EXTRACT ATTACH
  [SESSION(data-area)]
  [PROCESS(data-area)]
  [RESOURCE(data-area)]
  [RPROCESS(data-area)]
  [RRESOURCE(data-area)]
  [QUEUE(data-area)]
  [IUTYPE(data-area)]
  [DATASTR(data-area)]
  [RECFM(data-area)]
```

The options of the EXTRACT ATTACH command have the following meanings:

DATASTR(data-area)
contains a value specifying the IMS output component.

The data-area must be a halfword binary field. The values set by IMS are as follows:

0-7	X'00'	– must be set to zero
8-11	0000	– (user-defined datastream)
12-15	0000	– IMS Component 1
	0001	– IMS Component 2
	0010	– IMS Component 3
	0011	– IMS Component 4

IUTYPE(data-area)
indicates SNA chaining information for the message and the type of MFS paged output.

The data-area must be a halfword binary field. The values set by IMS are as follows:

0-7	X'00'	– must be set to zero
8-15	X'00'	– multiple RU chains, MFS autopaged output
	X'01'	– single RU chains, MFS non-paged output
	X'05'	– single RU chains, MFS demand-paged output

PROCESS(data-area)
IMS returns either the return destination process name specified in the RPROCESS option of the BUILD ATTACH command, or a value set by the MFS MOD.

QUEUE(data-area)
IMS returns the LTERM name associated with the ISC session when MFS demand-paged output is ready to be sent. The returned value should be

used in the QMODEL FMH and the BUILD ATTACH QNAME when a paging request is to be sent.

RECFM(data-area)
contains the data format of the incoming user message.

The data-area must be a halfword binary field. The values set by IMS are as follows:

0-7	X'00'	– reserved – must be set to zero
8-15	X'01'	– variable length variable blocked (VLVB) format
	X'04'	– chain of RUs (can also be X'00' or X'05')

If VLVB is specified, your application program must deblock the message by using the halfword binary length field that precedes each record.

RESOURCE(data-area)
IMS returns either the return resource name specified in the RRESOURCE option of the BUILD ATTACH command, or a value set by the MFS MOD.

RPROCESS(data-area)
IMS sends the chained MFS MID name if MFS is being used. Otherwise, no value is sent.

RRESOURCE(data-area)
IMS sends the value set by the MFS MOD if MFS is being used. Otherwise, no value is sent.

Initial state of back-end transaction

The back-end transaction is initiated in receive state, and should issue RECEIVE as its first command or after EXTRACT ATTACH.

The conversation

The conversation between the front-end and the back-end transactions is held using the usual SEND, RECEIVE, and CONVERSE commands. Details of these commands are given in the *CICS/MVS Application Programmer's Reference* manual.

In each of these commands, you must name the session in the SESSION option unless the conversation is with the principal facility.

Deferred transmission

On ISC sessions, when you issue a SEND command, CICS will normally defer sending the data until it becomes clear what your further intentions are. This mechanism enables CICS to avoid unnecessary flows by adding, or "piggy-backing", control indicators on the data that is awaiting transmission.

In general, IMS does not accept indicators such as change-direction, syncpoint-request, or end-bracket as stand-alone transmissions on null RUs. You should therefore always allow deferred transmission to operate, and avoid using the WAIT option or the WAIT TERMINAL command to force transmissions to take place.

Using the LAST option

The LAST option on the SEND command indicates the end of the conversation. No further data flows can occur on the session, and the next action must be to free the session. However, the session can still carry CICS syncpointing flows before it is freed.

The LAST option and syncpoint flows

A syncpoint on an ISC session is initiated explicitly by a SYNCPOINT command, or implicitly by a RETURN command.

If your conversation has been terminated by a SEND LAST command, without the WAIT option, transmission will have been deferred, and the syncpointing activity will cause the final transmission to occur with a piggy-backed syncpoint request. The conversation will thus be automatically involved in the syncpoint.

Freeing the session

The command used to free the session has the following format:

```
FREE SESSION(name)
```

where "name" is the name of the conversation.

You must free the session after issuing a SEND LAST command, or when the EIBFREE field has been set.

CICS allows you to issue the FREE command at any time that your transaction is in send state. CICS determines whether the end-bracket indicator has already been transmitted, and transmits it if necessary before freeing the session. If there is also deferred data to transmit, the end-bracket indicator is transmitted with the data. Otherwise, the indicator is transmitted by itself.

Because only certain IMS input components accept a "stand-alone" end-bracket indicator, this use of FREE is not recommended for CICS-to-IMS communication.

The EXEC interface block (EIB)

Full details of the EIB are given in the *CICS/MVS Application Programmer's Reference* manual. This section highlights the fields that are of particular significance in ISC applications. For further details of how and when these fields should be tested, or saved, see "Command sequences for CICS-to-IMS sessions" on page 281.

Conversation identifier fields

The following EIB fields enable you to obtain the name of the ISC session.

EIBTRMID
contains the name of the principal facility. For a back-end transaction, or for a front-end transaction started by ATI, it is the conversation identifier (SESSION). You must acquire this name if you want to state the session name of the principal facility explicitly.

EIBRSRCE
contains the session identifier (SESSION) for the session obtained by means of an ALLOCATE statement. You must acquire this name immediately after issuing the ALLOCATE statement.

Procedural fields

These fields contain information on the state of the session. In most cases, the settings relate to the session named in the last-executed RECEIVE or CONVERSE command, and should be tested, or saved for later testing, after the command has been issued. Further information on the use of these fields is given in "Command sequences for CICS-to-IMS sessions" on page 281.

EIBRECV
indicates the conversation state following RECEIVE or CONVERSE. If it is *off* (= X'00'), your conversation partner is inviting you to send, otherwise you would normally issue a further RECEIVE command. It does not necessarily reflect *receive state* at any other time.

EIBCOMPL
This field is used in conjunction with the RECEIVE NOTRUNCATE command; it is set to indicate that the data is complete.

EIBSYNC
indicates that CICS syncpointing is in progress and that the application must issue a SYNCPOINT command.

EIBSIG
indicates that the conversation partner has issued an ISSUE SIGNAL command.

EIBFREE
indicates that the receiver must issue a FREE command for the session.

Informatory fields

The following fields contain information about FMHs received from the remote transaction:

EIBATT
indicates that the data received contained an attach header. The attach header is not passed to your application program; however, EIBATT indicates that an EXTRACT ATTACH command is appropriate.

EIBFMH
indicates that the data passed to your application program contains a concatenated FMH.

If you want to use these facilities, you must ensure that you use communication profiles that specify INBFMH(ALL). The default profile (DFHCICSA) for a session allocated by a CICS front-end transaction has this specification. However, the default principal facility profile (DFHCICST) for a CICS back-end transaction does not. Further information on this subject is given under "Defining communication profiles" on page 151.

Command sequences for CICS-to-IMS sessions

The command sequences that you use to communicate between the front-end and the back-end transactions are governed both by the requirements of your application and by a set of high-level protocols designed to ensure that commands are not issued in inappropriate circumstances.

The protocols presented in this section do not cover all possible command sequences. However, by following them, you will ensure that each transaction takes account of the requirements of the other, and hence reduce the error rate during program development.

The protocols are based on the concept of a number of states. These states apply only to the particular conversation, not to your application program as a whole. In each state, there are a number of commands that might most reasonably be issued. After the command has been issued, fields in the EIB must be tested in the order shown in the state diagrams, Figure 98 on page 282 through Figure 104 on page 285, to check on the current requirements of the conversation. The results of these tests, together with the command that has been issued, may cause a transition to another state, in which another set of commands become appropriate.

The states that are defined for the purposes of this section are:

- State 1 — Session not allocated
- State 2 — Send state
- State 3 — Receive pending after INVITE
- State 4 — Receive state
- State 5 — Receiver take syncpoint
- State 6 — Free pending after SEND LAST
- State 7 — Free session.

Initial states

Normally, the front-end transaction in a conversation will initially be in state 1 — session not allocated — and must issue an ALLOCATE command to acquire a session.

An exception to this occurs when the front-end transaction is started by automatic transaction initiation (ATI) , in the local system, with an LUTYPE6.1 session as its principal facility. In this case, the session is already allocated, and the transaction in is state 2. For transactions of this type, you must immediately obtain the session name from EIBTRMID if you want to be able to name the session explicitly on subsequent commands.

You must always assume that the back-end transaction is initially in state 4 (receive state). Even if it is designed only to send data to the front-end transaction, you must issue a RECEIVE to receive the SEND INVITE issued by the front-end transaction and get into send state.

State diagrams

The following diagrams are intended to enable you to construct valid command sequences. Each diagram relates to one particular state, as previously defined, and shows the commands that you might reasonably issue and the tests that you should make after issuing the command. Where more than one test is shown, they must be made in the order indicated.

The combination of the command issued and a particular positive test result lead to a resultant state, shown in the final column.

Your program should not make assumptions as to the current or next state of the conversation. Always test the EIB flags to determine the current state of the conversation and what to do next.

See "Using the state diagrams" on page 205 for an example of how to use the state diagrams.

Other tests

The tests that are shown in the diagrams are those that are significant to the state of the conversation. Tests for other conditions that may possibly arise, for example, INVREQ or NOTALLOC, should be made in the normal way.

STATE 1 CICS to IMS CONVERSATIONS	SESSION NOT ALLOCATED	
Commands You Can Issue	What To Test	New State
ALLOCATE [NOQUEUE] *	SYSIDERR	1
	SYSBUSY *	1
	Otherwise (obtain session name from EIBRSRCE)	2

* If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command and do not code a HANDLE command for the SYSBUSY condition.

If you want control to be returned to your program if a session is not immediately available, either specify NOQUEUE on the ALLOCATE command and test EIBRCODE for SYSBUSY (X'D3'), or code a HANDLE CONDITION SYSBUSY command.

Figure 98. State 1 — session not allocated

STATE 2 CICS to IMS CONVERSATIONS		SEND STATE
Commands You Can Issue *	What To Test	New State
SEND		2
SEND INVITE	–	3 or 4
SEND LAST	–	6
CONVERSE Equivalent to: SEND INVITE WAIT RECEIVE	Go to the STATE 4 table and make the tests shown for the RECEIVE command	–
RECEIVE	Go to the STATE 4 table and make the tests shown for the RECEIVE command	–
SYNCPOINT	(transaction will ABEND if SYNCPOINT fails)	2
FREE Equivalent to: SEND LAST WAIT FREE	–	1
* For the front-end transaction, the first command used after the session has been allocated must be a SEND command or CONVERSE command that initiates the back-end transaction in one of the ways described in 'Attaching the Remote Transaction' above.		

Figure 99. State 2 — send state

STATE 3 CICS to IMS CONVERSATIONS		RECEIVE PENDING AFTER INVITE
Commands You Can Issue	What To Test	New State
SYNCPOINT	(transaction will ABEND if SYNCPOINT fails)	4

Figure 100. State 3 — receive pending after INVITE

STATE 4 CICS to IMS CONVERSATIONS		RECEIVE STATE
Commands You Can Issue	What To Test	New State
RECEIVE [NOTRUNCATE] *	EIBCOMPL *	–
	EIBSYNC	5
	EIBFREE	7
	EIBRECV	4
	Otherwise	2
* If NOTRUNCATE is specified, a zero value in EIBCOMPL indicates that the data passed to the application by CICS is incomplete (because, for example, the data-area specified in the RECEIVE command is too small). CICS will save the remaining data for retrieval by subsequent RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back. If the NOTRUNCATE option is not specified, overlength data is indicated by the LENGERR condition, and the remaining data is discarded by CICS.		

Figure 101. State 4 – receive state

STATE 5 CICS to IMS CONVERSATIONS		RECEIVER TAKE SYNCPOINT
Commands You Can Issue	What To Test	New State
SYNCPOINT	EIBFREE (saved value)	7
	EIBRECV (saved value)	4
	Otherwise	2

Figure 102. State 5 – receiver take syncpoint

STATE 6 CICS to IMS CONVERSATIONS		FREE PENDING AFTER SEND LAST
Commands You Can Issue	What To Test	New State
SYNCPOINT	–	7
FREE	–	1

Figure 103. State 6 – free pending after SEND LAST

STATE 7 CICS to IMS CONVERSATIONS		FREE SESSION
Commands You Can Issue	What To Test	New State
FREE	–	1

Figure 104. State 7 — free session

Part 5. Recovery and restart

This part tells you what CICS can do when things go wrong in an intercommunication environment, and what you can do to help.

"Chapter 5.1. Recovery and restart in interconnected systems" deals with individual system failure, and with system failure and restart.

"Chapter 5.2. Intercommunication and XRF" discusses those aspects of the CICS extended recovery facility (XRF) that affect intercommunication.

Chapter 5.1. Recovery and restart in interconnected systems

This chapter describes those aspects of CICS recovery and restart that apply particularly in the intercommunication environment. It is assumed that you are familiar with the concepts of logical units of work (LUWs), synchronization points (syncpoints), dynamic transaction backout, and other topics related to recovery and restart in a single CICS system. These topics are presented in detail in the *CICS/MVS Recovery and Restart Guide*.

In the intercommunication environment, most of the single-system concepts remain unchanged. Each system has its own system and dynamic logs (or the equivalent for non-CICS systems), and is normally capable of either committing or backing out changes that it makes to its own recoverable resources.

In the intercommunication environment, however, a logical unit of work can include actions that are to be taken by two or more connected systems. This means that the participating systems must reach mutual agreement to commit the changes they have made which, in turn, means that they must exchange syncpoint requests and responses over the intersystem sessions. This requirement represents the major single difference between recovery in single and multiple systems.

Terminology

The task that initiates the syncpoint activity is called the **initiator**. All other tasks in the syncpoint sequence receive syncpoint requests from the initiator and are called **agents**.

Syncpoint exchanges

Consider the following example:

Syncpoint example

An order-entry transaction is designed so that, when an order for a particular item is entered from a terminal, (1) an inventory file is queried and decremented by the order quantity, (2) an order for dispatch of the goods is written to an intrapartition transient data queue, and (3) a synchronization point is taken to indicate the end of the current LUW.

In a single CICS system, the syncpoint will cause both (1) and (2) to be committed.

The same result is required if the inventory file is owned by a remote system and is accessed by means of, say, CICS function shipping. This is achieved in the following way:

1. When the local transaction issues the syncpoint request, CICS sends a syncpoint request to the remote transaction (in this case, the CICS mirror transaction).
2. The remote transaction commits the change to the inventory file and sends a positive response to the local CICS system.
3. CICS commits the change to the transient data queue.

During the period between the sending of the syncpoint request to the remote system and the receipt of the reply, the local system does not know whether the remote system has committed the change. This period is known as the **indoubt** period, as illustrated in Figure 105 on page 293.

If the intersystem session fails before the indoubt period is reached, both sides will back out in the normal way. After this period, both sides will have committed their changes. If, however, the intersystem session fails during the indoubt period, the local CICS system cannot tell whether the remote system committed or backed out its changes. The local system performs backout according to the INDOUBT attribute of the local transaction (see page 291); this action may be inconsistent with the action taken by the partner system.

For LUTYPE6.2 sessions, CICS reduces the risk explained in the example by attempting resynchronization on a separate session (see "Action following failure during the indoubt period" on page 295). For LUTYPE6.1 sessions, and also for LUTYPE6.2 sessions if the resynchronization attempt fails, there are three possible courses of action that an application can take:

1. Commit the changes unilaterally.
2. Back out the changes unilaterally.
3. Neither commit nor back out the changes, but wait until the session is reestablished and attempt resynchronization.

The INDOUBT attribute of the transaction definition

You can control, in some part, the action that CICS takes after failure during the indoubt period by specifying transaction backout attributes when you define the transaction. This is done by means of the INDOUBT operand of the CEDA DEFINE and ALTER TRANSACTION commands, or the DTB operand of the DFHPCT resource definition macro. The INDOUBT attribute of a transaction is honored when communication is lost with a partner and the task is in the indoubt period. This may occur at the time of a session failure (for example, agent's system failure), or during initiator's system emergency restart (for example, initiator's system failure).

MRO and LUTYPE6.1 restrict support of the INDOUBT attribute to the initiator, because they rely on not-last[3] agents committing if they are in-doubt. LUTYPE6.2 supports the INDOUBT attribute for the initiator and not-last agents.

The INDOUBT operand has the following format:

INDOUBT({<u>BACKOUT</u>|COMMIT|WAIT})

and the parameters have the following meanings:

BACKOUT
specifies that the transaction is to be backed out if a failure occurs during the indoubt period. This is the default value.

Transaction backout is always performed for failures that occur outside the indoubt period, irrespective of what is specified in the INDOUBT operand. Because a dynamic transaction backout buffer is not acquired until a protected resource is modified, the transaction backout overhead for a transaction that never modifies a protected resource is negligible.

COMMIT
specifies that the changes made by the transaction are to be committed when failure occurs during the indoubt period. A typical situation in which COMMIT is appropriate is when transaction backout can cause data to be lost, but a unilateral commit can result only in duplicated data.

WAIT
(LUTYPE6.1 and LUTYPE6.2 only) specifies that, if the session fails during the indoubt period:

- Changes to recoverable temporary storage are to be neither committed nor backed out.
- Transactions started by an interval control START PROTECT request are neither committed or backed out.
- Changes to other recoverable resources are backed out.

[3] A CICS internal algorithm determines the order in which syncpoint requests are issued. An optimized syncpoint protocol is used for the last session in which a syncpoint request is sent.

The changes to recoverable temporary storage are **locked** until the session is recovered. A comparison is then made with the remote system and the locked resources are either committed or backed out to coordinate with it.

This option is honored only in the following circumstances:

- The only changes to recoverable resources are local
- The changes are made by either WRITEQ TS (without REWRITE) or START PROTECT, and
- Only one connection with a partner exists for the local transaction at the time of the syncpoint.

In other circumstances, normal backout is provided.

If session recovery is unsuccessful, the START commands are canceled but temporary storage queue changes are committed.

Macro equivalents of INDOUBT attributes: If you are using DFHPCT macros, rather than RDO, to define your transactions, you can specify your transaction-backout requirements in the DTB operand.

DTB=NO has no equivalent in RDO

DTB=YES is equivalent to INDOUBT(BACKOUT)

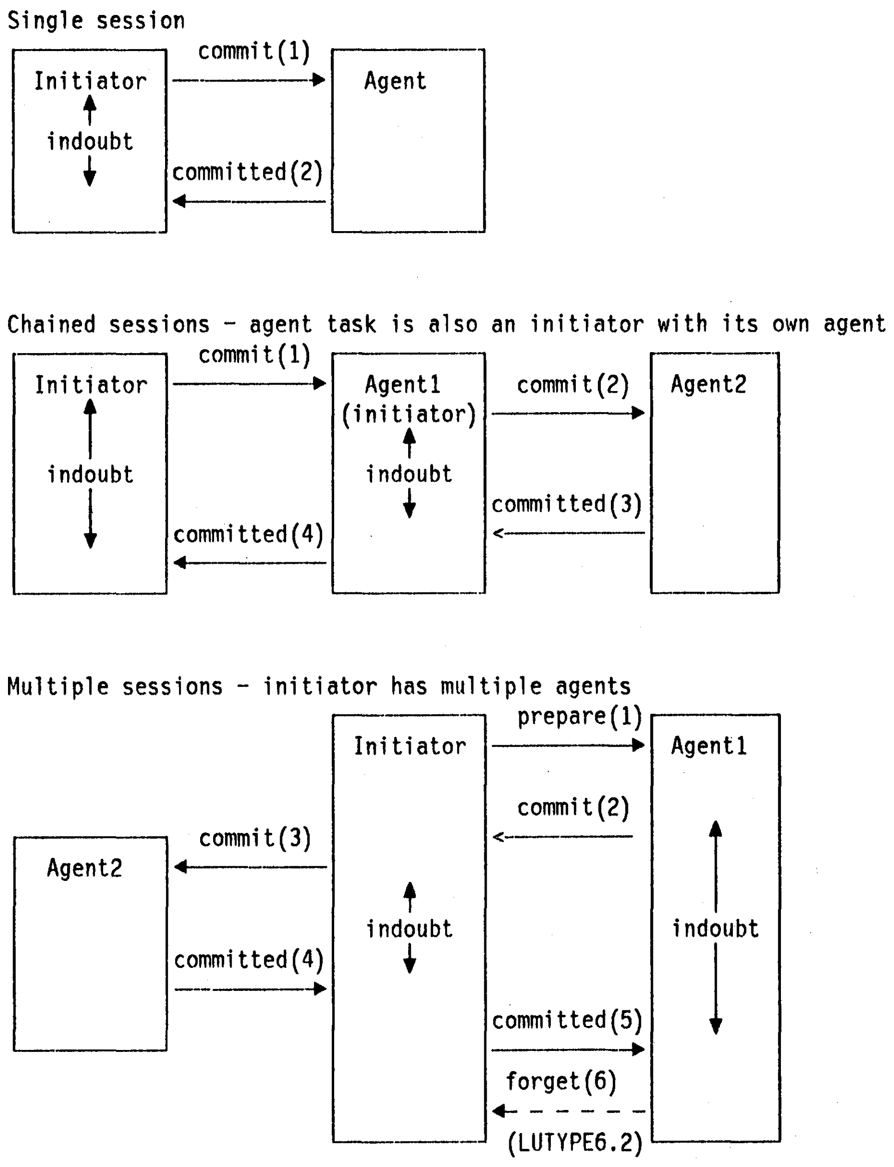
DTB=(YES,NO) is equivalent to INDOUBT(COMMIT)

DTB=(YES,WAIT) is equivalent to INDOUBT(WAIT)

Syncpoint flows

The ways in which syncpoint requests and responses are exchanged on intersystem conversations are defined in the LUTYPE6.1 and LUTYPE6.2 architectures. CICS multiregion operation uses the LUTYPE6.1 protocols. Although the formats of syncpoint flows for LUTYPE6.1 and LUTYPE6.2 differ, the concepts of syncpoint exchanges are similar.

The flows involved in syncpoint exchanges are illustrated in Figure 105 on page 293. In CICS, all of these flows are generated automatically in response to explicit or implicit SYNCPOINT commands issued by a transaction. However, a basic understanding of the flows that are involved can assist you in the design of your application and give you an appreciation of the consequences of session or system failure during the syncpoint activity. For more information about these flows, see "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171.



The numbers in brackets, for example, (1), show the sequence of the actions in each flow.

A. Initiator must be in send state on all its sessions.
B. When they issue their syncpoint, agents must be in send state on all sessions except the session on which they receive, the prepare, or commit.

Figure 105. Syncpointing flows

In the simplest case, the initiator has a single conversation with an agent that has no conversations other than with the initiator. At the start of the syncpoint activity, the initiator sends a **commit** request to the agent. The agent commits its changes and responds with **committed**. The initiator then commits its changes, and the logical unit of work is complete.

If the agent transaction also has a conversation with a third transaction, it must itself initiate syncpoint activity on this latter conversation before it responds to its initiator. The third transaction commits first, then the agent transaction, and finally the initiator transaction.

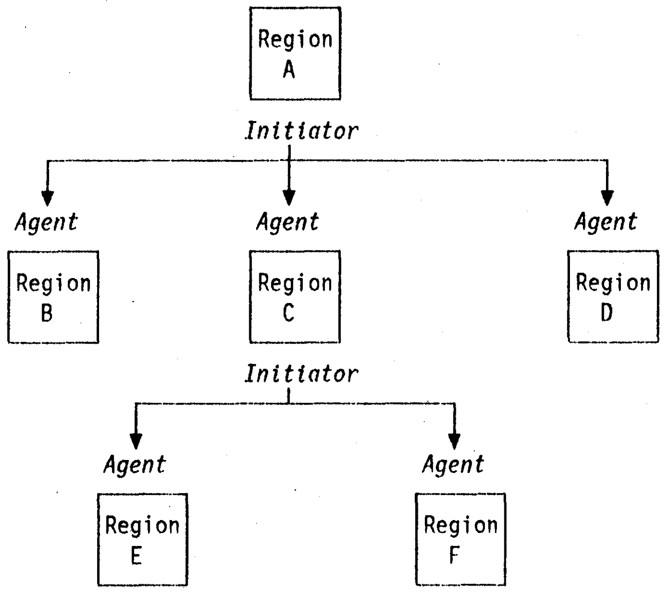
In the more general case, the initiator transaction can have more than one agent, and must inform each of them that a syncpoint is being taken. It does this by sending a 'prepare' request to all of its agents except one. (The order in which this is done and the identity of the "last" agent are not defined.) An agent that receives a 'prepare' request responds by sending a 'commit' request back to the initiator.

When all these 'prepare' requests have been sent and all the 'commit' responses received, the initiator sends a 'commit' request to its "last" agent. When this responds with a 'committed' indication, the initiator then sends 'committed' requests to all the other agents. For LUTYPE6.2 conversations only, these agents respond 'forget' to show that they do not require resynchronization.

Initiator—agent relationship

The diagram below shows the relationship of the initiator and agent regions. Region A contains the initiator task for a syncpoint sequence, regions B, C, and D contain the agent tasks for the same sequence.

Note that this relationship exists only for the duration of syncpoint processing for a single LUW. In distributed transaction processing, the same tasks might process a subsequent unit of work in which a different task is the initiator.



The lower part of the diagram shows a more complicated situation that can exist. As well as communicating with tasks in regions A, B, and D, the task in region C may also be communicating with tasks in regions E and F.

Before responding to a syncpoint request from the task in A, the task in C must first initiate synchronization of its processing with the tasks in E and F. The task in C thus becomes the initiator in its relationship with the tasks in E and F. How C responds to A depends on the outcome of the syncpoint processing for C, E, and F. If C loses contact with E or F during the indoubt period, then C honors its INDOUBT attribute.

The flows between C, E, and F, correspond to those between A, B, C, and D, the only difference being that there is one fewer agent.

Failures in connected systems

The failures that can occur in connected systems are:

1. Session failures, either between the CICS systems or between CICS and the terminal associated with a transaction. These failures cause transaction abends of the transaction or transactions connected to the session and resource recovery is performed as for non-connected transaction abends.
2. Total CICS system failures. The failing system is recovered using emergency restart as for a non-connected system though there are extra features, described following, added for intersystem communication. Any remote system connected at the time of the system failure sees the failure as a session failure and treats it as such. Thus, a remote system failure causes a local transaction abend.
3. Transaction abends. These are recovered using dynamic transaction backout as for non-connected systems. The mirror and relay transactions are not special in this respect and, to make full use of CICS recovery, you should specify dynamic transaction backout for both.

   The transaction restart facility can also be used. You cannot specify this for the mirror or relay transaction, but, if you specify it for the associated user-written transaction, the mirror or relay transaction will be restarted.

Action following failure during the indoubt period

This section discusses CICS actions following failure during the indoubt period under the following headings:

- "LUTYPE6.2 connections"
- "LUTYPE6.1 connections"
- "MRO connections"
- "Messages that can help recovery"
- "Restoration of data integrity."

LUTYPE6.2 connections

When an LUTYPE6.2 session fails during the indoubt period, CICS tries immediately to contact the partner system using a different session. If this is successful, CICS completes the syncpoint by comparing unit-of-recovery descriptors (URDs)[4] to decide whether to commit or back out the resources that are in doubt. The failed session can no longer be used by this task, having been freed by the system. If either transaction issues a command for the failed conversation, the result is a 'termerr' condition, which, if not expressly handled, leads to an abend and (if necessary) back out, but with no loss of synchronization.

Immediate recovery is not always possible, for example when all sessions are out of service because of a serious failure of one of the systems or the connecting hardware. In this case, the system must abend the transaction and obey the INDOUBT option. It must then wait until the connection has been reestablished, possibly after emergency restart, before it can determine and report the state of the distributed logical unit of work.

When the intersystem session is recovered, URDs are compared by the two systems to find out and report (to the CSMT log) whether the unilateral actions taken by the systems matched or not, and commit or backout any temporary storage changes locked due to the INDOUBT(WAIT) option.

LUTYPE6.1 connections

The absence of an LU services manager makes immediate recovery impossible. Recovery is possible only after sessions have been reestablished. When the sessions are reestablished, the two sides exchange message sequence numbers to determine (and report to the CSMT log) whether the actions taken by the systems matched, and to decide whether to commit or back out temporary storage changes locked due to the INDOUBT(WAIT) option.

MRO connections

The INDOUBT(WAIT) option is not available for MRO conversations. Consequently, at the time of failure, the transactions are either committed or backed out, according to whether INDOUBT(COMMIT) or INDOUBT(BACKOUT) is specified in the PCT.

Messages that can help recovery

The messages associated with intersystem session failure and recovery are shown in two figures. Figure 106 on page 297 shows the messages associated with the INDOUBT(BACKOUT or COMMIT) attributes. Figure 107 on page 297 shows the messages associated with the INDOUBT(WAIT) attribute. Full details are in the *CICS/MVS Messages and Codes* manual.

[4] A URD is a CICS control block that describes the progress of a unit of work through the sequence of syncpoint messages. It is recovered at CICS restart.

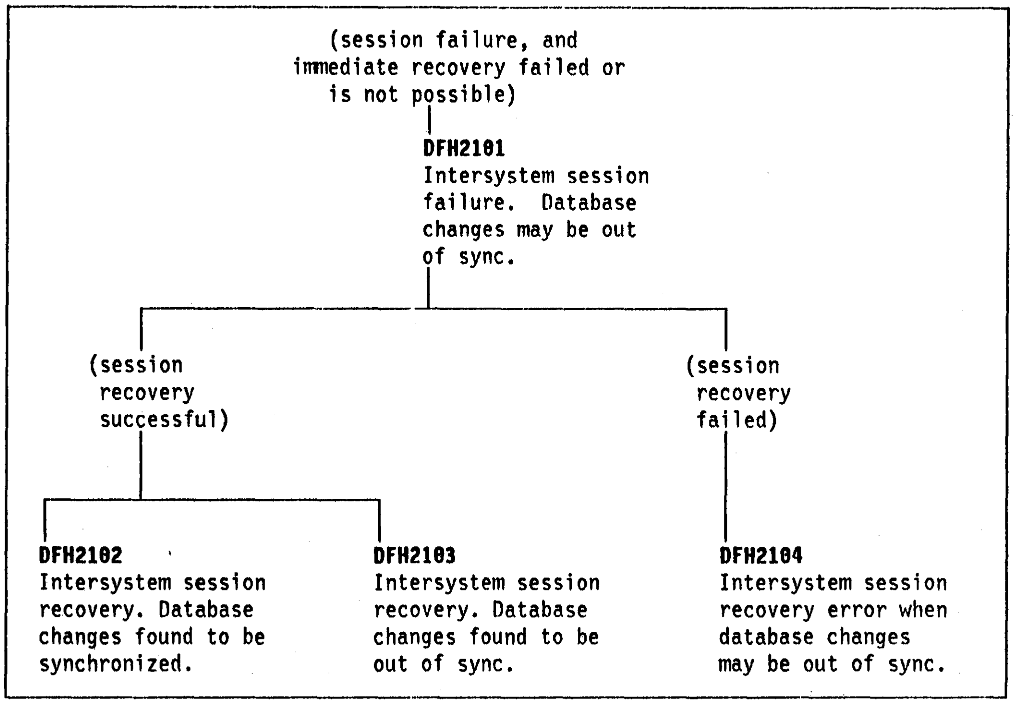

(session failure, and immediate recovery failed or is not possible)
DFH2101
Intersystem session failure. Database changes may be out of sync.
(session recovery successful)
(session recovery failed)
DFH2102
Intersystem session recovery. Database changes found to be synchronized.
DFH2103
Intersystem session recovery. Database changes found to be out of sync.
DFH2104
Intersystem session recovery error when database changes may be out of sync.

Figure 106. Session failure messages for INDOUBT(BACKOUT or COMMIT)

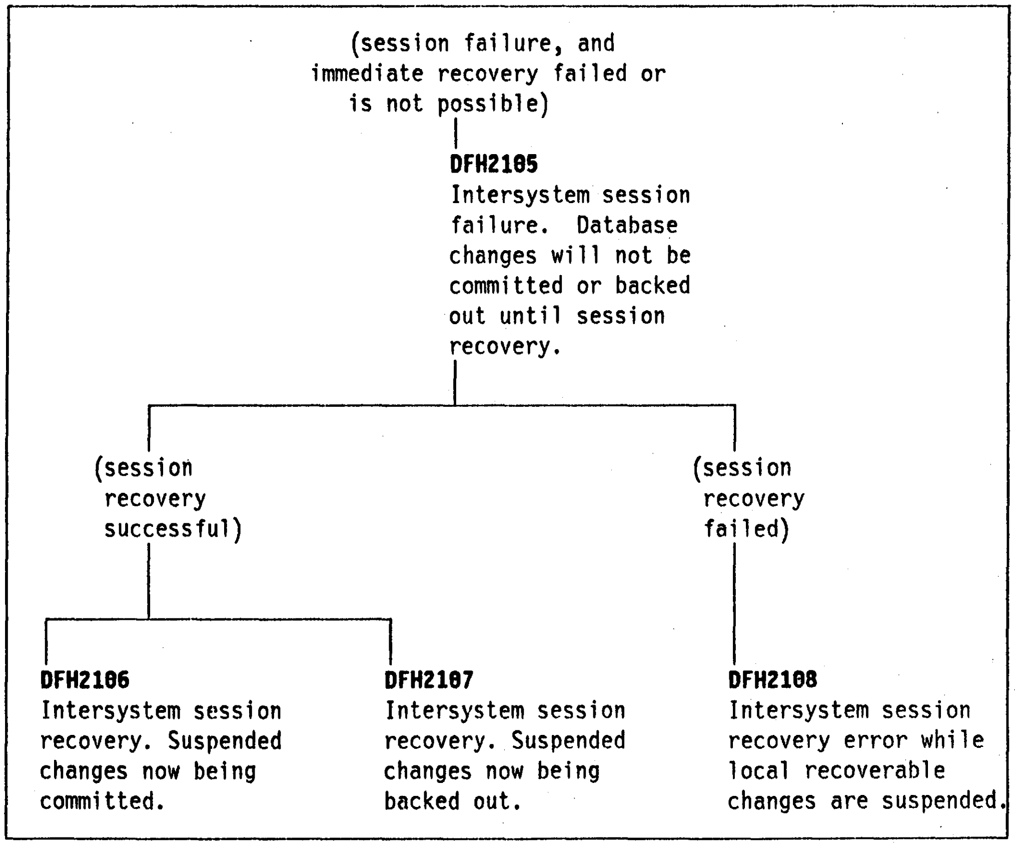

(session failure, and immediate recovery failed or is not possible)
DFH2105
Intersystem session failure. Database changes will not be committed or backed out until session recovery.
(session recovery successful)
(session recovery failed)
DFH2106
Intersystem session recovery. Suspended changes now being committed.
DFH2107
Intersystem session recovery. Suspended changes now being backed out.
DFH2108
Intersystem session recovery error while local recoverable changes are suspended.

Figure 107. Session failure messages for INDOUBT(WAIT)

All these messages contain the following information, which enables the messages to be correlated:

- The time
- The transaction identifier and task number
- The remote system identifier
- The intersystem terminal identifier (the session name)
- The operator identifier
- The operator terminal identifier
- The unit-of-work identifier.

Because the partner region may have resolved the logical unit of work (LUW) differently, a region issues message DFH2101 when it loses communication with a partner. The message may appear at the time of a session failure or partner region failure, or during emergency restart.

When the connection has been reestablished, the state of the LUW is determined, and a DFH2102, DFH2103, or DFH2104 message is issued for each session. For MRO and LUTYPE6.1, these messages appear only on the initiator side.

If an agent region successfully commits but a session failure occurs before the initiator receives confirmation of this, the region does not issue a DFH2101 message. After session recovery, a DFH2102, DFH2103, or DFH2104 message may be issued by the agent region.

As the following example shows, the system or application design can mean that there is no exposure to data integrity even though DFH2103 or DFH2104 has been issued.

Example

An order entry transaction is designed to update a recoverable file which is defined on a remote CICS region. The update is achieved using a function shipped file control request and there are no other recoverable resources involved in the transaction.

If the intersystems session fails during the indoubt period, the local CICS region reports the possible integrity exposure with DFH2101 and takes unilateral action to commit or backout the LUW. Because there are no recoverable changes in this region, there can be no loss of synchronization with the remote file.

After session recovery, CICS may issue DFH2102 or DFH2103. You would ignore this if data integrity were your only concern. However, if the message shows that the units of work are out of step, this could still be significant. For instance, it would tell a terminal operator whether the order entered last was registered or not.

Restoration of data integrity

If CICS messages indicate that database changes are or may be out of sync, restoration of data integrity is made possible by the inclusion of the UOWID in all in-doubt messages and in a logged correlation record for each agent that has updated a recoverable resource. A user-written log-scanning utility can read all log records for the UOW in the affected CICS regions, and determine what action is needed to synchronize the databases.

Recovery for LUTYPE6.2 connections

This section describes recovery for LUTYPE6.2 connections under the following headings:

- "Exchange-lognames process"
- "Pending units of work" on page 300
- "Connected system recovery — an example" on page 301.

Exchange-lognames process

When a CICS system is restarted, operational constraints can cause a new or different system log to be used. If the restarted system has been communicating with a partner that is waiting to perform session recovery, the recovery process is corrupted. The exchange-lognames process detects this situation and is performed whenever a connection is established.

The exchange-lognames process is a defined piece of the LUTYPE6.2 architecture. For a full description of the concepts and physical flows, see the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*.

Exchange Lognames — an example

A networking failure occurs during the indoubt period of a transaction causing a failure of the connection between two CICS systems. INDOUBT(WAIT) is coded in the PCT for both transactions. One CICS system is shut down and cold-started. The partner system remains active and, when the connection is reestablished, the exchange-lognames process detects the cold start. CICS treats this as an operational error, and will not attempt session recovery. The master operator is made aware of the exchange-lognames failure by console messages issued by CICS. Alternatively, the CEMT INQUIRE CONNECTION command can be used to determine whether a failure has occurred.

The exchange-lognames process alerts the master operator if the logs (used to restore URDs) are not the same ones that were in use at the preceding failure.

The exchange-lognames process affects only SL(2) conversations. If it fails, SL(2) conversations are not allowed on the link until the failure is resolved. This resolution can be achieved only by operator action. However, SL(0) and SL(1) traffic on the link are unaffected by the failure, and continue as normal.

The CEMT INQ CONNECTION command can be used to determine whether the exchange-lognames process has completed successfully. The status is shown as 'XOK' if the process was successful. If the connection status is shown as "XNOtdone" (exchange lognames not done) and "ACQuired", CICS will not allow any synclevel 2 conversations. This may (depending on the design of your system) effectively mean that the connection is not available to applications.

One or more of the following messages will appear on the CSMT log:

DFH2110	**ABNORMAL REPLY TO EXCHANGE LOG NAME COMMAND SENT TO SYSTEM:** *sysid*
DFH2111	**COLD/WARM RESTART MISMATCH WITH SYSTEM** *sysid*
DFH2112	**LOG NAME MISMATCH WITH SYSTEM** *sysid*. **EXPECTED LUNAME.LOGNAME** *logname* **RECEIVED LUNAME.LOGNAME** *logname*

In these messages the term 'warm' means that a connection has previously been established with the partner system, and the lognames have been exchanged and saved. A system is 'cold' if the logname has not been exchanged with the partner, or if the memory of it has been erased. The memory can be erased by:

- Cold start of the CICS system.
- The CEMT SET CONNECTION(...) NOTPENDING command

Note, however, that the CEMT SET CONNECTION(...) NOTPENDING command deletes any outstanding resynchronization data for the connection. Depending on what information is present, this could lead to integrity problems. Issue this command with care.

The *CICS/MVS Messages and Codes* manual gives possible actions to correct the various conditions without damaging data integrity; these involve restarting one or both CICS systems with the correct logs. Note, however, that the "XNOtdone" status also means that at least one end of the connection has pending URDs. The next section, "Pending units of work," explains a way to resolve the situation without restarting, and describes the factors that determine the best action to take.

For more detailed diagnosis-type information on the exchange-logname process, see the *CICS/MVS Diagnosis Reference* manual.

Pending units of work

When an LUTYPE6.2 session failure leaves a URD requiring session recovery, CICS sets the connection status to "pending". After successful session recovery, this status is removed. The CEMT INQ CONNECTION command can be used to discover whether there are any pending URDs for the named system.

You should determine whether data integrity in your system is dependent on successful session recovery. You may find that intersystems communication does not involve recoverable resources, or that recoverable resources reside on only one of the communicating systems. In these cases, communication failure does not affect resource integrity. Alternatively, application design may provide a method other than session recovery to restore data integrity following a connection failure.

If you are dependent on session recovery and resynchronization and this has failed, you should carefully determine why. This situation indicates an abnormal operation such as an unscheduled cold start of a connected CICS system.

You can allow normal operation to continue by using the command CEMT SET CONNECTION(....) NOTPENDING to override the normal resynchronization process and put the CICS system into a state in which it is prepared to accept any log name chosen by the remote system. **This action prevents CICS from determining whether or not a loss of integrity actually occurred.**

CEMT SET CONNECTION(....) NOTPENDING has the following effects.

- A connection can immediately be acquired with the remote system.
- If the connection is already acquired, the exchange-lognames process is successfully executed and synclevel 2 conversations are permitted.
- The connection is set to 'notpending' status.
- The normal resynchronization process is overridden by the deletion of all information (URDs) describing the unresolved units of work.
- The URD containing the logname of the partner system is erased; the connection is in the same state as if the CICS system had been cold started.
- If INDOUBT(WAIT) has been specified and temporary storage changes are suspended (DFH2105 has been issued), the changes are unilaterally committed. You can determine the status of data integrity and restore it as described in "Restoration of data integrity" on page 299.

In summary, you can resolve the "pending" and "XNOtdone" status by

- Restarting both CICS systems using the correct logs
- Issuing the CEMT SET CONNECTION NOTPENDING command on whichever system displays the "pending" status.

Which of these actions is best for your installation depends on your requirements for availability and data integrity, and on your own procedures for recovering data integrity.

Connected system recovery — an example

As an illustration of connected system recovery design, consider the following simple example:

> A transaction is given a part number; it checks the entry in a local file to see whether the part is in stock, decrements the quantity in stock and updates the stock file, and sends a record to a remote transient data queue to initiate the dispatch of the part.

It is assumed that function request shipping is used, which means that a mirror transaction runs in the remote system. However, the same principles would apply if DTP is being used and the remote transaction is user-written.

Ideally, the update to the local file should take place only if the addition is made to the remote transient data (TD) queue, and the TD queue should only be updated if an addition is made to the local file. The first step towards achieving this is to specify both the file and the TD queue as recoverable resources and to specify INDOUBT(BACKOUT), or allow it to default, on the definitions of the local transaction and the mirror transaction in the remote system. This ensures synchronization of the changes to the resources (that is, both changes will either be backed out or committed) in all cases except for an intersystem session failure during the indoubt period of syncpoint processing.

For failure during the indoubt period (and, for LUTYPE6.2 sessions, following a failure of the resynchronization attempt), the change made to the stock file is backed out, and message DFH2101, warning that the resources might be out of synchronization, is sent to the master terminal destination.

Under these conditions, the mirror transaction may, or may not, have backed out, and it is possible that the entry dispatching the part was added to the remote TD queue, but the stock file was not updated. Consequently there is a danger that the part might be dispatched elsewhere before the mismatch between the two resources can be corrected.

A more acceptable solution is to update the stock file even though there is a danger that the dispatch record has not been added to the TD queue, especially if the delayed dispatch can readily be reinitiated on session recovery. This can be achieved by specifying INDOUBT(COMMIT) for the local transaction.

When the session is eventually recovered, CICS checks whether the resources are in fact out of synchronization. If they are not, message DFH2102 is issued. Otherwise, DFH2103 is issued and a transaction to reconcile the mismatch should be run. In this case, the reconciliation process is simply to retransmit the dispatch record to the remote transient data queue. This could be implemented by the same application with special logic to inhibit local changes.

In general, the reconciliation process is a rerun of the original transaction with local changes inhibited if INDOUBT(COMMIT) is specified, or with remote changes inhibited if INDOUBT(BACKOUT) is specified.

Intersystem communication and emergency restart

If a partner system totally fails, to a conversation it appears as if only the connection had failed. The failed system will usually be emergency restarted, and so its local resources will be recovered in the normal way. Because there were connected systems, emergency restart restores these to the state they were in when the partner system failed.

LUTYPE6.1 message sequence numbers and LUTYPE6.2 unit-of-work states are both recovered from the system log, as well as sufficient information to take actions as for session failures. Consequently, recoverable resources are backed out, committed, or held, and the appropriate messages are issued. When the session is restored, normal resynchronization occurs.

Error handling programs for intercommunication

CICS intercommunication uses CICS terminal control facilities to exchange messages with connected systems. When an unrecoverable situation is detected in either CICS system, the exchange of messages is terminated by means of a special negative response. This special response will be sent to the CSMT destination by the receiving system. It is followed by a detailed error recovery message. The sense code in the error message will lead to abnormal termination of the transactions, so that CICS dynamic transaction backout processing can be invoked to guard against inconsistent resource updates.

For LUTYPE6.1 and LUTYPE6.2, the negative response received by CICS is handled by the node abnormal condition program (DFHZNAC) and passed to the user-supplied node error program (DFHZNEP) if present. The default actions set by CICS ensure that CICS reads in the following error message. The sense code in this message is made available to DFHZNAC and DFHZNEP in the same way as system sense codes carried by the LUSTATUS commands or negative responses. CICS default actions based on this system sense code are set by DFHZNAC, before making the code available to DFHZNEP. Error conditions occurring on intersystem communication sessions are therefore handled exactly like errors on other SNA sessions through VTAM.

It is not necessary to write a node error program to handle intersystem communication sessions, because the default actions set by DFHZNAC have been selected to enforce correct recovery based on the error condition detected. When the system sense code indicates that the original request to VTAM may be retried, CICS will do so transparently to the application program attempting to send a message.

For details of user-supplied DFHZNEP programs, see the *CICS/MVS Customization Guide*.

Database interlock

As a part of database and application design in a single CICS system, you must be careful not to design programs in such a way that two programs running concurrently can request the same records in such a way as to interlock on each others requests.

This problem continues to exist in interconnected systems where application programs in two different systems can cause transactions in a third system to interlock in a similar manner. Such an interlock will be detected by means of a timeout value specified for the PCT, which will expire when a program has waited the specified period without a reply from the deadlocked transaction. CICS will abend the task that has been waiting the longest, so breaking the interlock and allowing the contending task (or tasks) to continue.

Use of transaction chaining can lead to such a situation. Chaining also opens the possibility for a designer employing function request shipping or transaction routing (though not DTP) to define a specific resource (including a transaction or terminal) as being in a remote CICS system, and further define that resource in the remote system to be in yet another system. If the definition in the third system inadvertently specifies the resource to be in the first, any request for that resource will be routed to all three systems and will then deadlock until the specified timeout value expires, abending all the transactions. For these reasons great care should be taken during system definition to guard against unintended use or misuse of chained transactions.

Problem determination

Application programs that make use of CICS intercommunication facilities are liable to be subject to error conditions not experienced in single CICS systems. The new conditions result from the intercommunication component not being able to establish a session with the requested system (for example, it is not defined to CICS, or is not available).

In addition, some types of request may cause a transaction abend because invalid data is being passed to the CICS function manager (for instance the file control program). Where the resource is remote, the function manager is also remote, so the transaction abend is suffered by the remote transaction. This in turn causes the local transaction to be abended with a transaction abend code of ATNI (for communication via VTAM) or AZI6 (for communication via MRO) rather than the particular code used in abending the remote transaction. However, the remote system sends the local CICS system an error message identifying the reason for the remote failure. This message is sent to the local CSMT destination. Therefore, if an application program uses SETXIT and user task abend exits to continue processing when abends occur while accessing resources, it will be unable to do so in the same way when those resources are remote.

Application programs not using the command level or DL/I CALL interfaces may inadvertently attempt to access, via function request shipping, resources defined as remote. In this case, the request will fail with a condition indicating that the resource is not defined to CICS.

Trace and dump facilities will exist in both local and remote CICS systems. When the remote transaction is abended, its CICS transaction dump is available at the remote site to assist in locating the reason for an abend condition.

Applications to be used in conjunction with remote systems should be well tested to minimize the probability of failing when accessing remote resources. It should be remembered that a "remote test system" can actually reside in the same processor as the local system and so be tested in a single location where the transaction dumps from both systems, and the corresponding trace data, are readily available. The two transactions can be connected via MRO or via the VTAM application-to-application facility.

Detailed sequences and request formats for CICS intercommunication can be found in the *CICS/MVS Diagnosis Reference* manual and the *CICS/MVS Problem Determination Guide.*

Recovery and restart with non-CICS systems

The cross-link exchanges used by CICS to establish the state of the other system during recovery are defined by SNA. They are therefore independent of the nature of the remote system. CICS follows the same recovery procedures whether the other system is CICS or not.

Chapter 5.2. Intercommunication and XRF

The extended recovery facility (XRF) of CICS/MVS is described in the *CICS/MVS XRF Guide*. This chapter looks at those aspects of XRF that apply to ISC and MRO sessions. For more details of the link definitions mentioned in this chapter, refer to "Chapter 3.1. Defining links to remote systems" on page 91.

MRO and ISC sessions are not XRF-capable because they cannot have backup sessions to the alternate CICS system.

You can use the AUTOCONNECT operand in your link definitions to cause CICS to attempt to reestablish the sessions following a a takeover by the alternate CICS system.

Also, the bound/unbound status of some ISC session types can be tracked. In these cases, CICS can attempt to reacquire bound sessions irrespective of the AUTOCONNECT specification.

In all cases, the timing of the attempt to reestablish sessions is controlled by the AUTCONN operand of DFHSIT (the *CICS/MVS Resource Definition (Macro)* manual describes DFHSIT in detail).

MRO sessions: The status of MRO sessions cannot be tracked. Following a takeover by the alternate CICS system, CICS attempts to reestablish MRO sessions according to the value specified for the INSERVICE operand of the CONNECTION definition.

LUTYPE6.1 sessions: Following a takeover, CICS will attempt to reestablish LUTYPE6.1 sessions in either of the following cases:

1. The AUTOCONNECT operand of the SESSIONS definition specifies YES.
2. The sessions are being tracked, and are bound when the takeover occurs. The status of LUTYPE6.1 sessions is tracked unless RECOVOPTION(NONE) is specified in the SESSIONS definition[5].

Single-session APPC devices: Following a takeover, CICS will attempt to reestablish single LUTYPE6.2 sessions in either of the following cases:

1. The AUTOCONNECT operand of the SESSIONS or TYPETERM definition specifies YES.
2. The session is being tracked, and is bound when the active CICS fails. Single LUTYPE6.2 sessions are tracked unless RECOVOPTION(NONE) is specified in the SESSIONS or the TYPETERM definition (depending upon which form of definition is being used)[5].

[5] Although the RECOVOPTION has five possible values, for ISC there is effectively a choice between NONE (no tracking) and *any one* of the other options (tracking).

Parallel LUTYPE6.2 sessions: Following a takeover, CICS will attempt to reestablish the LU services manager sessions in either of the following cases:

1. The AUTOCONNECT operand of the CONNECTION definition specifies YES or ALL.
2. The sessions are being tracked, and are bound when the active CICS fails. Only the LU services manager sessions (SNASVCMG) can be tracked in this case; tracking is not available for user sessions.

Once the LU services manager sessions are reestablished, CICS will attempt to establish the sessions for any mode group that specifies autoconnect.

Effect on application programs: To application programs that are using the intercommunication facilities, a takeover in the remote CICS system is indistinguishable from a session failure.

Part 6. Link services

This part contains two chapters:

"Chapter 6.1. Security in the intercommunication environment" on page 311 describes security in the intercommunication environment. Before reading this chapter, you should be familiar with the contents of "Chapter 3.1. Defining links to remote systems."

"Chapter 6.2. Master terminal operations for LUTYPE6.2 connections" on page 329 gives special guidance on the management of LUTYPE6.2 links using the CICS-supplied transaction CEMT.

Chapter 6.1. Security in the intercommunication environment

This chapter discusses those aspects of security that apply to CICS intersystem communication and multiregion operation.

The security requirements of a CICS system that uses ISC or MRO to communicate with other, remote, systems are basically an extension of the security requirements of a single, stand-alone, CICS system. The *CICS/MVS Facilities and Planning Guide* describes how to plan for and implement security in a single CICS system, using either CICS security facilities or an external security manager such as the Resource Access Control Facility (RACF) of MVS.

References to RACF in this chapter refer to the Resource Access Control Facility of MVS or to any external security manager of equivalent function.

This chapter assumes that you are familiar with security planning and implementation for a single CICS system. In particular, you should understand the concepts of operator signon, how the relationship between operator security and transaction security determines which transactions a particular user is allowed to invoke, and how resource security determines which other resources a user is allowed to access.

```
REMOTE SYSTEM              SECURE CICS SYSTEM

 Remote User          Security Level             Enforced by

Bind Session ========> Bind-Time Security:
                        (BIND received)

                       1. Accept bind session    LU6.1  No check.
                          request?               LU6.2  Bind Password in link
                                                        definition.
                                                 MRO    Security Name in link
                                                        definition.
                       Bind-Time Security:
             <========= (BIND sent)

                       1. Is the remote system   LU6.1  No check.
                          the correct one?       LU6.2  Bind Password in link
                                                        definition.
                                                 MRO    No check.
Attach
Transaction =========> Transaction Security:

                       1. Has the session        Link security authority
                          authority to            established when session
                          attach transaction      is bound.  Authority obtained
                                                  through CICS sign-on table
                                                  or from link definition.

                       2. Has the remote user    LU6.1  No check.
                          authority to access    LU6.2  Authorized remote user
                          this system?                  defined in CICS
                                                        sign-on table.
                                                 MRO    As LU6.2.

                       3. Has the remote user    LU6.1  Link security authority
                          authority to           LU6.2  User security authority
                          attach transaction            established at this
                                                        attach request (or
                                                        possibly at earlier
                                                        attach request from
                                                        same user).  Authority
                                                        obtained through CICS
                                                        sign-on table.
                                                 MRO    As LU6.2.
```

Figure 108 (Part 1 of 2). Summary of intersystem and interregion security

Resource Security:	
1. Has the session authority to access other resources that the transaction uses?	Link security authority established when session is bound. Authority obtained from CICS sign-on table or from link definition.
2. Has the remote user authority to access other resources that the transaction uses?	LU6.1 Link security authority LU6.2 User security authority established at this attach request (or possibly at earlier attach request from same user). Authority obtained through CICS sign-on table. MRO As LU6.2.

Figure 108 (Part 2 of 2). Summary of intersystem and interregion security

Planning for intercommunication security

When you are planning a CICS system that will use ISC or MRO, you must extend your concept of a **user** of your system to include remote systems and also the "users" of those systems. Your security concerns for your CICS system will be directed primarily towards incoming requests for access to your resources, rather than with requests that you send to other systems.

The security problem with incoming requests is:

A particular user at a particular remote system is trying to access one or more of the resources of your CICS system. Is this access authorized, or should it be rejected?

The following sections describe the points in the processing of an incoming request at which you can apply security checks:

1. The first requirement is for a session to be established between the two systems. This does not, of course, happen on every request; a session, once established, is usually long-lived. Also, the connection request that establishes the session can, depending on the circumstances, be issued either by the remote system or by your CICS system. However, the initial establishment of a session presents the first potential security exposure for your system.

   Your security concern is to prevent the attachment of unauthorized remote systems to your CICS system; that is, to ensure that the remote system is **really** the system that it claims to be.

   This level of security is called **bind-time security**. You can specify bind-time security for MRO and LUTYPE6.2 links, but not for LUTYPE6.1 links.

Note: In this chapter, the term **bind** is used to refer both to the SNA BIND command that is used to establish SNA sessions between systems and to the CICS **connection request** that is used to establish MRO sessions for CICS interregion communication.

2. You also have to decide whether you want to limit the remote system's access to your transactions and resources. This level of security is called **link security**. It is concerned with the security profile that you assign to the remote system as a whole. Like operator security in a single-system environment, it governs:

   - **Transaction security** This controls the link's authorization to attach specific transactions.
   - **Resource security** This controls the link's authorization to access specific resources. Transactions that are initiated by EXEC CICS START are regarded as resources.

   There are two methods of defining link security. For one of these, you simply specify *preset* security keys in the resource definition for the link. For the other, you set up a signon table entry, which is used to sign on each session to the system at bind-time. An important point to remember is that link security values are established when a session is bound and remain in effect until the session is broken.

3. Within the bounds of the security profile that you set up for the link, you may want to restrict each remote user's access to the transactions and resources in your system. This is done by defining a signon table entry for each user, as you would for a single CICS system. This **user security**, like link security, distinguishes between transaction security and resource security.

   It is important to note that a user cannot access any transaction or resource that the link itself is not authorized to access. This means that each user's access keys must be included in those defined for the link. Also, if you do not expressly specify link security, CICS will sign the link on at the minimum security level, which will prevent you from applying any meaningful user security.

   You can specify user security for MRO and LUTYPE6.2 links, but not for LUTYPE6.1 links. For LUTYPE6.1 links, the user security is taken to be the same as the link security.

Implementing intercommunication security

Security in the intercommunication environment is implemented through **resource definition**. The following sections tell you how to define your intersystem links, your signon table, and all your protected resources such as transactions and databases. Once you have set it up, your security system functions quite independently and requires no intervention from remote terminal operators or application programs.

If you are using RACF, you will also need RACF profiles for your resources and users.

Bind-time security

Bind-time security is the CICS security that can be applied when a request to establish a session is received from or sent to a remote system.

From Figure 108 on page 312, you will see that LUTYPE6.2 links provide a password facility that can be used to prevent sessions from being established with any remote system that does not know the password. For MRO links, only incoming requests can be checked. No bind-time security can be applied to LUTYPE6.1 links.

Although link security values are established at bind-time, there is no need to apply bind-time security to make link security work. If there is no inherent bind-time security exposure in your network configuration, you may choose not to apply the bind-time checks.

LUTYPE6.2 links

For LUTYPE6.2 links, you can achieve bind-time security by specifying a **bind password** when you define the connection to the remote system.

Note: The implementation of bind-time security is optional in the LUTYPE6.2 architecture. Do not specify bind-time security if the remote system does not support it.

The method of defining a bind password is shown in Figure 109.

```
RDO Definition                Macro-Level Definition


DEFINE                        DFHTCT  TYPE=SYSTEM
  CONNECTION(name)                ,SYSIDNT=name
  GROUP(groupname)
  ACCESSMETHOD(VTAM)              ,ACCMETH=VTAM
  SECURITYNAME(name)              ,XSNAME=name
  PROTOCOL(APPC)                  ,TRMTYPE=LUTYPE62
  SINGLESESS(N)                   ,FEATURE=PARALLEL
  NETNAME(name)                   ,NETNAME=name
  BINDPASSWORD(password)          ,BINDPWD=password
Note: For APPC terminals defined as a TERMINAL-TYPETERM pair, the
BINDPASSWORD operand is on the DEFINE TERMINAL command.
```

Figure 109. Defining LUTYPE6.2 bind-time security

A password consists of up to 16 hexadecimal digits (0 through F), optionally surrounded by quotes. If you specify less than 16 digits, the password is padded on the **left** with hexadecimal zeros.

Each pair of communicating systems must have the **same** password for the link between them. For example, if you are defining a link between two CICS systems, the CONNECTION definition in each system must specify the same password. You are advised to select a unique password for each system that your CICS system will communicate with, rather than using the same password for two or more remote systems.

The specification of a bind password causes CICS to perform password checking each time a session is bound. This enables bind requests from systems that do not know the password to be rejected. It also prevents your bind requests from being intercepted and accepted by systems that do not know your password.

Failure to reconcile the two BINDPASSWORDs results in a message at the master terminal destination. The session is not bound, and the system reacts to a user request for a session with *sysiderr*.

The checking of a password involves three flows between the two systems. The protocol for these flows, which carry encryption information, is defined by the LUTYPE6.2 architecture, and is illustrated in Table 7. The **seeds** used to test the passwords are random numbers.

Table 7. LUTYPE6.2 bind-time security

System 1	**System 2**
1. Send SEED(1) to system 2	
	2. Encrypt SEED(1) using BINDPW(2)
	3. Return encrypted SEED(1/2) and SEED(2) to system 1
4. Encrypt SEED(1) using BINDPW(1)	
5. Compare encrypted SEED(1/1) with encrypted SEED(1/2) from system 2	
6. If equal, system 2 is the correct system	
7. Encrypt SEED(2) using BINDPW(1)	
8. Send encrypted SEED(2/1) to system 2	
	9. Encrypt SEED(2) using BINDPW(2)
	10. Compare encrypted SEED(2/2) with encrypted SEED(2/1) from system 1
	11. If equal, system 1 is the correct system

Although only one password is defined for each LUTYPE6.2 connection, the exchange of encryption information occurs for every session that is bound, to ensure continuing security. Information about the password itself is protected in the following ways:

1. The BINDPASSWORD field in the CEDA DEFINE CONNECTION panel is a non-display field. For this reason, it is better to use resource definition online to define connections, rather than coding the password in a DFHTCT TYPE=SYSTEM macro.
2. CICS does not store a "plain language" copy of the password, either in its internal control blocks or on the CICS system definition file (CSD).

3. The bind password is never transmitted between systems; it is used only as a factor in the encryption algorithm.

The LUTYPE6.2 architecture implemented by CICS is also designed to prevent the recording and subsequent misuse of bind-time password exchanges.

MRO links

For MRO links, the connection between CICS systems is handled by the CICS-supplied interregion SVC, which invokes the interregion program DFHIRP in supervisor state.

The establishment of an interregion link between two systems, say CICSA and CICSB, involves two connection requests.

1. The **send** sessions of CICSA must be bound to the **receive** sessions of CICSB. For these sessions, CICSA is the primary, or requesting, system, and CICSB must decide whether to allow the connection.
2. The **send** sessions of CICSB must be bound to the **receive** sessions of CICSA. For these sessions, CICSB is the primary, or requesting, system, and CICSA must decide whether to allow the connection.

CICS applies bind-time security to MRO links by comparing a **local security name** specified for the receiving system with a **passed security name** received from the requesting system.

In your particular CICS system, therefore, you need to provide two security names to achieve MRO bind-time security:

1. A security name for passing to the remote CICS system, in your role as a requesting system.
2. A local security name for the link, in your role as a receiving system.

A requesting system finds out its own security name by calling a security identification module (DFHACEE). The security name that this module returns is determined as follows:

- If RACF is present and active, DFHACEE returns the name specified in the USER= parameter of the CICS startup jobstream JOB card.
- If RACF is not active, or if the USER= parameter is omitted from the job card, DFHACEE returns a string of eight null (X'00') characters. In this case, the security name of the system is said to be **unknown**.

If you are not using RACF, but wish to implement MRO bind-time security, you can write your own version of DFHACEE to provide a security name for your system. Details are given in the *CICS/MVS Customization Guide*.

No matter how it is obtained, the security name is always passed by the requesting system along with the connection request.

The local security name of an MRO link, for **incoming** connection requests, is specified in the SECURITYNAME operand when you define the link (see "Chapter 3.1. Defining links to remote systems" on page 91), as shown in Figure 110 on page 318.

```
RDO Definition                 Macro-Level Definition

DEFINE                         DFHTCT  TYPE=SYSTEM
  CONNECTION(sysidnt)             ,SYSIDNT=sysidnt
  GROUP(groupname)
  ACCESSMETHOD(IRC|XM)            ,ACCMETH={IRC|(IRC,XM)}
  NETNAME(name)                   ,NETNAME=name
  SECURITYNAME(name)              ,XSNAME=name
```

Figure 110. Defining MRO bind-time security

The name specified in the SECURITYNAME operand is the security name that the requesting system must have for the connection to be allowed. If you do not specify a security name, any incoming connection request will be accepted.

During connect processing, CICS compares the security name passed by the remote system with the security name you have specified in the link definition. The possible results of the comparison are shown in Figure 111.

	Passed Security Name	
SECURITYNAME Operand	Valid Name	"Unknown" (null) Name
Specified	If the passed name is the same as the name specified in the SECURITYNAME operand, accept the connection request. Otherwise, reject the connection request.	Reject the connection request.
Omitted	Accept the connection request.	Accept the connection request.

Figure 111. Security-name checking for MRO links

LUTYPE6.1 links

For LUTYPE6.1 links, CICS cannot check the identity of the requesting system, and the bind request is never rejected on security grounds. For this reason, you are advised to use the intersystem security offered by LUTYPE6.2 links wherever possible.

Link security

Link security limits a remote system's authorization to attach your transactions and access your resources. You can specify it in one of two different ways:

1. By using the CICS signon table: You use the CICS signon table (DFHSNT) in your system to define CICS security values, or to specify RACF signon, or both, for the link. You must specify a security name when you define the connection, as shown in Figure 112. CICS matches the SECURITYNAME operand against the USERID of the DFHSNT TYPE=ENTRY macro each time a session is bound, and performs a signon with the security parameters defined in the macro. If you are using RACF or some other external security manager, you may only need to code the DFHSNT TYPE=(ENTRY,DEFAULT) macro.

| CICS does not sign the link on to RACF if the SECURITYNAME in the CONNECTION definition is the same as the RACF user identifier of the CICS system (USER parameter in the JOB card for the CICS startup jobstream).

If the signon fails, the usual signon failure message is sent to the master terminal destination, and the link is given the minimum security level; that is, it is able to access only unprotected resources.

```
RDO Definition                Macro-Level Definition

DEFINE                        DFHTCT  TYPE=SYSTEM
  CONNECTION                    .
  .                             .
  SECURITYNAME(name)           ,XSNAME=name
  .                             .
  .                             .

Note: For APPC terminals defined as a TERMINAL-TYPETERM pair, the
SECURITYNAME operand is on the DEFINE TERMINAL command.
```

Figure 112. Defining a link security name

For MRO links, the security name is also used for bind-time security; the bind will fail unless the requesting system passes the same name with the connection request (see Figure 111 on page 318). If you do not specify a security name, but the requesting system passes a name, the connection request is accepted and the **passed** name is used as the link security name.

| No signon for the link takes place if you are using RACF and the requesting system passes a name that is the same as the RACF user identifier of the receiving system (USER parameter on the JOB card of the CICS startup jobstream). Therefore, if you are using bind-time security and you want to apply effective link security, the SECURITYNAME on one side of an MRO link must not match the SECURITYNAME on the other side.

2. By specifying security in the link definition: You can specify CICS security keys for each set of sessions bymeans of the OPERSECURITY and OPERRSL operands, as shown in Figure 113 below.

```
RDO Definition                  Macro-Level Definition

DEFINE                          DFHTCT  TYPE=SYSTEM
  CONNECTION                       .
  .                                .
  .                                .
                                   .
DEFINE                             .
  SESSIONS                         .
  .                                .
  OPERSECURITY(number-list)       ,OPERSEC=number-list
  OPERRSL(number-list)            ,OPERRSL=number-list
  .                                .


Note: For APPC terminals defined as a TERMINAL-TYPETERM pair, the
OPERSECURITY and OPERRSL operands are on the DEFINE TERMINAL
command.
```

Figure 113. Defining preset link security in a link definition

As with ordinary terminals, OPERSECURITY and OPERRSL specify one or more keys as decimal values. A value of 1 is always implied for OPERSECURITY, meaning that the link always has access to unsecured transactions.

If you specify OPERSECURITY and/or OPERRSL, CICS will not perform a signon for the link, even if you have specified a security name.

User security

The real user of an interregion or intersystem link resides at the remote system. For example, if someone who has signed on at a terminal on a remote system invokes a transaction on your system (via CICS transaction routing), then that person is the real user of your transaction.

With MRO and LUTYPE6.2 links, information about the real user can be transmitted with the attach request from the remote system. This means that you can protect your resources not only on the basis of which remote system is making the request, but also on the basis of which actual user at the remote system is making the request.

If you decide to apply CICS user security, you must specify an option on the link definition as described in "User security in link definitions" on page 322. Then you define each remote user by an entry in the CICS signon table (DFHSNT TYPE=ENTRY). The entry in the signon table can specify CICS security values, or external security, or both. For those users who are subject to external security only you can provide a suitable default entry (DFHSNT TYPE=(ENTRY,DEFAULT)). If you select external security, you probably have to define your users to the appropriate external security manager: for example, RACF.

If a remote user is not defined by an entry in the CICS signon table (and no external security manager is in use), then any attach requests from that remote user will be rejected.

There can also be *hidden* users of your system. For example, the remote CICS system itself may want to attach transactions (see Table 8 on page 322). Depending on your security strategy, you may need to cover each of these by an entry in the signon table, a RACF definition, or both.

Make sure that the security profile you define for the link is a collation of all the separate profiles that you have allotted to remote users.

Note that remote users are "signed on" to your system automatically. They do not even have to be aware that requests are being shipped on their behalf. As stated earlier in this chapter, you are responsible for defining what security is required, and CICS takes care of the rest.

With LUTYPE6.1 links, information about the remote user is not available for security purposes. In this case, the authority of the user is taken to be that of the link itself, and you have to rely on link security alone to protect your resources.

Information about remote users

This section describes some of the concepts associated with user security, and how CICS sends and receives user information.

CICS user security for LUTYPE6.2 links conforms to the LUTYPE6.2 architecture, and CICS implements similar facilities for MRO links. No user security is available with LUTYPE6.1 links.

The LUTYPE6.2 architecture allows user identifiers, user passwords, and user profiles to be transmitted with requests to attach a transaction. User profiles can be transmitted instead of or as well as user identifiers. CICS, however, makes no use of profiles, and ignores them if they are received. They are therefore omitted from the rest of this discussion.

CICS always sends a user identifier with attach requests that it sends on LUTYPE6.2 and MRO links. Table 8 on page 322 shows how CICS decides what user identifier to send.

CICS never sends a password with attach requests that it sends to remote systems. This is because CICS can perform its own password checking, and it is assumed that the security set up in the local system does not allow unauthenticated users access to remote systems.

Special rules apply to links with CICS/VM and CICS OS/2 systems. See the *CICS/VM System Support and Administration* manual and the *CICS OS/2 System and Application Guide* for details.

Table 8. LUTYPE6.2 and MRO attach-time user identifiers

Characteristics of the Local Task	**User Identifier Sent by CICS to the Remote System**
Task with associated terminal – user signed on or USERID specified in the terminal definition.	Terminal user identifier
Task with associated terminal – no user signed on and no USERID specified in the terminal definition.	Default (null) user identifier
Task with no associated terminal started by interval control START command	User identifier for the task that issued the START command
Task with no associated terminal started by transient data trigger	Default (null) user identifier
CICS system task	The SYSIDNT of the CICS system

Signing on the remote user has two purposes:

1. To ensure that the remote user is allowed to access the CICS system.
2. If the signon is successful, to establish a security profile for the remote user.

Apart from the fact that it is performed automatically by CICS, this signon is just the same as a normal operator signon in a single CICS system.

CICS signs off the remote user under the circumstances described in "CICS action with ATTACHSEC(Identify)" on page 323 and "CICS action with ATTACHSEC(Verify)" on page 324.

User security in link definitions

The level of user security you require for a remote system is specified in the ATTACHSEC operand of the DEFINE CONNECTION command, as shown in Figure 114.

```
RDO Definition                  Macro-Level Definition

DEFINE                          DFHTCT  TYPE=SYSTEM
  CONNECTION(name)                 ,SYSIDNT=name
  GROUP(groupname)
  .                                .
  ATTACHSEC(Local|                 ,USERSEC={LOCAL|
            Identify|                        IDENTIFY|
            Verify)                          VERIFY}

Note: For APPC terminals defined as a TERMINAL-TYPETERM pair, the
ATTACHSEC operand is on the DEFINE TERMINAL command.
```

Figure 114. User security in link definition

The ATTACHSEC operand specifies your security requirements for **incoming** attach requests. It has no effect on attach requests that are issued by your system to a remote system.

Except for CICS transaction routing, where special rules apply (see "Transaction routing security" on page 326), the parameters of the ATTACHSEC operand have the following meanings:

Local

specifies that a user identifier is not required from the remote system and if one is received it is ignored. Here, CICS makes the user security profile equivalent to the link security profile. You do not need to specify entries in the signon table for the remote users. Local is the default value.

ATTACHSEC(Local) **must** be used for all LUTYPE6.1 links and also for LUTYPE6.2 links to systems that do not send user identifiers.

You can choose to use ATTACHSEC(Local) for other links if you think that the link security profile alone provides sufficient security for your system.

Identify

specifies that a user identifier is required on every attach request. This level of user security is recommended for CICS-to-CICS communication over LUTYPE6.2 and MRO links.

ATTACHSEC(Identify) is appropriate when CICS can "trust" the remote system to verify its users (by some sort of signon mechanism) before allowing them to use the link to CICS.

If an attach request with both a user identifier and a password is received on a link with ATTACHSEC(Identify), CICS will **verify** by performing a signon with password.

If a null user identifier is received, CICS applies minimum security capabilities only.

Verify

specifies that, in addition to a user identifier, a user password is required for verification against the locally-defined password.

ATTACHSEC(Verify) is appropriate when the remote system does not have a security manager and therefore cannot verify its own users.

CICS action with ATTACHSEC(Identify)

When an attach request with a particular user identifier is first received on a link defined with ATTACHSEC(Identify), CICS carries out a signon without password to ensure that the user has authority to access this system, and to establish security levels for the user.

Thereafter, CICS accepts attach requests from the same user without a new signon until one of the following occurs:

1. A period of over 30 minutes has elapsed since the previous attach request from this user.
2. The link to the remote system is broken.

In either of these events, CICS "forgets" the remote user, and carries out a new signon to reestablish the user's authority on the next attach request.

Warning: If you are using RACF, and you alter the authority of a signed-on remote user, CICS will continue to use the security values acquired at the first attach request until one of the events previously listed causes the user to be forgotten.

CICS action with ATTACHSEC(Verify)

A specification of ATTACHSEC(Verify) means that you require a user identifier and a user password with every attach request.

Every time that an attach request with a user identifier and user password is received on a link defined with ATTACHSEC(Verify), CICS carries out a signon with password to ensure that the user has authority to access this system, to verify that the password is correct, and to establish security levels for the user.

CICS "forgets" verified users between attach requests; a new signon is performed every time.

Securing transactions and resources

The last step in defining security for your system is to make sure that the access parameters for your transactions and resources match the profiles you have defined for the link and the individual remote users.

Transactions

The security requirements of a transaction are specified when the transaction is defined (see the *CICS/MVS Resource Definition (Online)* manual or the *CICS/MVS Resource Definition (Macro)* manual). A transaction can be defined with no security, with CICS security, or with external security (RACF), as shown in Figure 115.

```
No Security          CICS Security          External Security

DEFINE               DEFINE                 DEFINE
  TRANSACTION          TRANSACTION            TRANSACTION
  .                    .                      .
  EXTSEC(NO)           EXTSEC(NO)             EXTSEC(YES)
  TRANSEC(1)           TRANSEC(2-64)
```

Figure 115. Specifying transaction security

Before a secure transaction can be initiated, two basic security requirements must be met.

1. The link must have sufficient authority to initiate the transaction.
2. The "user" who is making the request via the link must have sufficient authority to access the system and to initiate the transaction.

If the transaction specifies resource security checking, both the link and the user must also have sufficient authority for the resources that the attached transaction will access.

Resources

Resource security in the intercommunication environment is handled in much the same way as it is in a single-system environment.

Resource security checking is performed only if the program control table entry for the transaction specifies that it is required (YES or EXTERNAL in the RSLC operand). If it is specified, two conditions must be satisfied before the attached transaction is allowed to access a protected resource:

1. The link must have sufficient authority to access the resource.
2. The remote user must have sufficient authority to access the resource. For LUTYPE6.1 links, and for links that specify ATTACHSEC(Local), the user authority is the same as the link authority.

The way in which resource security values for links and remote users are acquired has been described earlier in this chapter.

DL/I databases

For resources other than DL/I databases, you can use either RACF security or CICS security. For DL/I databases, an external security manager such as RACF must be used if resource security is required.

CICS function shipping security

When CICS receives a shipped function request, the transaction that is invoked is the mirror transaction. The CICS-supplied definitions of the mirror transactions (CSMx and CVMI) specify resource security level checking, but not transaction security. They can therefore be invoked by any remote system and user that are allowed to access your system, but they can access only those resources for which the link and remote user have authority.

You can modify the appropriate mirror transaction definitions to achieve the level of security that you require.

Security considerations for the user

If you include a remote resource in your resource definitions, you can arrange for security checking to be done locally, just as if the resource were a local one. Also, the system that owns the resource can be made to apply an independent check, if it is able to receive the user identifier from you. You can therefore choose to apply security restrictions on both sides, on either side, or not at all.

If you specify RSLC(YES) or RSLC(EXTERNAL) for a transaction, the NOTAUTH condition is raised if the transaction attempts to issue an EXEC CICS command with the SYSID option specified. This is to prevent transactions from bypassing the local security check.

The NOTAUTH condition

If a transaction attempts to access a resource, but fails the resource security checks, the NOTAUTH condition is raised.

When the transaction is the CICS/MVS mirror transaction, the NOTAUTH condition is returned to the requesting transaction, where it can be handled in the usual way.

Note: If you have problems with a mirror transaction abending during function shipping, wrong resource security levels are a likely cause.

Transaction routing security

In CICS transaction routing, the transaction in the application-owning region has as its principal facility a remote terminal (the "surrogate") that represents the "real" terminal in the terminal-owning region (see "Chapter 1.6. CICS transaction routing" on page 45). The way in which the remote terminal is defined (see "Chapter 3.2. Defining remote resources" on page 133) can have an effect on the way in which user security is applied.

1. If the definition of the remote terminal contains **neither** OPERSECURITY **nor** OPERRSL values:
   - For links with ATTACHSEC(Identify), the transaction security and resource security of the user are established when the remote user is signed on.
   - For links with ATTACHSEC(Local), the transaction security and resource security are handled in different ways. Transaction security is set to that of the link but resource security is set to minimum capability; that is, the user is allowed to access unprotected resources only. This is to prevent unauthenticated users from achieving wider security capabilities in the AOR than they might have in the TOR.
2. If the definition of the remote terminal contains **either** OPERSECURITY **or** OPERRSL values, the security characteristics of the remote user are ignored, and the values coded in the terminal definition are used.

In both cases, tests against the **link** security are made as described earlier in this chapter.

Notes:

1. OPERSECURITY and OPERRSL values are not transferred when a terminal definition is shipped from one region to another (see "Shipping terminal definitions" on page 141). Terminal definitions that have been acquired by shipping therefore always correspond to case 1 given above.
2. During transaction routing, the operator identifier from the terminal-owning region (if it is available) is transferred to the surrogate terminal entry in the application-owning region. This identifier is not used for security purposes, but it may be referred to in messages and audit trails.

Use of MVS cross-memory services

MVS cross-memory services can be used for interregion communication (function shipping, transaction routing, distributed transaction processing, and asynchronous processing).

If you use cross-memory services, you will lose the total separation between systems that is normally provided by separate address spaces.

The risk of accidental interference between two CICS address spaces connected by a cross-memory link is small. However, an application program in either system could access the other system's storage (subject to key-controlled protection) by using a sequence of cross-memory instructions.

If this situation would create a security exposure in your installation, you should use the CICS SVC for interregion communication.

Chapter 6.2. Master terminal operations for LUTYPE6.2 connections

This chapter offers advice on managing LUTYPE 6.2 connections using the master terminal transaction (CEMT) and the interaction between these commands and the way these resources have been defined to CICS.

The commands are described under the headings:

- Acquiring the connection
- Controlling and monitoring sessions on the connection
- Releasing the connection.

The commands used to achieve these actions are:

- CEMT SET CONNECTION ACQUIRED/RELEASED
- CEMT SET MODENAME AVAILABLE/ACQUIRED/CLOSED

Detailed formats and options of CEMT commands are given in the *CICS/MVS CICS-Supplied Transactions* manual.

The information is mainly about parallel-sessions connections between CICS systems.

General information

The operator commands controlling LUTYPE6.2 connections cause CICS to execute many internal processes, some of which involve communication with the partner systems. The major features of these processes are described below, but you should note that the processes are sometimes independent of one another and can be asynchronous. This makes simple descriptions of them imprecise in some respects. The execution can occasionally be further modified by independent events occurring in the network, or simultaneous operator activity at both ends of an LUTYPE6.2 connection; these circumstances are more likely when a component of the network has failed and recovery is in progress. The following sections explain the normal operation of the commands.

Note: The principles of operation described in this chapter also apply to the EXEC CICS INQUIRE CONNECTION, INQUIRE MODENAME, SET CONNECTION, and SET MODENAME commands (for more information, see the *CICS/MVS Customization Guide*).

Acquiring a connection

The SET CONNECTION ACQUIRED command causes CICS to establish a connection with a partner system. The major processes involved in this operation are:

- Establishment of the two LU services manager sessions in the modegroup SNASVCMG.
- Initiation of the change-number-of-sessions (CNOS) process by the partner initiating the connection.

CNOS negotiation is executed (using one of the LU services manager sessions) to determine the numbers of contention-winner and contention-loser sessions defined in the connection. The results of the negotiation are reported in messages DFH4900 and DFH4901.

- Establishment of the sessions that carry CICS application data.

The following processes, also part of connection establishment, are described in "Part 5. Recovery and restart" on page 287.

- Exchange lognames
- Resolution and reporting of synchronization information.

Connection status during the acquire process

The status of the connection before and during the acquire process is reported by the INQUIRE CONNECTION command as follows:

Released Initial state before the SET CONNECTION ACQUIRED command. All the sessions in the connection are released.

Obtaining Contact has been made with the partner system, and CNOS negotiation is in progress.

Acquired CNOS negotiation has completed for all modegroups. In this status CICS has bound the LU services manager sessions in the modegroup SNASVCMG. Some of the sessions in the user modegroups may also have been bound, either as a result of the AUTOCONNECT operand on the SESSIONS definition or to satisfy allocate requests from applications.

Acquiring sessions: effect of the AUTOCONNECT operand

The results of requests for the use of a connection by application programs depend on the status of the sessions. You can control the status of the sessions with the AUTOCONNECT operand of the SESSIONS definition as described in the following section.

CICS can bind contention-winner sessions to satisfy an application request, but not contention losers. However, it can assign contention-loser sessions to application requests if they are already bound. Considerations for binding contention losers are described in the next section. For more detailes about the use of sessions by an application, see "Session allocation" on page 174.

The meanings of the AUTOCONNECT operand for LUTYPE6.2 connections are described in "The AUTOCONNECT operand" on page 125. The effect of the AUTOCONNECT operand of the SESSIONS definition is to control the acquisition of sessions in modegroups associated with the connection. Each modegroup has its own AUTOCONNECT parameter and the setting of this parameter affects the sessions in the modegroup as described in Table 9.

Table 9. Effect of AUTOCONNECT on the SESSIONS definition.

Setting	Effect
Yes	CNOS negotiation with the partner system is performed for the modegroup, and all negotiated contention-winner sessions are acquired when the connection is acquired.
No	CNOS negotiation with the partner system is performed, but no sessions are acquired. Contention-winner sessions can be bound individually according to the demands of application programs (for example, when a program issues an ALLOCATE command), or the SET MODENAME ACQUIRED command can be used to bind contention-winner sessions.
All	CNOS negotiation with the partner system is performed for the modegroup, and all negotiated sessions, contention winners, and contention losers are acquired when the connection is acquired. This setting should be necessary only on connections to non-CICS systems.

When the connection is in ACQUIRED status, the INQUIRE MODENAME command can be used to determine whether the user sessions have been made available and activated as required. The binding of user sessions is not completed instantaneously, and you may have to repeat the command to see the final results of the process.

Binding contention-loser sessions

Contention-loser sessions on one system are contention-winner sessions on the partner system, and should be bound by the partner as described above. If you want all sessions to be bound, you must make sure each side binds its contention winners.

If the connection is between two CICS systems, specify AUTOCONNECT(YES) on the SESSIONS definition for each system, or issue CEMT SET MODENAME ACQUIRED from both systems. If you are linked to a non-CICS system that is unable to send bind requests, specify AUTOCONNECT(ALL) on your SESSIONS definition.

If the remote system can send bind requests, find out how you can make it bind its contention winners so that it does so immediately after the SNASVCMG sessions have bound.

The ALLOCATE command, either as an explicit command in your application or as implied in automatic transaction initiation, cannot bind contention-loser sessions, although it can assign them to conversations if they are already bound.

Effects of the MAXIMUM operand

The MAXIMUM operand of the SESSIONS definition specifies

- The maximum number of sessions that can be supported for the modegroup
- The number of these that are supported as contention winners.

Operation of LUTYPE6.2 connections is made easier if the maximum number of sessions at each end of the connection match, and the number of contention-winner sessions specified at the two ends add up to this maximum number. If this is done, CNOS negotiation does not change the numbers specified.

If the specifications at each end of the connection do not match, as has just been described, the actual values are negotiated by the LU services managers. The result of the negotiation on the maximum number of sessions is to adopt the lower of the two values. An architected algorithm is used to determine the number of contention winners for each partner, and the results of the negotiation are reported in messages DFH4900 and DFH4901.

These results can also be deduced as shown in Table 10 by issuing a CEMT INQUIRE MODENAME command.

Table 10. INQ MODENAME display

Display	Interpretation
MAXimum	The value specified in the sessions definition for this modegroup. This represents the true number of usable sessions only if it is equal to or less than the corresponding value displayed on the partner system.
AVAilable	Represents the result of the most recent CNOS negotiation for the number of sessions to be made available and potentially active. Following the initial CNOS negotiation it reports the result of the negotiation of the first operand of the MAXIMUM operand.
ACTive	The number of sessions currently bound.

To change the MAXIMUM values, release the connection, set it OUTSERVICE, redefine it with new values, and install it using the CEDA transaction.

Controlling sessions with the SET MODENAME commands

The SET MODENAME commands can be used to control the sessions within the modegroups associated with an LUTYPE6.2 connection, without releasing or reacquiring the connection. The processes executed to accomplish this are:

- CNOS negotiation with the partner system to define the changes that are to take place.
- Binding or unbinding of the appropriate sessions.

The algorithms used by CICS to negotiate with the partner the numbers of sessions to be made available are complex, and the numbers of sessions actually acquired may not match your expectation. The outcome can depend on the following:

- The history of preceding SET MODENAME commands
- The activity in the partner system
- Errors that have caused CICS to place sessions out of service.

Modegroups can normally be controlled with the few simple commands described in Table 11 on page 333.

Table 11. SET MODENAME commands

Command	**Effect**
SET MODENAME ACQUIRED	Acquires all negotiated contention-winner sessions.
SET MODENAME CLOSED	Negotiates with the partner to reduce the available number of sessions to zero, releases the sessions, and prevents any attempt by the partner to negotiate or activate any sessions in the modegroup. Only the system issuing the command can subsequently increase the session count.
SET MODENAME AVAIL(maximum) ACQUIRED	If this command is issued when the modegroup is closed, the sessions are negotiated as if the connection had been newly acquired, and the contention-winner sessions are acquired. It can also be used to rebind sessions that have been lost due to errors that have caused CICS to place sessions out of service.

Commands, scope and restrictions

User modegroups, which are built from CEDA DEFINE SESSIONS (or macro equivalent) definitions, can be modified by using the SET MODENAME command or by overtyping the INQUIRE MODENAME screen.

The SNASVCMG modegroup is built from the CONNECTION definition and any attempts to modify its status with a SET MODENAME command, or by overtyping the INQUIRE MODENAME screen, will be suppressed. It is controlled by the SET CONNECTION command, or by overtyping the INQUIRE CONNECTION screen, which will also affect associated user modegroups.

CEMT INQUIRE NETNAME(....), where the netname is the APPLID of the partner system, displays the status of all sessions associated with that connection and can be useful in error diagnosis. Any attempt to alter the status of these sessions by overtyping, will be suppressed.

You must use the SET/INQ CONNECTION/MODENAME to manage the status of user sessions and to control negotiation with remote systems.

Releasing the connection

The SET CONNECTION RELEASE command causes CICS to quiesce a connection and release all sessions associated with it. The major processes involved in this operation are:

- Execution of the CNOS process to inform the partner system that the connection is closing down. The number of available sessions on all modegroups is reduced to 0.

- Quiescing of transaction activity using the connection. This process allows the completion of transactions that are using sessions and queued allocate requests; new requests for session allocation are refused with the SYSIDERR condition.
- Unbinding of the user and LU services manager sessions.

Connection status during the release process

The following states are reported by the CEMT INQUIRE CONNECTION command before and during the release process.

Acquired	Sessions are acquired; the sessions can be allocated to transactions.
Freeing	Release of the connection has been requested and is in progress.
Released	All sessions are released.

If you have control over both ends of the connection, or if your partner is unlikely to issue commands that conflict with yours, you can use SET CONNECTION RELEASED to quiesce activity on the connection. When the connection is in the RELEASED state, SET CONNECTION OUTSERVICE can be used to prevent any attempt by the partner to reacquire the connection.

If you do not have control over both ends of the connection, you should use the sequence of commands described in "Making the connection unavailable."

The effects of limited resources

If an LUTYPE6.2 connection traverses non-leased links (such as Dial, ISDN, X.25, X.21 or Token Ring links) to communicate to remote systems, the links can be defined within the network as limited resources. CICS recognizes this definition and automatically unbinds the sessions as soon as no transactions require them. If new transactions are invoked that require the connections, CICS will bind the appropriate number of sessions. The connection status is shown by the CEMT INQUIRE CONNECTION command as follows:

Acquired	Some of the sessions in the connection are bound, and are probably in use. The LU services manager sessions in modegroup SNASVCMG may be unbound.
Available	The connection has been acquired, but there are no transactions that currently require the use of the connection. All the sessions have been unbound because they are defined in the network as limited resources.

The connection behaves in other ways exactly as for a connection over non-limited-resource links. The SET MODENAME and SET CONNECTION RELEASED commands operate normally.

Making the connection unavailable

The SET CONNECTION RELEASED command quiesces transactions using the connection and releases the connection. It cannot, on its own, prevent reacquisition of the connection from the partner system. To prevent your partner from reacquiring the connection, you must execute a sequence of commands. The choice of command sequence determines the status the connection adopts and how it responds to further commands from either partner.

If the number of available sessions for every modegroup of a connection is reduced to zero[6], ALLOCATE requests that do not specify NOQUEUE or NOSUSPEND are suspended. Transaction routing and function shipping requests are also suspended. The connection is effectively unavailable. However, because the remote system can renegotiate the availability of sessions and cause those sessions to be bound, you cannot be sure that this state will be held.

To prevent your partner from acquiring sessions that you have made unavailable, use the CEMT SET MODENAME CLOSED command. This reduces the number of available user sessions in the modegroup to zero and also ***locks*** the modegroup. Even if your partner now issues SET CONNECTION RELEASED followed by SET CONNECTION ACQUIRED, no sessions in the locked modegroup become bound until you specify an AVAILABLE value greater than zero.

If you lock all the modegroups, you make the connection unavailable, because the remote system can neither bind sessions nor do anything to change the state. On both sides, transaction programs that need sessions are suspended, unless the NOQUEUE or NOSUSPEND option has been specified on the ALLOCATE request, and all further progress depends on the action of only one of the two operators. Make sure the connection does not remain in this state for any length of time.

Having closed all the modegroups for a connection, you can go a step further by issuing CEMT SET CONNECTION RELEASED. This unbinds the SNASVCMG (LU services manager) sessions. An inquiry on the CONNECTION returns INSERVICE RELEASED (or INSERVICE FREEING if the release process is not complete). Transaction programs suspended by CEMT SET MODENAME CLOSED now resume but return SYSIDERR as for *connection out of service*.

If you now enter SET CONNECTION ACQUIRED, you free all locked modegroups and the connection is fully established. If, instead, your partner issues the same command, only the SNASVCMG sessions are bound and transaction programs on both sides may well suspend as earlier described.

You can prevent your partner from binding the SNASVCMG sessions by invoking CEMT SET CONNECTION OUTSERVICE, which is ignored if the connection is already in the RELEASED state.

[6] By, for example, a CEMT SET MODENAME AVAILABLE(0) command

To summarize, you can make a connection unavailable and retain it under your control by issuing these commands in the order shown:

Making the connection unavailable.

```
CEMT SET MODENAME(*) CONNECTION(....) CLOSED
```

[The CONNECTION option is significant only if the MODENAME applies to more than one connection.]

```
        INQ MODENAME(*) CONNECTION(....)
```

[Repeat this command until the AVAILABLE count for all non-SNASVCMG modegroups becomes zero.]

```
SET CONNECTION(....) RELEASED
     INQ CONNECTION(....)
```

[Repeat this command until the RELEASED status is displayed.]

```
SET CONNECTION(....) OUTSERVICE
```

Diagnosing and correcting error conditions

User sessions that have become unavailable because of earlier failures can be brought back into use by restoring or increasing the *available count* with the SET MODENAME AVAILABLE(n) command. The addition of the ACQUIRED option to this command will result in the binding of any unbound contention-winner sessions.

If the SNASVCMG sessions become unbound while user sessions are active, the connection is still acquired. A SET CONNECTION ACQUIRED command binds all contention-winner sessions in all modegroups, and may be sufficient to re-establish the SNASVCMG sessions.

Sometimes, you may not be able to recover sessions, although the original cause of failure has been removed. Under these circumstances, you should first release then reacquire the connection.

Summary

Figure 116 on page 337 summarizes the effect of CEMT commands on the status of an LUTYPE6.2 link.

Commands issued in sequence shown	1	1	1						SET MODENAME AVAILABLE(0)
				1	1	1			SET MODENAME CLOSED
		2	2		2	2	1	1	SET CONNECTION RELEASED
			3			3		2	SET CONNECTION OUTSERVICE
Resulting states and reactions	Y	N	N	Y	N	N	N	N	ALLOCATE requests suspended
	Y	Y	N	N	N	N	Y	N	Partner can renegotiate
	N	Y	Y	N	Y	Y	Y	Y	ALLOCATE rejected with SYSIDERR
	N	Y	Y	N	Y	Y	Y	Y	SNASVCMG sessions released
	n/a	Y	N	n/a	Y	N	Y	N	Partner can rebind SNASVCMG

Figure 116. Effect of CEMT commands on an operational LUTYPE6.2 link.

Appendix A. Rules and restrictions checklist

This appendix provides a checklist of the rules and restrictions that apply to intersystem communication and multiregion operation. Most of these rules and restrictions also appear in the body of the book.

Macro-level and command-level transactions

- In general, remote data resources can be accessed only by command-level transactions.
- Function-shipping, distributed transaction processing, and asynchronous processing are available only to command-level transactions.
- Macro-level interval control requests cannot be used for starting a transaction in a remote system (asynchronous processing). In other words, transactions initiated by macro-level interval control requests must reside on the same system as the transaction that initiates them.
- Transaction routing is available to both command-level and macro-level transactions. Both command-level and macro-level transactions can be initiated from a remote terminal, and can acquire a remote terminal when they are started by automatic transaction initiation.
- All programs, tables, and maps that are used by a transaction must reside on the system that owns the transaction (the programs, tables, and maps can be duplicated in as many systems as necessary).

Transaction routing

- A transaction routing path between a terminal and a transaction must not turn back on itself. For example, if system A specifies that a transaction is on system B, system B specifies that it is on system C, and system C specifies that it is on system A, the attempt to use the transaction from system A will be abended when system C tries to route back to system A.

  This restriction also applies if the routing transaction (CRTE) is used to establish all or part of a path that turns back on itself.
- Transaction routing using the following terminals is not supported:
  - LUTYPE6.1 sessions
  - IBM 7770 terminals
  - Pipeline logical units with pooling
  - Pooled TCAM terminals
  - The MVS system console. (Messages entered through a console can be directed to any CICS system via the MODIFY command.)
- The transactions CEOT and CSOT are not supported by the transaction routing facility.
- The transaction CSMT is not supported from a remotely-owned 2780 for which BSCODE=ASCII is specified.

- The execution diagnostic facility (EDF) can be used in single terminal mode to test a remote transaction provided that the routing transaction CRTE is used to invoke both EDF and the transaction under test.

  EDF running in two-terminal mode is supported only when both of the terminals and the user transaction reside on the same system; that is, when no transaction routing is involved.

- The user area of the TCTTE is updated at task attach and task detach times. Therefore, a user exit program running on the terminal-owning region and examining the user area while the terminal is executing a remote transaction will not necessarily see the same values as a user exit running at the same time in the application-owning region. Note also that the user areas must be defined as having the same length in both systems.

- Application programs that use the DFHTC TYPE=SIGNAL macro instruction without the WAIT option may behave differently when running on a remotely-owned terminal. This is because the signal indicator is passed to the application only when a terminal control request with a WAIT implied is issued by the application program. If the application does not issue such a request then continually testing the signal indicator by means of DFHTC TYPE=SIGNAL macro instruction will continually fail to detect the inbound signal.

Basic mapping support

- BMS support must reside on each system that owns a terminal through which paging commands can be entered.

- A BMS ROUTE request cannot be used to send a message to a selected remote operator or operator class unless the terminal at which the message is to be delivered is specified in the route list.

Automatic transaction initiation

- A terminal-associated transaction that is initiated by the transient data trigger level facility must reside on the same system as the transient data queue that causes its initiation. This restriction applies to both macro-level and command-level application programs.

- If a transaction is started by ATI on a remotely-owned terminal, the transaction must be defined on the terminal-owning region as a remote resource owned by the system where the ATI request is issued (but see "Shipping terminals for automatic transaction initiation" on page 48).

Allocating LUTYPE6.1 sessions

- If an application issues a ALLOCATE command for an LUTYPE6.1 connection, and the remote system is unavailable, the connection is placed out-of-service.

- If the remote system is a CICS system that uses AUTOCONNECT, the connection is placed back in service when the initialization of the remote system is complete.

- If the remote system does not specify AUTOCONNECT, or if it is a non-CICS system that does not have autoconnect facilities, you must place the connection back in service by using a CEMT SET CONNECTION command or by issuing an EXEC CICS SET CONNECTION command from an application program.

Syncpointing

- SYNCPOINT ROLLBACK calls are supported only by LUTYPE6.2 and MRO sessions.

Local and remote names

- Transaction identifiers are translated from local names to remote names when a request to execute a transaction is transmitted from one CICS system to another.

  However, a transaction identifier specified in an EXEC CICS RETURN command (or a DFHPC TYPE=RETURN macro) is not translated when it is transmitted from the application-owning region to the terminal-owning region.

- Terminal identifiers are translated from local names to remote names when a transaction routing request to execute a transaction on a specified terminal is shipped from one CICS system to another.

  However if an EXEC CICS START command specifying a terminal identification is function shipped from one CICS system to another, the terminal identification is not translated from local name to remote name.

Master terminal transaction

- Only locally-owned terminals can be queried and modified by the master terminal transactions CSMT and CEMT. The only terminals visible to these transactions are those owned by the system on which the master terminal transaction is actually running.

Installation and operations

- Module DFHIRP must be made LPA-resident, otherwise jobs and console commands may abend on completion.
- Interregion communication requires subsystem interface (SSI) support.
- The SIT, or system initialization overrides, must not specify SRT=NO.

Resource definition

- The PRINTER and ALTPRINTER operands for a VTAM terminal must (if specified) name a printer owned by the same system as the terminal being defined.
- The terminals listed in the terminal list table (DFHTLT) must reside on the same system as the terminal list table.

Customization

- Communication between node error programs, user exits, and user programs is the responsibility of the user.
- The DFHTC CTYPE macros cannot be used for terminals that are owned by remote systems.
- Transactions that recover input messages for protected tasks after a system crash must run on the same system as the terminal that invoked the protected task.

MRO abend codes

- An MRO transaction in send state is unable to receive an error reason code if its partner has to abend. It abends itself with code AZI2, which should be interpreted as a general indication that the other side is no longer there. The real reason for the failure can be read from the CSMT destination of the CICS region that first detected the error. For example, a security violation in attaching a back-end transaction is reported as such by the front end only if the initiating command is CONVERSE and not SEND.

Appendix B. Sample application programs

This appendix describes several CICS application programs that illustrate the use of distributed transaction processing and asynchronous processing on LUTYPE6.2 and LUTYPE6.1 links.

The samples, all of which are written in assembler language, are presented in the following order:

1. The transfer of a temporary storage queue from a local CICS system to a remote CICS system, using distributed transaction processing and LUTYPE6.2 protocols.
2. Browsing a remote file, using distributed transaction processing. This sample can be used on LUTYPE6.2, LUTYPE6.1, or MRO links.
3. The retrieval of a record from a remote temporary storage queue, using asynchronous processing. This sample can be used on LUTYPE6.2, LUTYPE6.1, or MRO links.
4. A CICS to remote LUTYPE6.1 system (CICS or IMS) conversation. LUTYPE6.1 links must be used for this sample.
5. A simple CICS to IMS conversation. LUTYPE6.1 links must be used for this sample.
6. A CICS to IMS sample showing the use of demand paging. LUTYPE6.1 links must be used for this sample.

The sample programs and their associated BMS mapsets are provided in both source and object form on the CICS distribution volume.

Sample programs: The source modules are on library CICS212.SAMPLIB and the object modules are on library CICS212.LOADLIB. The source and object modules have the same names. For all the sample programs, the transaction name is the last four characters of the module name.

Sample mapsets: The source modules and the symbolic description maps are on library CICS212.SAMPLIB. The physical maps are on CICS212.LOADLIB. All the BMS mapset source modules have names of the form DFH$IMx. The symbolic description map and the physical map generated from DFH$IMx are both named DFH$IGx.

Resource definition: For resource definition online (RDO), all the transaction, program, and mapset definitions for the sample programs are provided in the CICS-supplied group DFH$ICOM. If you are not using RDO, you must supply appropriate definitions in the program control table (DFHPCT) and the processing program table (DFHPPT).

The names of the source modules for the sample programs and mapsets are shown in Table 12 on page 344.

Table 12. Summary of sample application programs

Sample	**Description**	**Source modules**	**Module description**
1	Transfer of a temporary storage queue to a remote CICS system.	DFH$IQXL DFH$IQXR DFH$IMX	Local transaction Remote transaction BMS mapset DFH$IGX
2	Browsing a remote file.	DFH$IFBL DFH$IFBR DFH$IMB	Local transaction Remote transaction BMS mapset DFH$IGB
3	Retrieval of a record from a remote temporary storage queue.	DFH$IQRL DFH$IQRR DFH$IQRD DFH$IM1 DFH$IM2	Local request transaction Remote retrieve transaction Local display transaction BMS mapset DFH$IG1 BMS mapset DFH$IG2
4	CICS to CICS or IMS conversation	DFH$ICIC DFH$IMC	Local/remote transaction BMS mapset DFH$IGC
5	CICS to IMS conversation	DFH$IMSN DFH$IMS	CICS transaction BMS mapset DFH$IGS
6	CICS to IMS conversation with demand paging	DFH$IMSO DFH$IMS	CICS transaction BMS mapset DFH$IGS

Sample 1 — temporary storage queue transfer

This sample illustrates the use of distributed transaction processing to transmit a temporary storage queue to a remote system. It consists of a front-end transaction (DFH$IQXL), a back-end transaction (DFH$IQXR), and a BMS mapset (DFH$IMX) that is used by the front-end transaction.

The front-end transaction is invoked by the transaction code IQXL, and displays the following menu at the user's terminal:

```
              CICS-CICS QUEUE TRANSFER
                SAMPLE PROGRAM MAP
              ***************************


           LOCAL TS Q NAME ...
           REMOTE TS Q NAME ..  REMOTEQ
           REMOTE SYSTEM ID ..


           TYPE IN VALUES, THEN PRESS ENTER
           OR HIT "PF3" TO TERMINATE.
```

The displayed menu has three input fields:

LOCAL TS Q NAME
specifies the name of the local temporary storage queue that is to be transferred to the remote system.

If this field is left blank, the front-end transaction will itself build a small (5 records) temporary storage queue to transfer to the remote system.

REMOTE TS Q NAME
specifies the name that the transferred queue is to be given on the remote system.

The menu supplies the default name REMOTEQ.

REMOTE SYSTEM ID
specifies the name of the remote system.

This name must be the connection name of an LUTYPE6.2 link.

The front-end transaction initiates the back-end transaction and transmits the temporary storage records for writing on the remote queue.

The user is informed of data input errors, and also of the progress of the queue transfer operation. The local temporary storage queue is deleted after successful completion.

Figure 117 on page 346 shows the overall flow of the queue-transfer sample.

```
         Local Transaction                    Remote Transaction
           (front-end)                           (back-end)

1. . . . Get User Requirements
2. . . . Create TS Queue if Needed
3. . . . ALLOCATE SYSID( )
         (MVC ATCHSESS,EIBRSRCE)
4. . . . CONNECT PROCESS
5. . . . SEND remote queue name  =====>   RECEIVE queue name
6. . . . SEND record (loop)      =====>   RECEIVE record (loop)
7. . . . SYNCPOINT               =====>
                                <=====    SYNCPOINT
8. . . . FREE SESSION                     RETURN
9. . . . DELETE TS queue
         RETURN
```

1. The user's input values are received and are validated.
2. If a local queue name is not supplied, a queue is constructed.
3. The front-end transaction allocates a conversation and acquires its name from the EIB.
4. The back-end transaction is initiated, using a SYNCLEVEL of 2 to allow CICS syncpointing.
5. The name of the remote queue is transmitted to the back-end transaction.
6. Using consecutive sends, the front-end transaction sends one queue record at a time to the back-end transaction, until the end of the queue is reached.

   The back-end transaction receives one record at a time and writes it to the temporary storage queue.
   The end of the transfer is indicated by the EIB settings.
7. When all the records have been sent, the front-end transaction issues a syncpoint.
   The back-end transaction, on checking the EIB, does the same.
8. The front-end transaction frees the session.
   The back-end transaction terminates (thus freeing the session) when the EIB shows FREE.
9. Finally, the front-end transaction deletes the local temporary storage queue and terminates.

Figure 117. Sample 1: temporary storage queue transfer — overall design

Source listing of sample 1 front-end transaction (DFH$IQXL)

```
DFH$IQXL TITLE 'CICS INTERCOMMUNICATION SAMPLE - QUEUE TRANSFER - LOCAL*
               PROCESSING'
***********************************************************************
*                                                                     *
*                    CICS SAMPLE PROGRAM IQXL                         *
*                     SYNCHRONOUS PROCESSING                          *
*                        'QUEUE TRANSFER'                             *
*                        LOCAL TRANSACTION                            *
*                                                                     *
***********************************************************************
*                                                                     *
*    INPUTS TO THIS PROGRAM             LOCAL TS Q NAME               *
*    ( FROM USERS SCREEN  )             REMOTE TS Q NAME              *
*                                       REMOTE SYSTEM ID              *
*                                                                     *
*    OUTPUTS FROM THIS PROGRAM          REMOTE TS Q NAME              *
*    ( PASSED TO TRANSACTION IQXR )     TS Q RECORDS                  *
*                                                                     *
***********************************************************************
DFH$IQXL CSECT
*
***********************************************************************
*        SEND INITIAL REQUEST MENU TO USERS SCREEN                    *
***********************************************************************
RESETMAP DS    0H
         XC    SPMAPAO(SPMAPAL),SPMAPAO      CLEAR MAP STORAGE
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGX') MAPONLY        *
               ERASE
*     *
***********************************************************************
*        RECEIVE USERS REQUIREMENTS FROM THE SCREEN                   *
***********************************************************************
RETRY    DS    0H
         EXEC CICS RECEIVE MAP('SPMAPA') MAPSET('DFH$IGX')             *
               RESP(RESP)
*
         CLI   EIBAID,DFHPF3            PF3 PRESSED ?
         BE    USEREXIT                 YES, GO END TRANSACTION
         CLC   RESP,DFHRESP(MAPFAIL)    MAP FAILED ?
         BE    RESETMAP                 YES, GO TRY AGAIN
         CLC   RESP,DFHRESP(NORMAL)     NORMAL RESPONSE ?
         BNE   BADRCV                   NO, GO TERMINATE TRANSACTION
         CLI   EIBAID,DFHENTER          ENTER PRESSED
         BNE   INVMSG                   NO, GO TO ERROR MSG ROUTINE
*     *
         CLC   RQNML,=H'0'              REMOTE Q NAME ENTERED ?
         BE    NORQNAME                 NO, GO REQUEST Q NAME
         CLC   RQNMI,BLANKS             REMOTE Q NAME BLANK ?
         BE    NORQNAME                 YES, GO REQUEST Q NAME
*
         CLC   RSYSL,=H'0'              REMOTE SYS ID ENTERED ?
         BE    NOSYSID                  NO, GO REQUEST SYS ID
         CLC   RSYSI,BLANKS             REMOTE SYS ID BLANK ?
         BE    NOSYSID                  YES, GO REQUEST SYS ID
*
```

Figure 118 (Part 1 of 8). Sample 1: temporary storage queue transfer — front-end transaction (DFH$IQXL)

```
         CLC   QNAML,=H'0'             LOCAL TS Q NAME SUPPLIED ?
         BE    NOQNMGVN                NO, SO GO CREATE TS Q
         CLC   QNAMI,BLANKS            LOCAL TS Q NAME SUPPLIED ?
         BE    NOQNMGVN                NO, SO GO CREATE TS Q
*
         MVC   TSQNAME,QNAMI           SAVE SUPPLIED TS Q NAME
*                                      CLEAR SCREEN
         EXEC CICS SEND FROM(MSGWA) LENGTH(0) ERASE
         B     GOTTSQNM                GO READ FIRST RECORD FROM TS Q
*
NOQNMGVN DS    0H
         BAL   R7,BUILDTSQ             BUILD TS Q WITH DEFAULT NAME
*                                     (N.B. WILL NOT RETURN HERE IF
*                                      ANY PROBLEM BUILDING TS QUEUE.)
*
***********************************************************************
*        READ FIRST RECORD IN Q TO ENSURE Q SPECIFIED EXISTS.         *
***********************************************************************
GOTTSQNM DS    0H
         LA    R6,1                   INITIALIZE RECORD ITEM
         STH   R6,ITEMNUM             NUMBER FOR 'READ Q'.
         MVC   TSLEN,=H'80'           SET A MAX ON THE REC LENGTH.
*      *
         EXEC CICS READQ TS QUEUE(TSQNAME) INTO(TSRECORD)              *
               LENGTH(TSLEN) ITEM(ITEMNUM) RESP(RESP)
*
         CLC   RESP,DFHRESP(QIDERR)   TS Q KNOWN TO CICS ?
         BE    BADQNM                 NO, SO GO INFORM USER
         CLC   RESP,DFHRESP(NORMAL)   RECORD READ OK ?
         BNE   BADREADQ               NO, GO TERMINATE TASK
*
***********************************************************************
*        SEND 'ALLOCATING SESSION AND TRANSFERRING Q'                 *
*        MESSAGE TO USERS SCREEN.                                     *
***********************************************************************
         EXEC CICS SEND MAP('SPMAPC') MAPSET('DFH$IGX')
*      *
***********************************************************************
*        ACQUIRE A SESSION TO THE REMOTE SYSTEM                       *
***********************************************************************
         EXEC CICS ALLOCATE SYSID(RSYSI) RESP(RESP)
*
         CLC   RESP,DFHRESP(SYSIDERR) SYSTEM ID ERROR ?
         BE    SYSERR                 YES, GO CHECK REASON
         CLC   RESP,DFHRESP(NORMAL)   ACQUIRED SESSION OK ?
         BNE   BADALLOC               NO, GO TERMINATE TASK
         MVC   ATCHSESS,EIBRSRCE      SAVE NAME OF CONVERSATION
*
***********************************************************************
*        INITIATE THE ACQUIRED CONVERSATION                           *
***********************************************************************
         EXEC CICS CONNECT PROCESS CONVID(ATCHSESS)                    *
               PROCLENGTH(4) PROCNAME('IQXR') SYNCLEVEL(2)
*
***********************************************************************
*        SEND THE Q NAME TO BE ASSIGNED, TO THE REMOTE TRANSACTION    *
***********************************************************************
         EXEC CICS SEND CONVID(ATCHSESS) FROM(RQNMI) LENGTH(8)
*
```

Figure 118 (Part 2 of 8). Sample 1: temporary storage queue transfer – front-end transaction (DFH$IQXL)

```
***********************************************************************
*       SEND EACH RECORD ( VIA THE SESSION ) TO                        *
*       THE REMOTE INITIATED TRANSACTION                               *
***********************************************************************
SENDLOOP DS    0H
         EXEC CICS SEND CONVID(ATCHSESS) FROM(TSRECORD)                *
               LENGTH(TSLEN)
*
***********************************************************************
*       READ NEXT RECORD FROM THE TEMP. STORAGE QUEUE                  *
***********************************************************************
         LA    R6,1(R6)                 INCREMENT AND STORE THE...
         STH   R6,ITEMNUM               ...RECORD ITEM NUMBER.
         MVC   TSLEN,=H'80'             RESET MAX REC LENGTH.
*
         EXEC CICS READQ TS QUEUE(TSQNAME) INTO(TSRECORD)              *
               LENGTH(TSLEN) ITEM(ITEMNUM) RESP(RESP)
*
         CLC   RESP,DFHRESP(NORMAL)     GOT RECORD ?
         BE    SENDLOOP                 YES, GO SEND IT TO REM. TRAN.
         CLC   RESP,DFHRESP(ITEMERR)    ALL RECORDS READ ?
         BNE   BADREADQ                 NO, BAD RESPONSE
*
***********************************************************************
*       THE QUEUE HAS NOW BEEN SENT, SO TAKE A SYNCPOINT               *
***********************************************************************
         EXEC CICS SYNCPOINT
*
***********************************************************************
*       FREE THE SESSION AS SOON AS POSSIBLE, AS NO LONGER NEEDED      *
***********************************************************************
         EXEC CICS FREE CONVID(ATCHSESS)
*
***********************************************************************
*       DELETE THE TS QUEUE                                            *
***********************************************************************
         EXEC CICS DELETEQ TS QUEUE(TSQNAME)
*
***********************************************************************
*       SEND 'TRANSFER COMPLETE, TRANSACTION SUCCESSFUL'               *
*       MESSAGE TO USER'S SCREEN.                                      *
***********************************************************************
         EXEC CICS SEND MAP('SPMAPD') MAPSET('DFH$IGX')
*
***********************************************************************
*       RETURN TO CICS                                                 *
***********************************************************************
EXIT     DS    0H
         EXEC CICS RETURN
         EJECT
***********************************************************************
*       ROUTINES TO HANDLE INSUFFICIENT INPUT                          *
***********************************************************************
NORQNAME DS    0H               *** NO REMOTE TS Q NAME SUPPLIED ***
         MVC   ERRMSGO(L'NORQNMSG),NORQNMSG  MOVE ERROR MSG. TO MAP
         MVC   RQNML,=H'-1'                  SET CURSOR POSITION
         B     RESEND                        GO RESEND MAP
*
```

Figure 118 (Part 3 of 8). Sample 1: temporary storage queue transfer — front-end transaction (DFH$IQXL)

```
NOSYSID  DS    0H                   *** NO REMOTE SYSTEM ID SUPPLIED ***
         MVC   ERRMSGO(L'NOSYSMSG),NOSYSMSG  MOVE ERROR MSG. TO MAP
         MVC   RSYSL,=H'-1'                  SET CURSOR POSITION
*
RESEND   DS    0H                        RESEND MAIN MAP WITH ERR. MSG.
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGX')                *
               DATAONLY CURSOR
*        *
         B     RETRY                         GO CHECK NEW INPUT
*
*
***********************************************************************
*        SUBROUTINE TO BUILD A TS QUEUE WITH A DEFAULT Q NAME          *
*        OF 'TSQLCL'.                                                  *
*                                                                      *
*        FIRST SEND 'CONSTRUCTING Q' MESSAGE TO USERS SCREEN           *
***********************************************************************
BUILDTSQ DS    0H
         EXEC CICS SEND MAP('SPMAPB') MAPSET('DFH$IGX') ERASE
*        *
         MVC   TSRECORD,INITTSR        INITIALIZE TS RECORD AREA
         MVC   TSLEN,=AL2(L'INITTSR)   INITIALIZE TS RECORD LENGTH
         LA    R5,5                    INITIALIZE LOOP COUNT
*
TSLOOP   DS    0H
         IC    R4,DIGIT1               INCREMENT AND STORE THE...
         LA    R4,1(R4)                ...RECORD NUMBER DIGIT INTO...
         STC   R4,DIGIT1               ...THE TS RECORD AREA TO BE...
         STC   R4,DIGIT2               ...WRITTEN TO THE TS Q.
*
         EXEC CICS WRITEQ TS QUEUE(DEFAULTQ) FROM(TSRECORD)            *
               LENGTH(TSLEN) NOSUSPEND RESP(RESP)
*
         CLC   RESP,DFHRESP(NOSPACE)   TS Q FULL ?
         BE    NOSPACE                 YES, GO INFORM USER
         CLC   RESP,DFHRESP(NORMAL)    RECORD WRITTEN OK ?
         BNE   BADWRTQ                 NO, GO TERMINATE TRANSACTION
         BCT   R5,TSLOOP               LOOP UNTIL 5 RECS. WRITTEN
*
         MVC   TSQNAME,DEFAULTQ        SAVE TS Q NAME
         BR    R7                      RETURN TO MAIN LINE CODE
*
*
***********************************************************************
*        ROUTINE TO HANDLE LOCAL TS QUEUE FULL CONDITION.              *
***********************************************************************
NOSPACE  DS 0H
         CH    R5,=H'5'                ANY RECORDS WRITTEN ?
         BE    BYPDELQ                 NO, BYPASS DELETEQ
*
         EXEC CICS DELETEQ TS QUEUE(DEFAULTQ)
*
BYPDELQ  DS 0H
         MVC   ERRMSGO,TSFULMSG        MOVE MESSAGE TO MAP
         MVC   QNAML,=H'-1'            SET CURSOR POSITION
         B     SENDAGN                 GO RESEND ORIGINAL MAP
*
```

Figure 118 (Part 4 of 8). Sample 1: temporary storage queue transfer – front-end transaction (DFH$IQXL)

```
*
***********************************************************************
*        ROUTINE TO HANDLE UNKNOWN TS QUEUE NAME                      *
***********************************************************************
BADQNM   DS    0H
         MVC   MSGWA,QIDERMSG      MOVE SKELETON MSG. TO WORK AREA
         MVC   QNAMWA,TSQNAME      INSERT TS Q NAME INTO MSG.
         MVC   QNAML,=H'-1'        SET CURSOR POSITION
         B     MOVEM               GO RESEND MAIN MAP
*
***********************************************************************
*        ROUTINE TO HANDLE SYSIDERR RESPONSE FROM ALLOCATE COMMAND    *
***********************************************************************
SYSERR   DS    0H
         CLI   EIBRCODE+1,X'08'    LINK OUT OF SERVICE ?
         BNE   UNKNOWN             NO, GO SET UP 'UNKNOWN' MSG.
         MVC   MSGWA,OUTSVMSG      MOVE SKELETON MSG. TO WORK AREA
         MVC   SYSWA1,RSYSI        INSERT SYSID INTO MESSAGE.
         B     SETCURS             GO RESEND ORIGINAL SCREEN
*
UNKNOWN  DS    0H                  UNKNOWN SYSTEM NAME
         MVC   MSGWA,UNKWNMSG      MOVE SKELETON MSG. TO WORK AREA
         MVC   SYSWA2,RSYSI        INSERT SYSID INTO MESSAGE.
*
SETCURS  DS    0H
         MVC   RSYSL,=H'-1'        SET CURSOR POSITION
*
***********************************************************************
*        ROUTINE TO DISPLAY THE ORIGINAL MAP WITH AN ERROR MESSAGE    *
***********************************************************************
MOVEM    DS    0H
         MVC   ERRMSGO,MSGWA       MOVE ERROR MESSAGE TO MAP
*
SENDAGN  DS    0H
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGX')                *
               CURSOR ERASE WAIT
*
         B    RETRY
*
SENDAGN1 DS    0H
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGX')                *
               CURSOR ERASE WAIT
*      *
         EXEC CICS RECEIVE
*      *
         B    RESETMAP
*
***********************************************************************
*        ROUTINES FOR UNRECOVERABLE ERRORS                            *
***********************************************************************
BADRCV   DS    0H          BAD RESPONSE FROM 'EXEC CICS RECEIVE'
         MVC   EMBRCVM,BRMTXT           SET UP MESSAGE
         MVC   EMLEN,BRMTLEN            SET MESSAGE LENGTH
         B     COMNERR                  GO SET UP COMMON PART OF MSG.
*
```

Figure 118 (Part 5 of 8). Sample 1: temporary storage queue transfer — front-end transaction (DFH$IQXL)

```
BADWRTQ  DS    0H              BAD RESPONSE FROM 'EXEC CICS WRITEQ TS'
         MVC   EMBWTQ,BWQTXT             SET UP MESSAGE
         MVC   EMLEN,BWQTLEN             SET MESSAGE LENGTH
         B     COMNERR                   GO SET UP COMMON PART OF MSG.
*
BADREADQ DS    0H              BAD RESPONSE FROM 'EXEC CICS READQ TS'
         MVC   EMBRDQ,BRQTXT             SET UP MESSAGE
         MVC   EMLEN,BRQTLEN             SET MESSAGE LENGTH
         B     COMNERR                   GO SET UP COMMON PART OF MSG.
*
BADALLOC DS    0H              BAD RESPONSE FROM 'EXEC CICS ALLOCATE'
         MVC   EMBALL,BATXT              SET UP MESSAGE
         MVC   EMLEN,BATLEN              SET MESSAGE LENGTH
*
COMNERR  DS    0H
         MVC   EMBRESP,BRTXT             SET UP COMMON PART OF MSG.
         B     ENDTRAN                   GO COMPLETE MSG. & TERM. TRAN.
*
***********************************************************************
*        HANDLE INVALID PF KEY                                        *
***********************************************************************
INVMSG   DS    0H
         LA    R5,INVALID
         MVC   ERRMSGO,0(R5)
         B     SENDAGN1
*
***********************************************************************
*        HANDLE PF3 PRESSED BY USER                                   *
***********************************************************************
USEREXIT DS    0H
         MVC   EMUSER,USERTXT            SET UP MESSAGE
         MVC   EMLEN,USERTLEN            SET MESSAGE LENGTH
*
***********************************************************************
*        COMPLETE TERMINATION MESSAGE AND SEND IT TO SCREEN           *
***********************************************************************
ENDTRAN  DS    0H
         MVC   EMHEAD,EXITTXT            COMPLETE MESSAGE
*                                   SEND TERMINATATION MSG. TO SCREEN
        EXEC CICS SEND FROM(EXITMSG) LENGTH(EMLEN) ERASE
*
         B     EXIT                      GO TERMINATE TRANSACTION
         EJECT
***********************************************************************
*        CONSTANTS                                                    *
***********************************************************************
*
BLANKS   DC    CL8' '
DEFAULTQ DC    CL8'TSQLCL'
INITTSR  DC    C'000000 SYNCHRONOUS PROCESSING TS Q REC.0'
TSRECLEN DC    AL2(L'INITTSR)
NORQNMSG DC    CL36'PLEASE SUPPLY A REMOTE QUEUE NAME'
NOSYSMSG DC    CL36'PLEASE SUPPLY A REMOTE SYSID'
TSFULMSG DC    CL36'LOCAL TS FULL - PRESS ENTER TO RETRY'
OUTSVMSG DC    CL36'LINK TO .... IS OUT OF SERVICE'
UNKWNMSG DC    CL36'SYSTEM NAME .... IS NOT KNOWN'
QIDERMSG DC    CL36'QUEUE NAME ........ IS NOT KNOWN'
EXITTXT  DC    C'IQXL TRANSACTION TERMINATED BY '
```

Figure 118 (Part 6 of 8). Sample 1: temporary storage queue transfer -- front-end transaction (DFH$IQXL)

```
USERTXT  DC    C'USER'
BRTXT    DC    C'BAD RESPONSE TO '
BRMTXT   DC    C'RECEIVE MAP COMMAND'
BWQTXT   DC    C'WRITEQ TS COMMAND'
BRQTXT   DC    C'READQ TS COMMAND'
BATXT    DC    C'ALLOCATE COMMAND'
INVALID  DC    C'INVALID KEY - ENTER TO CONTINUE     '
USERTLEN DC    AL2(L'EXITTXT+L'USERTXT)
BRMTLEN  DC    AL2(L'EXITTXT+L'BRTXT+L'BRMTXT)
BWQTLEN  DC    AL2(L'EXITTXT+L'BRTXT+L'BWQTXT)
BRQTLEN  DC    AL2(L'EXITTXT+L'BRTXT+L'BRQTXT)
BATLEN   DC    AL2(L'EXITTXT+L'BRTXT+L'BATXT)
*
         LTORG
         EJECT
***********************************************************************
*        EQUATES                                                      *
***********************************************************************
         COPY  DFHAID              ATTENTION ID DEFINITIONS
*
         DFHREGS                   REGISTER EQUATES
         EJECT
***********************************************************************
*        WORKING STORAGE                                              *
***********************************************************************
DFHEISTG DSECT
         COPY  DFH$IGX             MAPSET DEFINITIONS
*
RESP     DS    F                   RESPONSES TO CICS COMMANDS
TSQNAME  DS    CL8                 TS QUEUE NAME
ITEMNUM  DS    H                   RECORD ITEM NUMBER
ATCHSESS DS    CL4                 ATTACHED SESSION ID
*
TSLEN    DS    H                   LENGTH OF TS RECORD
TSRECORD DS    0CL80               I/O AREA FOR TS RECORD
         DS    CL5
DIGIT1   DS    C
         DS    CL33
DIGIT2   DS    C
         DS    CL40
*
MSGWA    DS    CL36                WORK AREA FOR BUILDING ERROR MSG
         ORG   MSGWA+8
SYSWA1   DS    CL4
         ORG   MSGWA+12
SYSWA2   DS    CL4
         ORG   MSGWA+11
QNAMWA   DS    CL8
         ORG
*
```

Figure 118 (Part 7 of 8). Sample 1: temporary storage queue transfer – front-end transaction (DFH$IQXL)

```
EMLEN    DS    H                   LENGTH OF TRAN. TERMINATION MSG.
EXITMSG  DS    0CL66               WORK AREA FOR TRAN. TERMINATION MSG
EMHEAD   DS    CL31
EMUSER   DS    0CL4
EMBRESP  DS    CL16
EMBALL   DS    0CL16
EMBRDQ   DS    0CL16
EMBWTQ   DS    0CL17
EMBRCVM  DS    CL19
*
         END
```

Figure 118 (Part 8 of 8). Sample 1: temporary storage queue transfer – front-end transaction (DFH$IQXL)

Source listing of sample 1 back-end transaction (DFH$IQXR)

```
DFH$IQXR TITLE 'CICS INTERCOMMUNICATION SAMPLE - QUEUE TRANSFER - REMOT*
               E PROCESSING'
***********************************************************************
*                                                                     *
*                      CICS SAMPLE PROGRAM IQXR                       *
*                       SYNCHRONOUS PROCESSING                        *
*                          'QUEUE TRANSFER'                           *
*                         REMOTE TRANSACTION                          *
*                                                                     *
***********************************************************************
*                                                                     *
*      INPUTS TO THIS PROGRAM            NAME OF TRANSFERRED Q        *
*      ( FROM PROGRAM IQXL  )            TS Q RECORDS                 *
*                                                                     *
*      OUTPUTS FROM THIS PROGRAM         NONE                         *
*                                                                     *
***********************************************************************
DFH$IQXR CSECT
*
***********************************************************************
*        RECEIVE THE QUEUE NAME, PASSED TO THIS TRANSACTION           *
*        BY IQXL. SAVE EIB FIELDS AND THEN TEST THEM TO SEE           *
*        WHAT TO DO NEXT.                                             *
***********************************************************************
         EXEC CICS RECEIVE INTO(QNAME)
*
         MVC   XSYNC,EIBSYNC       SAVE EIB FIELDS
         MVC   XFREE,EIBFREE
         MVC   XRECV,EIBRECV
*
EIBTEST  DS    0H
         CLI   XSYNC,X'FF'         SYNCPOINT REQUIRED ?
         BNE   TESTFREE            NO, GO TEST IF FREE REQUIRED
*                                  ISSUE SYNCPOINT
         EXEC CICS SYNCPOINT
*
TESTFREE DS    0H
         CLI   XFREE,X'FF'         FREE REQUIRED ?
         BE    TERMNATE            YES, GO ISSUE FREE
*                                      AND TERMINATE TRANSACTION
*
         CLI   XRECV,X'FF'         RECEIVE REQUIRED ?
         BNE   TERMNATE            NO, GO TERMINATE TRANSACTION
*
*                                  RECEIVE TS RECORD PASSED BY IQXL
         EXEC CICS RECEIVE SET(R9) LENGTH(TSLEN)
*
         MVC   XSYNC,EIBSYNC       SAVE EIB FIELDS
         MVC   XFREE,EIBFREE
         MVC   XRECV,EIBRECV
         CLC   TSLEN,=H'0'         ANY DATA RECEIVED ?
         BE    EIBTEST             NO, SO GO TEST EIB IMMEDIATELY
*
*                DATA WAS RECEIVED, SO WRITE RECORD TO NAMED QUEUE
         EXEC CICS WRITEQ TS QUEUE(QNAME) FROM(0(R9))                  *
               LENGTH(TSLEN)
*
```

Figure 119 (Part 1 of 2). Sample 1: temporary storage queue transfer — back-end transaction (DFH$IQXR)

```
         B     EIBTEST              GO AND TEST EIB
*
*
***********************************************************************
*        TRANSACTION TERMINATION                                      *
***********************************************************************
TERMNATE DS    0H
         EXEC CICS FREE
*
         EXEC CICS RETURN
*
*
         LTORG
*
         DFHREGS                    REGISTER EQUATES
*
***********************************************************************
*        WORKING STORAGE                                              *
***********************************************************************
DFHEISTG DSECT
XSYNC    DS    C                    IF SET, EXECUTE SYNCPOINT
XFREE    DS    C                    IF SET, FREE TERMINAL / LU
XRECV    DS    C                    IF SET, EXECUTE RECEIVE
*
QNAME    DS    CL8
TSLEN    DS    H
*
         END
```

Figure 119 (Part 2 of 2). Sample 1: temporary storage queue transfer — back-end transaction (DFH$IQXR)

BMS mapset for sample 1 (DFH$IMX)

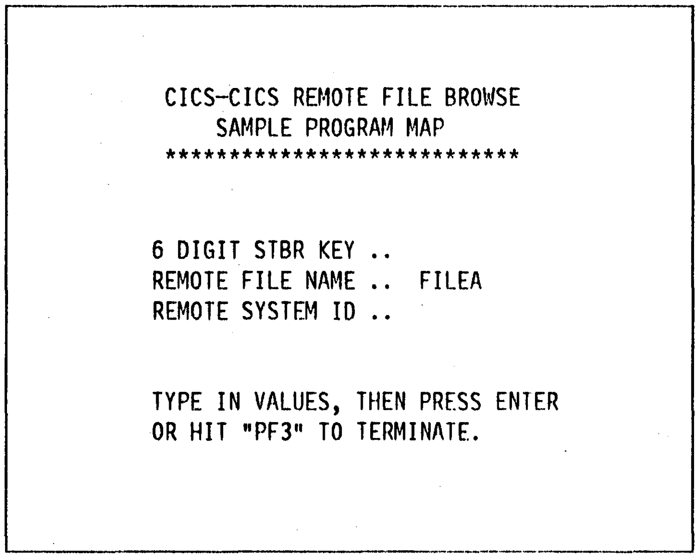
```
DFH$IMX  TITLE 'INTERCOMMUNICATION SAMPLE - LOCAL TO REMOTE TS Q XFER -*
               MAPSET'
         AIF   ('&SYSPARM' EQ 'DSECT').SKIPSD - OS SMP REQUIRES CSECT
DFH$IGXC CSECT
.SKIPSD  ANOP ,
DFH$IGX  DFHMSD TYPE=&SYSPARM,MODE=INOUT,                              *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES,EXTATT=NO
*
SPMAPA   DFHMDI SIZE=(24,80),CTRL=(ALARM,FREEKB)
         DFHMDF POS=(4,18),LENGTH=25,ATTRB=(ASKIP,BRT),                *
               INITIAL=' CICS-CICS Q TRANSFER '
         DFHMDF POS=(5,18),LENGTH=25,ATTRB=(ASKIP,BRT),                *
               INITIAL=' SAMPLE PROGRAM MAP   '
         DFHMDF POS=(6,15),LENGTH=29,ATTRB=(ASKIP),                    *
               INITIAL='*****************************'
         DFHMDF POS=(10,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='LOCAL TS Q NAME ...'
QNAM     DFHMDF POS=(10,25),LENGTH=8,ATTRB=(UNPROT,BRT,IC,FSET)
         DFHMDF POS=(10,34),ATTRB=(ASKIP),LENGTH=1
         DFHMDF POS=(12,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='REMOTE TS Q NAME ..'
RQNM     DFHMDF POS=(12,25),LENGTH=8,ATTRB=(UNPROT,BRT,FSET),          *
               INITIAL='REMOTEQ '
         DFHMDF POS=(12,34),ATTRB=(ASKIP),LENGTH=1
         DFHMDF POS=(14,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='REMOTE SYSTEM ID ..'
RSYS     DFHMDF POS=(14,25),LENGTH=4,ATTRB=(UNPROT,BRT,FSET)
         DFHMDF POS=(14,30),ATTRB=(PROT),LENGTH=1
ERRMSG   DFHMDF POS=(16,4),LENGTH=36,ATTRB=(ASKIP,BRT),                *
               INITIAL=' '
         DFHMDF POS=(18,4),LENGTH=32,ATTRB=(ASKIP),                    *
               INITIAL='TYPE IN VALUES, THEN PRESS ENTER'
         DFHMDF POS=(19,4),LENGTH=32,ATTRB=(ASKIP),                    *
               INITIAL='OR HIT "PF3" TO TERMINATE.'
*
SPMAPB   DFHMDI SIZE=(24,80)
         DFHMDF POS=(2,4),LENGTH=22,ATTRB=(PROT,BRT),                  *
               INITIAL='CONSTRUCTING TS QUEUE.'
*
SPMAPC   DFHMDI SIZE=(24,80)
         DFHMDF POS=(5,4),LENGTH=22,ATTRB=(PROT,BRT),                  *
               INITIAL='ALLOCATING SESSION AND'
         DFHMDF POS=(6,4),LENGTH=22,ATTRB=(PROT,BRT),                  *
               INITIAL='TRANSFERRING TS QUEUE.'
*
SPMAPD   DFHMDI SIZE=(24,80),CTRL=(ALARM,FREEKB)
         DFHMDF POS=(10,4),LENGTH=20,ATTRB=(PROT,BRT),                 *
               INITIAL='Q TRANSFER COMPLETE,'
         DFHMDF POS=(11,4),LENGTH=23,ATTRB=(PROT,BRT),                 *
               INITIAL='TRANSACTION SUCCESSFUL.'
*
         DFHMSD TYPE=FINAL
*
         END
```

Figure 120. Sample 1: temporary storage queue transfer – BMS mapset (DFH$IMX)

Sample 2 — remote file browse

This sample illustrates the use of distributed transaction processing to browse a remote file. it consists of a front-end transaction (DFH$IFBL), a back-end transaction (DFH$IFBR), and a BMS mapset (DFH$IMB) for the front-end transaction.

The front-end transaction is invoked by the transaction code IFBL, and displays the following menu at the user's terminal:



```
         CICS-CICS REMOTE FILE BROWSE
             SAMPLE PROGRAM MAP
         ****************************


        6 DIGIT STBR KEY ..
        REMOTE FILE NAME ..  FILEA
        REMOTE SYSTEM ID ..


        TYPE IN VALUES, THEN PRESS ENTER
        OR HIT "PF3" TO TERMINATE.
```

The displayed menu has three input fields:

6 DIGIT STBR KEY
specifies the key of the record at which the browse is to start.

REMOTE FILE NAME
specifies the name of the file that is to be browsed. The menu supplies the default name FILEA.

REMOTE SYSTEM ID
specifies the name of the remote system. This name can be the connection name of an LUTYPE6.2 or LUTYPE6.1 link.

Initially, the file is browsed forwards, and four records from the remote file are displayed. Thereafter, the user can choose to browse forwards (PF8), browse backwards (PF7), or terminate the browse (PF3).

Figure 121 on page 359 shows the overall flow of the file browse sample.

```
          Local Transaction              Remote Transaction
            (front-end)                      (back-end)

 1. . . . Get User Requirements
 2. . . . ALLOCATE SYSID( )
          (MVC ATCHSESS,EIBRSRCE)
 3. . . . CONVERSE                =====>  RECEIVE
 4. . . .                                  Process and Buffer Records
 5. . . .                        <=====   SEND
 6. . . .                                 RETURN
 7. . . . FREE Conversation
 8. . . .  Process Input and
           Send to User's Screen
 9. . . .  If More Browsing go to 2
10. . . . RETURN
```

1. The user's input values are received and are validated.
2. The front-end transaction allocates a conversation and acquires its name from the EIB.
3. The file name, the record key, and the browse direction are sent to the back-end transaction.

   Note: The CONVERSE COMMAND is equivalent to SEND, WAIT, RECEIVE.
   This is a "migration mode" command.
   No CONNECT PROCESS command is issued; instead, the remote transaction identifier is sent in the first four bytes of the message.
4. The back-end transaction retrieves four records from the data set and places them in a buffer.
5. The back-end transaction sends the buffered records to the front-end transaction, together with the key of the last record that was retrieved.
6. The back-end transaction terminates, thereby freeing the session.
7. The front-end transaction, after receiving the data, frees the session.
8. The front-end transaction unblocks the records and sends them (or possibly and error message) to the user.
9. If the user wants to browse more records (PF7 or PF8), the process is repeated from step 2.
10. Otherwise, the front-end transaction terminates.

Figure 121. Sample 2: remote file browse — overall design

Source listing of sample 2 front-end transaction (DFH$IFBL)

```
         TITLE 'DFH$IFBL - CICS INTERCOMMUNICATION SAMPLE - REMOTE FILE*
                BROWSE - LOCAL PROCESSING'
DFHEISTG DSECT
*
***********************************************************************
*                                                                     *
*                     CICS SAMPLE PROGRAM IFBL                        *
*                      SYNCHRONOUS PROCESSING                         *
*                       'REMOTE FILE BROWSE'                          *
*                         LOCAL TRANSACTION                           *
*                                                                     *
***********************************************************************
*                                                                     *
*                                                                     *
*        INPUTS TO THIS PROGRAM         START BROWSE KEY              *
*        ( FROM USERS SCREEN )          REMOTE FILE NAME              *
*                                       REMOTE SYSTEM ID              *
*                                                                     *
*        OUTPUTS FROM THIS PROGRAM      START BROWSE KEY              *
*        ( PASSED TO PROGRAM IFBR )     REMOTE FILE NAME              *
*                                       RUN AND DIRECTION FLAG        *
*                                       ( FIRST RUN AND               *
*                                         BROWSE FWD OR BKWD BITS )   *
*                                                                     *
***********************************************************************
*
***********************************************************************
*        STATUS FLAG AND EIB SESSION STORAGE AREA                     *
***********************************************************************
*
XDFEIFLG DS    0CL7
*
XSYNC    DS    C                    IF SET, EXECUTE SYNCPOINT
*
XFREE    DS    C                    IF SET, FREE TERMINAL / LU
*
XRECV    DS    C                    IF SET, EXECUTE RECEIVE
*
XSEND    DS    C                    RESERVED
*
XATT     DS    C                    IF SET, EXECUTE EXTRACT TO
*                                      ACCESS ATTACH HEADER DATA
XEOC     DS    C                    IF SET, END-OF-CHAIN WAS
*                                      RECEIVED WITH DATA
XFMH     DS    C                    IF SET, DATA PASSED TO APPL'N
*                                      CONTAINS FMH - N/A FOR LU6.2
*
***********************************************************************
*        MAP DEFINITIONS FOR MAPSET DFH$IGB                           *
***********************************************************************
*
         COPY  DFH$IGB
         COPY  DFHAID
*
```

Figure 122 (Part 1 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
OUTAREA  DS    0CL19                DATA OUTPUT AREA
TRANOUT  DS    CL4                  TRANSACTION TO INITIATE
FLAG     DS    XL1                  RUN AND DIRECTION FLAG
KEYOUT   DS    CL6                  START BROWSE KEY OUTPUT
BRDSET   DS    CL8                  FILE TO BE BROWSED
*
ATCHSESS DS    CL4                  ATTACHED SESSION ID
MESSAGE  DS    CL36                 OUTPUT MESSAGE
INAREA   DS    CL350                DATA INPUT AREA
RECVDMSG DS    CL20                 RECEIVED MESSAGE TO BE DISPLAYED
INLEN    DS    H                    LENGTH OF TS RECORD
OUTLEN   DS    H                    DATA OUTPUT LENGTH
RECLEN   DS    H                    LENGTH OF A RECVD RECORD
MSGLEN   DS    H                    LENGTH OF OUTPUT MSG
FROMLEN  DS    H                    COMMON SEND LENGTH VARIABLE
CURSOR   DS    H                    CURSOR POSITION ON PAGE
REGSTOR1 DS    F                    LINK REGISTER SAVE AREA 1
RESP     DS    F                    RESPONSES TO CICS COMMANDS
REGSTOR2 DS    F                    LINK REGISTER SAVE AREA 2
SAVEIT EQU     X'01'                FIRST RUN INDICATOR BIT
FORWARD  EQU   X'02'                FORWARD BROWSE INDICATOR BIT
R0       EQU   0
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R14      EQU   14
*
*
FILEA    DSECT
FILEREC  DS    CL86
*
*
PROGSTRT CSECT
*
*
         BAL   R14,GETDATA          RECEIVE CORRECT USER INPUT
*
         BAL   R14,SETUP            INITIALIZATION SECTION
*
         BAL   R14,CLEARSCN         CLEAR USERS SCREEN
*
GETMORE  DS    0H
*
         CH    R7,PAGEFULL          IS PAGE FULL ?
         BNE   NOTFULL              NO....  CONTINUE.
         LA    R7,1                 YES.... POSITION CURSOR
         STH   R7,CURSOR            AT TOP OF SCREEN.
*
         BAL   R14,CLEARSCN         CLEAR USERS SCREEN
*
NOTFULL  DS    0H
*
***********************************************************************
*        OBTAIN CONVERSATION LINK                                     *
***********************************************************************
*
*
```

Figure 122 (Part 2 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
***********************************************************************
*        ACQUIRE SESSION TO REMOTE SYSTEM                             *
***********************************************************************
*
         EXEC CICS ALLOCATE SYSID(RSYSI) RESP(RESP)
         CLC   RESP,DFHRESP(SYSIDERR)       CHECK FOR VALID SYSTEM ID
         BE    SYSERR1
         CLC   RESP,DFHRESP(NORMAL)         CHECK FOR NORMAL RESPONSE
         BNE   ERROR1
*
         MVC   ATCHSESS,EIBRSRCE   SAVE SESSION NAME
*
***********************************************************************
*        CONVERSE WITH REMOTE TRANSACTION ( IMPLICIT SEND/RECEIVE )   *
***********************************************************************
*
         EXEC CICS CONVERSE SESSION(ATCHSESS) FROMLENGTH(OUTLEN)      *
               FROM(OUTAREA) TOLENGTH(INLEN) INTO(INAREA)             *
               RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BE    CONVEROK
         CLC   RESP,DFHRESP(EOC)   EOC RESPONSE OK, ALL ELSE ERROR
         BNE   ERROR1
*
***********************************************************************
*        FREE THE SESSION AND PROCESS THE DATA RECEIVED               *
***********************************************************************
*
CONVEROK DS    0H
         EXEC CICS FREE SESSION(ATCHSESS) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         BAL   R14,DISPDATA        SEND DATA RECEIVED TO SCREEN
*
         BAL   R14,GOAGAIN         ASK USER IF MORE BROWSING REQUIRED
*
         B     GETMORE             LOOP BACK TO RECEIVE MORE RECORDS
*
GETDATA  DS    0H
*
***********************************************************************
*        ROUTINE TO RECEIVE USERS INPUT FROM SCREEN AND VALIDATE IT   *
***********************************************************************
*
MAPFAIL  DS    0H
         XC    SPMAPAI(SPMAPAE-SPMAPAI),SPMAPAI     CLEAR MAP STORAGE
         ST    R14,REGSTOR1                         SAVE LINK REGISTER.
*
***********************************************************************
*        SEND INITIAL REQUEST MENU TO SCREEN                          *
***********************************************************************
*
CLEANUP  DS    0H
*
```

Figure 122 (Part 3 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGB') MAPONLY        *
               ERASE WAIT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
RETRY    DS    0H
*
*
         EXEC CICS RECEIVE MAP('SPMAPA') MAPSET('DFH$IGB')             *
               RESP(RESP)
         CLI   EIBAID,DFHPF3           WAS PF3 PRESSED?
         BE    COMPLETE                .... YES, GO TO COMPLETE
         CLI   EIBAID,DFHCLEAR         WAS CLEAR KEY PRESSED?
         BE    CLEANUP                 .... YES, GO TO CLEANUP
         CLI   EIBAID,DFHENTER         WAS ENTER KEY PRESSED?
         BNE   INVMSG                  IF NOT, GO TO INVMSG
         CLC   RESP,DFHRESP(NORMAL)    WAS NORMAL REPONSE?
         BE    RECEVOK1                ... YES, GO TO RECEVOK1
         CLC   RESP,DFHRESP(EOC)       WAS EOC RESPONSE?
         BNE   ERROR1                  ... NO, ERROR
*
RECEVOK1 DS    0H
         CLI   STRTKEYI,0              START KEY FIELD CHANGED ?
         BE    BADKEY                  NO, SEND ERROR MSG.
         CLC   STRTKEYL,=H'0'          ANY DATA GIVEN ?
         BNE   KEYGVN                  YES, TEST FILE FIELD.
*
BADKEY   DS    0H
         LA    R5,=CL(L'ERRMSGO)'PLEASE SUPPLY A START BROWSE KEY'
         B     RESEND
*
INVMSG   DS    0H
         LA    R5,=CL(L'ERRMSGO)'INVALID KEY ENTERED'
         B     SENDMAP
*
KEYGVN   DS    0H
         CLI   RDSETI,0                FILE FIELD CHANGED ?
         BE    BADDSET                 NO, SEND ERROR MSG.
         CLC   RDSETL,=H'0'            ANY DATA GIVEN ?
         BNE   DSETGVN                 YES, TEST SYSID FIELD.
*
BADDSET  DS    0H
         LA    R5,=CL(L'ERRMSGO)'PLEASE SUPPLY A FILE NAME'
         B     RESEND
*
DSETGVN  DS    0H
         L     R14,REGSTOR1            RESET LINK REGISTER.
         CLI   RSYSI,0                 SYSID FIELD CHANGED ?
         BE    BADSYSID                NO, SEND ERROR MSG.
         CLC   RSYSL,=H'0'             ANY DATA GIVEN ?
         BNER  R14                     YES, ALL FIELDS PRESENT.
*
BADSYSID DS    0H
         LA    R5,=CL(L'ERRMSGO)'PLEASE SUPPLY A REMOTE SYSTEM ID'
*
RESEND   DS    0H                      VALUE NOT GIVEN, INFORM USER.
*
```

Figure 122 (Part 4 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
***********************************************************************
*       DATA RECEIVED WAS INCOMPLETE, SO SEND ERROR MESSAGE           *
*       WITH MAP AND RECEIVE REQUIREMENTS AGAIN                       *
***********************************************************************
*
         XC    SPMAPAI(SPMAPAE-SPMAPAI),SPMAPAI     CLEAR MAP STORAGE.
*
SENDMAP  DS    0H
         MVC   ERRMSGO,0(R5)                        PUT CORRECT MESSAGE
*                                                   IN OUTPUT FIELD.
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGB')               *
               DATAONLY WAIT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         B     RETRY
*
***********************************************************************
*       INITIALIZATION SECTION                                        *
***********************************************************************
*
SETUP    DS    0H
         MVC   RECLEN,=AL2(L'FILEREC)   SET MAX LEN OF EACH RECORD.
         MVC   INLEN,=AL2(L'INAREA)     SET DATA INPUT AREA LENGTH.
         MVC   OUTLEN,=AL2(L'OUTAREA)   SET DATA OUTPUT AREA LENGTH.
         MVC   BRDSET,RDSETI            SAVE FILE RECEIVED.
         MVC   TRANOUT,=C'IFBR'         ASSIGN REM TRANID TO INITIATE.
         MVC   KEYOUT,STRTKEYI          PUT RECVD START KEY IN OUTAREA.
         MVI   FLAG,X'00'               INITIALIZE FLAG BYTE.
         OI    FLAG,SAVEIT              SET SAVE FLAG TO 'SAVE'
         OI    FLAG,FORWARD             SET BROWSE DIRECTION TO FORWARD
         LA    R7,1                     POSITION CURSOR ON FIRST.
         STH   R7,CURSOR                LINE OF SCREEN.
         BR    R14                      RETURN CONTROL.
*
***********************************************************************
*       ROUTINE TO CLEAR THE USERS SCREEN                             *
***********************************************************************
CLEARSCN DS    0H
*
         ST    R14,REGSTOR1             SAVE LINK REGISTER.
*
         EXEC CICS SEND FROM(CURSOR) LENGTH(0) ERASE RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         L     R14,REGSTOR1             RESET LINK REGISTER.
         BR    R14                      RETURN CONTROL.
*
***********************************************************************
*       ROUTINE TO PROCESS AND DISPLAY THE DATA RECEIVED              *
***********************************************************************
DISPDATA DS    0H
*
         ST    R14,REGSTOR1          SAVE LINK REGISTER.
         NI    FLAG,X'FF'-SAVEIT     NEXT CONVERSE WILL NOT
*                                    BE FIRST SO SET SAVE FLAG.
```

Figure 122 (Part 5 of 9). Sample 2: remote file browse – front-end transaction (DFH$IFBL)

```
         LA    R5,INAREA              R5 -> BEGINNING OF DATA RECEIVED.
         MVC   KEYOUT,0(R5)           SAVE LAST KEY READ.
         LA    R5,L'KEYOUT(R5)        R5 -> BEGINNING OF RECORD DATA.
         LH    R6,INLEN               REDUCE THE LENGTH OF DATA
         SH    R6,=Y(L'KEYOUT)        RECEIVED BY LENGTH OF KEY.
         MVC   FROMLEN,=H'78'         TRUNCATE RECORD LENGTH SENT
*                                     TO SCREEN TO ONE LINE (78 CHARS)
NEXTLINE DS    0H
         CH    R6,RECLEN              ANY RECORDS LEFT ?
         BL    ERMSGOUT               NO, MUST BE ERROR MSG.
         BAL   R14,SENDTEXT           SEND RECEIVED RECORD TO SCREEN.
         LH    R7,CURSOR
         LA    R7,1(R7)               INCREMENT OUTPUT LINE NUMBER.
         STH   R7,CURSOR
         SH    R6,RECLEN              DECREMENT R6 BY LEN. OF 1 RECORD.
         LA    R5,L'FILEREC(R5)       INCREMENT INPUT DATA AREA PTR.
         BNZ   NEXTLINE               GET NEXT RECORD FROM INPUT BUFFER
*                                     IF POINTER IS NOT AT ZERO.
*
         EXEC CICS SEND PAGE TRAILER(PFMSG) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         L     R14,REGSTOR1           RESET LINK REGISTER.
         BR    R14                    RETURN CONTROL.
*
ERMSGOUT DS    0H
*
***********************************************************************
*        DATA RECEIVED CONTAINS MESSAGE, SO DISPLAY IT AND            *
*        TERMINATE AS NO DATA LEFT TO PROCESS                         *
***********************************************************************
*
         STH   R6,FROMLEN             SET LENGTH OF DATA TO SEND.
         BAL   R14,SENDTEXT           SEND MESSAGE TO USERS SCREEN.
*
         EXEC CICS SEND PAGE TRAILER(PFMSG) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         L     R14,REGSTOR1           RESET LINK REGISTER
         BR    R14                    RETURN CONTROL
*
***********************************************************************
*        RECEIVE FROM USER NEXT OPERATION REQUIRED BY ISSUING A       *
*        'RECEIVE' FROM THE SCREEN. REQUEST MUST BE PF3, PF7 OR       *
*        PF8, FOR TERMINATION, BROWSE BACK, OR FORWARD RESPECTIVELY.  *
***********************************************************************
GOAGAIN  DS    0H
*
         ST    R14,REGSTOR1         SAVE LINK REGISTER.
*
*
         EXEC CICS RECEIVE RESP(RESP)
         CLI   EIBAID,DFHPF3            WAS PF3 PRESSED?
         BE    COMPLETE                 .... YES, GO TO COMPLETE
         CLI   EIBAID,DFHPF7            WAS PF7 PRESSED?
         BE    BROWBK                   .... YES, GO TO BROWBK
         CLI   EIBAID,DFHPF8            WAS PF8 PRESSED?
```

Figure 122 (Part 6 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
         BE    BROWFD                  .... YES, GO TO BROWFD
         CLC   RESP,DFHRESP(NORMAL)    WAS NORMAL RESPONSE?
         BE    RECEVOK2                ... YES, GO TO RECEVOK2
         CLC   RESP,DFHRESP(EOC)       WAS EOC RESPONSE?
         BNE   ERROR1                  ... NO, ERROR
*
***********************************************************************
*        AN INVALID KEY WAS RECEIVED SO SEND MESSAGE                  *
*        WITH MAP AND TERMINATE.                                      *
***********************************************************************
*
RECEVOK2 DS    0H
         LA    R5,=CL(L'ERRMSGO)'INVALID KEY - TRANSACTION TERMINATED'
         MVC   ERRMSGO,0(R5)                        PUT CORRECT MESSAGE
*                                                   IN OUTPUT FIELD.
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IGB')                *
               ERASE WAIT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         B     TERMNATE
*
BROWBK   DS    0H
         NI    FLAG,X'FF'-FORWARD   BROWSE BACK REQUESTED, SO SET
*                                   DIRECTION FLAG.
         L     R14,REGSTOR1         RESET LINK REGISTER.
         BR    R14                  RETURN CONTROL.
*
BROWFD   DS    0H
         OI    FLAG,FORWARD         BROWSE FORWARD REQUESTED, SO SET
*                                   DIRECTION FLAG.
         L     R14,REGSTOR1         RESET LINK REGISTER.
         BR    R14                  RETURN CONTROL.
*
***********************************************************************
*        ROUTINE TO SEND RECEIVED DATA TO THE USERS SCREEN            *
***********************************************************************
*
SENDTEXT DS    0H
*
         ST    R14,REGSTOR2         SAVE LINK REGISTER.
*
         EXEC CICS SEND TEXT FROM(0(R5)) LENGTH(FROMLEN)               *
               JUSTIFY(CURSOR) ACCUM FREEKB ALARM RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         L     R14,REGSTOR2         RESET LINK REGISTER.
         BR    R14                  RETURN CONTROL.
*
***********************************************************************
*        PREPARE CORRECT SYSERR ERROR MESSAGE TO SEND TO USER.        *
***********************************************************************
*
SYSERR1  DS    0H
         CLI   EIBRCODE+1,X'0C'     SYSID ERROR ?
         BE    UNKNOWN
```

Figure 122 (Part 7 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
         CLI   EIBRCODE+1,X'08'    LINK OUT OF SERVICE ?
         BE    OUTSERV
*
NOLINK   DS    0H
         LA    R5,NOLNKMSG         ADDRESS NO LINK ERROR MSG.
         MVC   SYS1,RSYSI          INSERT SYSID INTO MESSAGE.
         B     SENDERR
*
UNKNOWN  DS    0H
         CLI   EIBRCODE+2,X'04'    SYSID NOT A REMOTE SYSID ?
         BE    NOTRENT
         LA    R5,UNKWNMSG         ADDRESS UNKNOWN SYS ERROR MSG.
         MVC   SYS2,RSYSI          INSERT SYSID INTO MESSAGE.
         B     SENDERR
*
OUTSERV  DS    0H
         LA    R5,OUTSVMSG         ADDRESS OUT OF SERVICE ERROR MSG.
         MVC   SYS3,RSYSI          INSERT SYSID INTO MESSAGE.
         B     SENDERR
*
NOTRENT  DS    0H
         LA    R5,NOTRMSG          ADDRESS NOT REMODE SYSID ERRMSG.
         MVC   SYS4,RSYSI          INSERT SYSID INTO MESSAGE.
         B     SENDERR
*
***********************************************************************
*        MESSAGE ASSIGNMENT IF CONDITION ERROR RAISED                 *
***********************************************************************
*
ERROR1   DS    0H
         LA    R5,ERRORMSG         ADDRESS ERROR MSG.
         B     SENDERR
*
***********************************************************************
*        PREPARE ERROR MESSAGE, SEND ERROR MAP TO USER THEN TERMINATE *
***********************************************************************
*
SENDERR DS     0H
         MVI   OUTMSGO,C' '           CLEAR ERROR MESSAGE OUTPUT AREA.
         MVC   OUTMSGO+1(L'OUTMSGO),OUTMSGO
         LH    R6,0(R5)               PUT MESSAGE LENGTH IN R6.
         STH   R6,MSGLEN
         BCTR  R6,R0
         EX    R6,MVCMSG              PREPARE MESSAGE TO BE SENT.
*
         EXEC CICS SEND MAP('SPMAPE') MAPSET('DFH$IGB') ERASE WAIT     *
               RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
         B     TERMNATE               TERMINATE TRANSACTION.
MVCMSG   MVC   OUTMSGO(0),L'MSGLEN(R5)
*
TERMNATE DS    0H
*
```

Figure 122 (Part 8 of 9). Sample 2: remote file browse — front-end transaction (DFH$IFBL)

```
***********************************************************************
*        TRANSACTION TERMINATION.                                     *
***********************************************************************
         EXEC CICS SEND CONTROL FREEKB RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
         EXEC CICS RETURN
*
COMPLETE DS    0H
*
         EXEC CICS SEND TEXT FROM(TERMMSG) LENGTH(LTERMMSG)            *
               FREEKB ERASE RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
         EXEC CICS RETURN
*
PAGEFULL DC    H'21'               PAGE FULL VARIABLE.
PFMSG    DS    0CL50               TRAILER MESSAGE
         DC    H'46'               TO BE PLACED AT
         DC    C' '                BOTTOM OF 'SEND
         DS    C                   PAGE' COMMAND.
TRLDATA  DC    CL46'PF7=BROWSE BACK  PF8=BROWSE FWD  PF3=TERMINATE'
*
***********************************************************************
*        ERROR MESSAGE STORAGE DEFINITIONS                            *
***********************************************************************
*
NOLNKMSG DC    AL2(L'MSG1)
MSG1     DC    C'UNABLE TO ESTABLISH LINK TO ....'
         ORG   MSG1+28
SYS1     DS    CL4
         ORG   ,
UNKWNMSG DC    AL2(L'MSG2)
MSG2     DC    C'SYSTEM NAME .... IS NOT KNOWN'
         ORG   MSG2+12
SYS2     DS    CL4
         ORG   ,
OUTSVMSG DC    AL2(L'MSG3)
MSG3     DC    C'LINK TO .... IS OUT OF SERVICE'
         ORG   MSG3+8
SYS3     DS    CL4
         ORG   ,
NOTRMSG  DC    AL2(L'MSG4)
MSG4     DC    C'.... IS NOT A REMOTE SYSTEM NAME'
         ORG   MSG4
SYS4     DS    CL4
         ORG   ,
ERRORMSG DC    AL2(L'MSG5)
MSG5     DC    C'AN ERROR HAS OCCURRED'
*
TERMMSG  DC    C'TRANSACTION TERMINATED'
LTERMMSG DC    AL2(L'TERMMSG)
         END
```

Figure 122 (Part 9 of 9). Sample 2: remote file browse – front-end transaction (DFH$IFBL)

Source listing of sample 2 back-end transaction (DFH$IFBR)

```
         TITLE 'DFH$IFBR - CICS INTERCOMMUNICATION SAMPLE - REMOTE FILE*
                BROWSE - REMOTE PROCESSING'
DFHEISTG DSECT
*
***********************************************************************
*                                                                     *
*                      CICS SAMPLE PROGRAM IFBR                       *
*                       SYNCHRONOUS PROCESSING                        *
*                        'REMOTE FILE BROWSE'                         *
*                         REMOTE TRANSACTION                          *
*                                                                     *
***********************************************************************
*                                                                     *
*       INPUTS TO THIS PROGRAM         START BROWSE KEY               *
*       ( RECEIVED FROM IFBL )         FILE TO BROWSE                 *
*                                      BROWSE DIRECTION REQD.         *
*                                      RUN FLAG ( FIRST RUN BIT )     *
*                                                                     *
*       OUTPUTS FROM THIS PROGRAM      UP TO 4 RECORDS READ FROM      *
*       ( PASSED BACK TO IFBL )        REQUIRED FILE, OR UP TO        *
*                                      3 AND AN ERROR MESSAGE         *
*                                                                     *
***********************************************************************
*
***********************************************************************
*        STATUS FLAG AND EIB SESSION STORAGE AREA                     *
***********************************************************************
*
XDFEIFLG DS    0CL7
*
XSYNC    DS    C                    IF SET, EXECUTE SYNCPOINT
*
XFREE    DS    C                    IF SET, FREE TERMINAL / LU
*
XRECV    DS    C                    IF SET, EXECUTE RECEIVE
*
XSEND    DS    C                    RESERVED
*
XATT     DS    C                    IF SET, EXECUTE EXTRACT TO
*                                      ACCESS ATTACH HEADER DATA
XEOC     DS    C                    IF SET, END-OF-CHAIN WAS
*                                      RECEIVED WITH DATA
XFMH     DS    C                    IF SET, DATA PASSED TO APPL'N
*                                      CONTAINS FMH - N/A FOR LU6.2
INAREA   DS    0CL20                DATA OUTPUT AREA
TRANIN   DS    CL4                  THIS TRANSACTION ID
RUNDNFLG DS    XL1                  RUN AND DIRECTION INDICATOR
KEYIN    DS    CL6                  START BROWSE KEY RECEIVED
DSETIN   DS    CL8                  FILE TO BE BROWSED
OUTAREA  DS    CL350                DATA INPUT AREA
INLEN    DS    H                    INPUT DATA LENGTH
OUTLEN   DS    H                    OUTPUT DATA LENGTH
RESP     DS    F                    RESPONSES TO CICS COMMANDS
SAVEIT EQU     X'01'                FIRST RUN INDICATOR
FORWARD  EQU   X'02'                BROWSE DIRECTION INDICATOR
```

Figure 123 (Part 1 of 6). Sample 2: remote file browse — back-end transaction (DFH$IFBR)

```
R0       EQU   0
R1       EQU   1
R2       EQU   2
R3       EQU   3
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
R10      EQU   10
R11      EQU   11
R12      EQU   12
R13      EQU   13
R14      EQU   14
R15      EQU   15
*
FILEA    DSECT
FILEREC  DS    CL86
*
PROGSTRT CSECT
*
*
         LA    R4,1                SET COUNTER TO 1.
         LA    R5,OUTAREA          R5 <- ADDRS OF OUTPUT BUFFER.
         LA    R5,L'KEYIN(R5)      MAKE ROOM FOR LAST RIDFLD USED.
         USING FILEA,R5            DECLARE FILE BASE REGISTER.
         LA    R6,L'INAREA         SET INLEN TO LENGTH OF
         STH   R6,INLEN            INPUT DATA AREA.
*
***********************************************************************
*        RECEIVE DATA PASSED FROM 'IFBL'                              *
***********************************************************************
*
         EXEC CICS RECEIVE INTO(INAREA) LENGTH(INLEN) RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    CHECK FOR NORMAL RESPONSE
         BE    RECEIVOK
         CLC   RESP,DFHRESP(EOC)       CHECK FOR EOC RESPONSE
         BNE   ERROR1
RECEIVOK DS    0H
         MVC   XDFEIFLG,EIBSYNC    SAVE EIB VALUES
*
EIBTEST  DS    0H
*
TESTSYNC DS    0H
         CLI   XSYNC,X'FF'
         BNE   TESTFREE
*
***********************************************************************
*        SYNCPOINT SET, SO ISSUE A SYNCPOINT REQUEST                  *
***********************************************************************
*
         EXEC CICS SYNCPOINT RESP(RESP)
         CLC  RESP,DFHRESP(NORMAL)
         BNE ERROR1
*
TESTFREE DS    0H
```

Figure 123 (Part 2 of 6). Sample 2: remote file browse — back-end transaction (DFH$IFBR)

```
         CLI   XFREE,X'FF'
         BNE   TESTRECV            FREE NOT ON SO TEST RECV
*
***********************************************************************
*        FREE SET, SO ISSUE A FREE REQUEST                            *
***********************************************************************
*
         EXEC CICS FREE RESP(RESP)
         CLC  RESP,DFHRESP(NORMAL)
         BNE  ERROR1
*
         B     IMMRET
*
TESTRECV DS    0H
         CLI   XRECV,X'FF'
         BNE   ENDTEST             RECEIVE NOT ON SO BEGIN BROWSE
*
***********************************************************************
*        RECEIVE SET, THIS SHOULD NOT OCCUR SO RETURN IMMEDIATELY     *
***********************************************************************
*
         B     IMMRET
*
ENDTEST  DS    0H
*
***********************************************************************
*        START BROWSE FROM SUPPLIED RECORD KEY, AND READ              *
*        THROUGH FILE, ( FORWARDS OR BACKWARDS AS REQUIRED )          *
*        BUFFERING FOUR RECORDS AND SENDING THEM                      *
*        BACK TO THE DRIVER TRAN. ON EACH TRAN. INVOCATION.           *
***********************************************************************
*
*
*
         EXEC CICS STARTBR FILE(DSETIN) RIDFLD(KEYIN)                 *
               RESP(RESP)
         CLC  RESP,DFHRESP(DSIDERR)    DOES FILE EXIST?
         BE   DSIDERR1                 ... NO, GO TO DSIDERR1
         CLC  RESP,DFHRESP(DISABLED)   IS FILE DISABLED?
         BE   DISABLED                 ... YES, GO TO DISABLED
         CLC  RESP,DFHRESP(NOTFND)     IS RECORD PRESENT?
         BE   NOTFND1                  ... NO, GO TO NOTFND1
         CLC  RESP,DFHRESP(NORMAL)     CHECK FOR NORMAL RESPONSE
         BNE  ERROR1
*
         TM    RUNDNFLG,FORWARD    BROWSE FORWARD REQUESTED ?
         BNO   BROWBK              NO, BROWSE BACK REQUESTD.
*
***********************************************************************
*        BROWSING FORWARD ROUTINE                                     *
***********************************************************************
*
BROWFWD  DS    0H
*
         EXEC CICS READNEXT INTO(FILEREC) FILE(DSETIN)                *
               RIDFLD(KEYIN) RESP(RESP)
         BAL  R2,BRWTEST           BRANCH TO BRWTEST TO TEST RESPONSES
*
```

Figure 123 (Part 3 of 6). Sample 2: remote file browse – back-end transaction (DFH$IFBR)

```
         BAL   R14,COMMON          EXECUTE COMMON PROCESSING ROUTINE.
         CH    R15,=H'4'           IS THE BUFFER FULL ?
         BE    RETURN              YES, SO RETURN RECORDS.
         B     BROWFWD             NO, SO READ ANOTHER RECORD
*
***********************************************************************
*        BROWSING BACKWARDS ROUTINE                                   *
***********************************************************************
*
BROWBK   DS    0H
*
         EXEC CICS READPREV INTO(FILEREC) FILE(DSETIN)                 *
               RIDFLD(KEYIN) RESP(RESP)
         BAL  R2,BRWTEST           BRANCH TO BRWTEST TO TEST RESPONSES
*
         BAL   R14,COMMON          EXECUTE COMMON PROCESSING ROUTINE.
         CH    R15,=H'4'           IS THE BUFFER FULL ?
         BE    RETURN              YES, SO RETURN RECORDS.
         B     BROWBK              NO, SO READ ANOTHER RECORD
*
***********************************************************************
*        COMMON RECORD PROCESSING ROUTINE                             *
*        NOTE.  THE SAVE INDICATOR FLAG IS USED TO PREVENT US FROM    *
*        SAVING THE FIRST RECORD READ FROM THE FILE. THIS IS          *
*        BECAUSE WE WILL RESTART THE FILE BROWSE AT THE SAME POINT    *
*        AS WE FINISHED IN THE LAST RUN ( ASSUMING THIS IS NOT        *
*        THE FIRST RUN OTHERWISE WE MISS OUT THE FIRST RECORD ).      *
***********************************************************************
*
COMMON   DS    0H
         SR    R15,R15             ZERO RETURN REG
         TM    RUNDNFLG,SAVEIT     DO WE WANT TO SAVE THIS RECORD?
         BO    CONTINUE            YES, SO CONTINUE
         OI    RUNDNFLG,SAVEIT     NO, SET UP TO SAVE NEXT RECORD
         B     COMMONR              AND RETURN
CONTINUE DS    0H
         LA    R4,1(R4)            INCREMENT RECORD COUNT.
         LA    R5,L'FILEREC(R5)    INCREMENT BUFFER POINTER.
         CH    R4,=H'5'            OUTPUT AREA FILLED ?
         BL    COMMONR             NO, RETURN
         LA    R15,4   '           YES, SET R/C FOR FULL BUFFER
*
COMMONR  DS    0H
         BR    R14                 RETURN
*
***********************************************************************
*        ERROR RECEIVED DURING PROCESSING SO PREPARE RELEVANT MESSAGE *
***********************************************************************
*
ERROR1   DS    0H                  ERROR HAS OCCURRED,
         LA    R4,ERRORMSG         SO RETURN WITH
         B     SENDERR             ERROR MESSAGE.
*
ENDFILE1 DS    0H                  END OF FILE REACHED,
         LA    R4,ENDFLMSG         SO SEND BACK
         B     SENDERR             'END OF FILE' MESSAGE.
*
```

Figure 123 (Part 4 of 6). Sample 2: remote file browse — back-end transaction (DFH$IFBR)

```
NOTFND1  DS    0H                    RECORD WAS NOT FOUND,
         LA    R4,NOTFDMSG           SO SEND ERROR MESSAGE.
         MVC   KEY1,KEYIN            INSERT KEY INTO MESSAGE.
         B     SENDERR
*
DISABLED DS    0H                    FILE WAS DISABLED,
         LA    R4,DISABMSG           SO SEND ERROR MESSAGE.
         MVC   DSET1,DSETIN          INSERT FILE INTO MESSAGE.
         B     SENDERR
*
SYSERR1  DS    0H                    FILE WAS NOT ACCESSIBLE,
         LA    R4,SYSERMSG           SO SEND ERROR MESSAGE.
         MVC   DSET2,DSETIN          INSERT FILE INTO MESSAGE.
         B     SENDERR
*
DSIDERR1 DS    0H                    FILE WAS NOT FOUND,
         LA    R4,DSTERMSG           SO SEND ERROR MESSAGE.
         MVC   DSET3,DSETIN          INSERT FILE INTO MESSAGE.
         B     SENDERR
*
SENDERR  DS    0H
         LH    R6,0(R4)                  PUT MESSAGE LENGTH IN R6.
         BCTR  R6,R0
         EX    R6,MVCMSG                 PUT ERROR MSG IN OUTAREA.
         LA    R5,1(R5,R6)               POINT TO END OF DATA
         B     RETURN                    RETURN DATA AND TERMINATE.
MVCMSG   MVC   0(0,R5),L'OUTLEN(R4)
*
***********************************************************************
*        ROUTINE FOR TESTING RESPONSE TO CICS COMMANDS WHILE BROWSING *
*        FILE                                                         *
***********************************************************************
BRWTEST  DS    0H
         CLC   RESP,DFHRESP(DSIDERR)     DOES FILE EXIST?
         BE    DSIDERR1                  ... NO, GO TO DSIDERR1
         CLC   RESP,DFHRESP(DISABLED)    IS FILE DISABLED?
         BE    DISABLED                  ... YES, GO TO DISABLED
         CLC   RESP,DFHRESP(ENDFILE)     HAS END OF FILE BEEN REACHED?
         BE    ENDFILE1                  ... YES, GO TO ENDFILE1
         CLC   RESP,DFHRESP(NOTFND)      CAN THE RECORD BE FOUND?
         BE    NOTFND1                   ... NO, GO TO NOTFND1
         CLC   RESP,DFHRESP(LENGERR)     IS RECORD TOO LONG?
         BER   R2                        ... YES, IGNORE AND RETURN
         CLC   RESP,DFHRESP(NORMAL)      CHECK FOR NORMAL RESPONSE
         BER   R2                        IF OK THEN RETURN
         B     ERROR1                    ELSE GO TO ERROR ROUTINE
*
RETURN   DS    0H
*
***********************************************************************
*        RETURN CONTROL ( AND DATA ) BACK TO MASTER TRANSACTION       *
***********************************************************************
*
         MVC   OUTAREA(L'KEYIN),KEYIN     PUT LAST KEY READ AT
*                                         FRONT OF OUTAREA.
         LA    R4,OUTAREA                 CALCULATE LENGTH
         SR    R5,R4                      OF DATA TO SEND.
         STH   R5,OUTLEN
*
```

Figure 123 (Part 5 of 6). Sample 2: remote file browse — back-end transaction (DFH$IFBR)

```
         EXEC CICS SEND FROM(OUTAREA) LENGTH(OUTLEN) WAIT LAST          *
               RESP(RESP)
         CLC  RESP,DFHRESP(NORMAL)
         BNE  ERROR1
IMMRET   DS    0H
         EXEC CICS RETURN
*
ENDFLMSG DC    AL2(L'MSG1)
MSG1     DC    C'END OF FILE'
NOTFDMSG DC    AL2(L'MSG2)
MSG2     DC    C'RECORD WITH KEY ...... NOT FOUND'
         ORG   MSG2+16
KEY1     DS    CL6
         ORG   ,
DISABMSG DC    AL2(L'MSG3)
MSG3     DC    C'FILE ........ IS DISABLED'
         ORG   MSG3+5
DSET1    DS    CL8
         ORG   ,
SYSERMSG DC    AL2(L'MSG4)
MSG4     DC    C'FILE ........ IS NOT ACCESSIBLE'
         ORG   MSG4+5
DSET2    DS    CL8
         ORG   ,
DSTERMSG DC    AL2(L'MSG5)
MSG5     DC    C'FILE ........ NOT FOUND'
         ORG   MSG5+5
DSET3    DS    CL8
         ORG   ,
ERRORMSG DC    AL2(L'MSG6)
MSG6     DC    C'ERROR HAS OCCURRED IN REMOTE TRANSACTION'
*
         END
```

Figure 123 (Part 6 of 6). Sample 2: remote file browse — back-end transaction (DFH$IFBR)

BMS mapset for sample 2 (DFH$IMB)

```
         TITLE 'DFH$IMB - INTERCOMMUNICATION SAMPLE - REMOTE FILE BROWS*
               E - MAPSET'
DFH$IGB  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(ALARM,FREEKB),          *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES,EXTATT=NO
SPMAPA   DFHMDI SIZE=(24,80)
         DFHMDF POS=(4,15),LENGTH=28,ATTRB=(PROT,BRT),                 *
               INITIAL='CICS-CICS REMOTE FILE BROWSE'
         DFHMDF POS=(5,15),LENGTH=25,ATTRB=(PROT,BRT),                 *
               INITIAL='   SAMPLE PROGRAM MAP   '
         DFHMDF POS=(6,15),LENGTH=29,ATTRB=(PROT),                     *
               INITIAL='*****************************'
         DFHMDF POS=(10,4),LENGTH=19,ATTRB=(PROT),                     *
               INITIAL='6 DIGIT STBR KEY ..'
STRTKEY  DFHMDF POS=(10,25),LENGTH=6,ATTRB=(UNPROT,BRT,IC,FSET),       *
               INITIAL='      '
         DFHMDF POS=(10,32),ATTRB=(PROT),LENGTH=1
         DFHMDF POS=(12,4),LENGTH=19,ATTRB=(PROT),                     *
               INITIAL='REMOTE FILE NAME ..'
RDSET    DFHMDF POS=(12,25),LENGTH=8,ATTRB=(UNPROT,BRT,FSET),          *
               INITIAL='FILEA   '
         DFHMDF POS=(12,34),ATTRB=(PROT),LENGTH=1
         DFHMDF POS=(14,4),LENGTH=19,ATTRB=(PROT),                     *
               INITIAL='REMOTE SYSTEM ID ..'
RSYS     DFHMDF POS=(14,25),LENGTH=4,ATTRB=(UNPROT,BRT,FSET)
         DFHMDF POS=(14,30),ATTRB=(PROT),LENGTH=1
ERRMSG   DFHMDF POS=(18,4),LENGTH=36,ATTRB=(PROT,BRT),INITIAL=' '
         DFHMDF POS=(20,4),LENGTH=32,ATTRB=(PROT),                     *
               INITIAL='TYPE IN VALUES, THEN PRESS ENTER'
         DFHMDF POS=(21,4),LENGTH=32,ATTRB=(PROT),                     *
               INITIAL='OR HIT "PF3" TO TERMINATE.'
SPMAPE   DFHMDI SIZE=(24,80)
OUTMSG   DFHMDF POS=(22,4),LENGTH=40,ATTRB=(PROT,BRT),                 *
               INITIAL=' '
         DFHMDF POS=(23,4),LENGTH=23,ATTRB=(PROT,BRT),                 *
               INITIAL='TRANSACTION TERMINATED.'
         DFHMSD TYPE=FINAL
         END
```

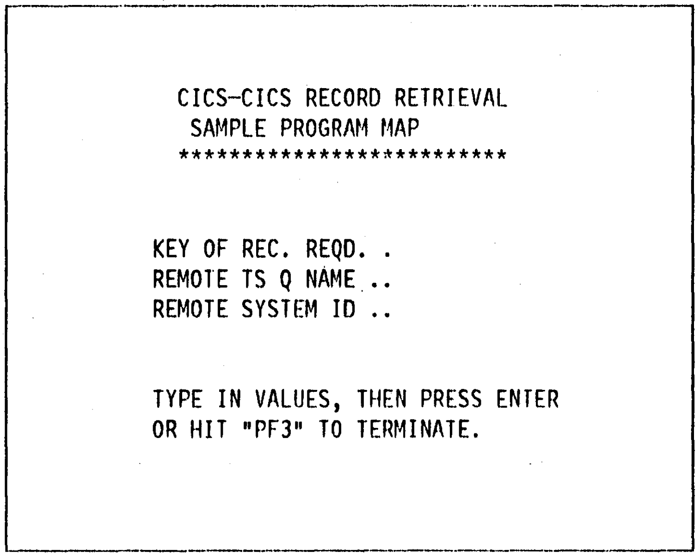
Figure 124. Sample 2: remote file browse — BMS mapset (DFH$IMB)

Sample 3 — remote record retrieval

This sample illustrates the use of asynchronous processing to retrieve a single record from a remote temporary storage queue. It consists of a local transaction (DFH$IQRL) to send the request to the remote system, a remote transaction (DFH$IQRR) to retrieve the record and return it to the local system, and a local transaction (DFH$IQRD) to receive the record and display it at the user terminal.

The remote temporary storage queue is assumed to consist of records that have unique user-defined keys in their first six bytes. If you want to run this sample, you will have to create a temporary storage queue of this form on the remote system.

The request transaction is invoked by the transaction code IQRL, and displays the following menu at the user's terminal:



```
          CICS-CICS RECORD RETRIEVAL
           SAMPLE PROGRAM MAP
          **************************


        KEY OF REC. REQD. .
        REMOTE TS Q NAME ..
        REMOTE SYSTEM ID ..


        TYPE IN VALUES, THEN PRESS ENTER
        OR HIT "PF3" TO TERMINATE.
```

KEY OF REC REQD
: specifies the user-defined key (that is, the first six bytes of data) of the remote temporary storage record.

REMOTE TS Q NAME
: specifies the name that the remote queue from which the record is to be retrieved.

REMOTE SYSTEM ID
: specifies the name of the remote system.

  This name can be the connection name of an LUTYPE6.2 or LUTYPE6.1 link.

The local request transaction uses a START command to start the remote retrieve transaction. The start request passes the name of the queue, the record number, the return transaction identifier (IQRD), and the return terminal identifier (obtained from the EIB). It also passes the APPLID of the local CICS system.

This enables the remote transaction to find the SYSID of the connection to system that issued the initial start request. Because both the local and the remote transactions name a SYSID explicitly on their START commands, neither of the systems requires a remote transaction definition.

The remote transaction retrieves the required record, and passes it back to the local system, again by means of a START command. This START command names the local display transaction IQRD.

The local display transaction then displays the record at the user's terminal.

Source listing of sample 3 local request transaction (DFH$IQRL)

```
DFH$IQRL TITLE 'CICS INTERCOMMUNICATION SAMPLE - TS RECORD RETRIEVAL - *
               LOCAL REQUEST'
***********************************************************************
*                                                                     *
*                      CICS SAMPLE PROGRAM IQRL                       *
*                      ASYNCHRONOUS  PROCESSING                       *
*                         'RECORD RETRIEVAL'                          *
*                         LOCAL TRANSACTION                           *
*                                                                     *
***********************************************************************
*                                                                     *
*     INPUTS TO THIS PROGRAM            KEY OF RECORD REQUIRED        *
*     ( FROM USERS SCREEN  )            REMOTE TS Q NAME TO SEARCH    *
*                                       REMOTE SYSTEM ID              *
*                                                                     *
*     OUTPUTS FROM THIS PROGRAM         KEY OF RECORD REQUIRED        *
*     ( PASSED TO TRANSACTION IQRR )    REMOTE TS Q NAME TO SEARCH    *
*                                       THIS SYSTEMS APPLID           *
*                                       USERS TERMINAL ID             *
*                                       RETURN TRANSACTION ID         *
*                                                                     *
***********************************************************************
*
DFH$IQRL CSECT
*
***********************************************************************
*        SEND INITIAL REQUEST MENU TO SCREEN                          *
***********************************************************************
RESETMAP DS    0H
         XC    SPMAPAO(SPMAPAL),SPMAPAO      CLEAR MAP STORAGE
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IG1') MAPONLY        *
               ERASE WAIT
*
***********************************************************************
*        RECEIVE AND VALIDATE THE REQUIREMENTS                        *
***********************************************************************
RETRY    DS    0H
         EXEC CICS RECEIVE MAP('SPMAPA') MAPSET('DFH$IG1')             *
               RESP(RESP)
*
         CLI   EIBAID,DFHPF3                  PF3 PRESSED ?
         BE    TERMNATE                       ...YES, GO TERMNATE TRANS.
         CLC   RESP,DFHRESP(MAPFAIL)          MAPFAIL ?
         BE    RESETMAP                       ...YES, GO RESEND MAP
         CLC   RESP,DFHRESP(NORMAL)           NORMAL RESPONSE ?
         BNE   BADRESP1                       ...NO, GO TERMINATE TRANS.
*
         CLC   KEYVALL,=H'0'                 RECORD KEY ENTERED ?
         BE    BADKEY                        ...NO, GO SEND ERROR MSG.
         CLC   KEYVALI,BLANKS                RECORD KEY BLANK ?
         BE    BADKEY                        ...YES, GO SEND ERROR MSG.
*
         CLC   RQNAML,=H'0'                  TS Q NAME ENTERED ?
         BE    BADQNM                        ...NO, GO SEND ERROR MSG.
         CLC   RQNAMI,BLANKS                 TS Q NAME BLANK ?
         BE    BADQNM                        ...YES, GO SEND ERROR MSG.
*
```

Figure 125 (Part 1 of 4). Sample 3: temporary storage record retrieval — local (request) transaction (DFH$IQRL)

```
         CLC   RSYSL,=H'0'              SYSID ENTERED ?
         BE    BADSYSID                 ...NO, GO SEND ERROR MSG.
         CLC   RSYSI,BLANKS             SYSID BLANK ?
         BE    BADSYSID                 ...YES, GO SEND ERROR MSG.
*
***********************************************************************
*        SET UP DATA TO BE PASSED BY START COMMAND.                   *
***********************************************************************
         MVC   KEYOUT,KEYVALI
         EXEC CICS ASSIGN APPLID(APPLID)
*
***********************************************************************
*        ISSUE START COMMAND FOR TRANSACTION 'IQRR' IN REMOTE         *
*        SYSTEM, PASSING REMOTE QNAME, RECORD KEY,                    *
*        THIS NETNAME AND RETURN TRANID AND TERMID.                   *
*        TO ENABLE THE REMOTE TRANSACTION TO FIND OUR SYSID, WE       *
*        PASS IT THE APPLID OF THIS SYSTEM IN THE 'START' COMMAND.    *
*        THE REMOTE TRANSACTION WILL USE THE 'EXTRACT TCT' COMMAND    *
*        TO ACQUIRE OUR SYSID. IT WILL THEN USE IT IN THE 'START'     *
*        COMMAND TO INITIATE ANOTHER TRANSACTION ON THIS SYSTEM.      *
***********************************************************************
*
         EXEC CICS START TRANSID('IQRR') SYSID(RSYSI) QUEUE(RQNAMI)    *
               FROM(DATAOUT) RTRANSID('IQRD') RTERMID(EIBTRMID)        *
               RESP(RESP)
*
         CLC   RESP,DFHRESP(SYSIDERR)     SYSIDERR ?
         BE    SYSERR                     ...YES, GO SEND ERROR MSG.
         CLC   RESP,DFHRESP(NORMAL)       NORMAL RESPONSE ?
         BNE   BADRESP2                   ...NO, GO TERMINATE TRANS.
         B     COMPLETE                   ...YES, GO TO COMPLETE TRAN.
*
***********************************************************************
*        TRANSACTION TERMINATION                                      *
***********************************************************************
TERMNATE DS    0H
         MVC   EMLEN,NORMLEN            SET UP MSG. LENGTH
EXIT     DS    0H
         MVC   EXITMSG1,NORMMSG         SET UP EXIT MESSAGE
         EXEC CICS SEND FROM(EXITMSG) LENGTH(EMLEN) ERASE
*
COMPLETE DS    0H
         EXEC CICS RETURN
         EJECT
***********************************************************************
*        ROUTINE TO HANDLE INVALID INPUT FROM USER                    *
***********************************************************************
BADKEY   DS    0H                  RECORD KEY NOT SUPPLIED
         MVC   ERRMSGO,KEYMSG           MOVE MESSAGE TO MAP
         MVC   KEYVALL,=H'-1'           SET CURSOR POSITION
         B     RESEND                   GO SEND ERROR MSG.
*
BADQNM   DS    0H                  REMOTE Q NAME NOT SUPPLIED
         MVC   ERRMSGO,QNMMSG           MOVE MESSAGE TO MAP
         MVC   RQNAML,=H'-1'            SET CURSOR POSITION
         B     RESEND                   GO SEND ERROR MSG.
*
```

Figure 125 (Part 2 of 4). Sample 3: temporary storage record retrieval — local (request) transaction (DFH$IQRL)

```
BADSYSID DS    0H                  REMOTE SYSTEM NAME NOT SUPPLIED
         MVC   ERRMSGO,SYSMSG           MOVE MESSAGE TO MAP
         MVC   RSYSL,=H'-1'             SET CURSOR POSITION
         B     RESEND                   GO SEND ERROR MSG.
*
*
***********************************************************************
*        ROUTINE TO HANDLE SYSIDERR RESPONSE FROM START COMMAND       *
***********************************************************************
*
SYSERR   DS    0H
         CLI   EIBRCODE+1,X'08'         LINK OUT OF SERVICE ?
         BNE   UNKNOWN                  ...NO, GO SET UP 'UNKNOWN' MSG.
         MVC   MSGWA,OUTSVMSG           MOVE SKELETON MSG. TO WORK AREA
         MVC   SYSWA1,RSYSI             INSERT SYSID INTO MESSAGE.
         B     MOVEM                    GO MOVE COMPLETE MSG.
*
UNKNOWN  DS    0H
         MVC   MSGWA,UNKWNMSG           MOVE SKELETON MSG. TO WORK AREA
         MVC   SYSWA2,RSYSI             INSERT SYSID INTO MESSAGE.
*
MOVEM    DS    0H
         MVC   ERRMSGO,MSGWA            MOVE ERROR MESSAGE TO MAP
         MVC   RSYSL,=H'-1'             SET CURSOR POSITION
*
***********************************************************************
*        DATA RECEIVED WAS INCOMPLETE, OR THERE WAS A PROBLEM         *
*        WITH THE SYSID, SO GIVE THE USER ANOTHER CHANCE TO           *
*        ENTER DATA.                                                  *
***********************************************************************
RESEND   DS    0H
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IG1') DATAONLY       *
               CURSOR WAIT
*
         B     RETRY               RECEIVE REQUIREMENTS AGAIN
*
*
***********************************************************************
*        HANDLE BAD RESPONSE FROM RECEIVE MAP COMMAND                 *
***********************************************************************
BADRESP1 DS    0H
         MVC   EXITMSG2,BADRCV          SET UP EXIT MSG.
         MVC   EMLEN,BRLEN              SET UP MSG. LEN
         B     EXIT                     GO TERMINATE TRANS.
*
***********************************************************************
*        HANDLE BAD RESPONSE FROM START COMMAND                       *
***********************************************************************
BADRESP2 DS    0H
         MVC   EXITMSG2(L'BADSTRT),BADSTRT   SET UP EXIT MSG.
         MVC   EMLEN,BSLEN              SET UP MSG. LEN
         B     EXIT                     GO TERMINATE TRANS.
         EJECT
```

Figure 125 (Part 3 of 4). Sample 3: temporary storage record retrieval — local (request) transaction (DFH$IQRL)

```
***********************************************************************
*        CONSTANTS                                                    *
***********************************************************************
BLANKS   DC    CL8' '
KEYMSG   DC    CL36'PLEASE SUPPLY KEY OF RECORD REQUIRED'
QNMMSG   DC    CL36'PLEASE SUPPLY A REMOTE QUEUE NAME.'
SYSMSG   DC    CL36'PLEASE SUPPLY A REMOTE SYSTEM ID.'
OUTSVMSG DC    CL36'LINK TO .... IS OUT OF SERVICE'
UNKWNMSG DC    CL36'SYSTEM NAME .... IS NOT KNOWN'
NORMMSG  DC    C'IQRL TRANSACTION TERMINATED'
NORMLEN  DC    AL2(L'NORMMSG)
BADRCV   DC    C' DUE TO BAD RESPONSE FROM RECEIVE MAP COMMAND'
BRLEN    DC    AL2(L'NORMMSG+L'BADRCV)
BADSTRT  DC    C' DUE TO BAD RESPONSE FROM START COMMAND'
BSLEN    DC    AL2(L'NORMMSG+L'BADSTRT)
*
         LTORG
         EJECT
***********************************************************************
*        EQUATES                                                      *
***********************************************************************
         COPY  DFHAID                DEFINITIONS FOR ATTENTION ID'S
*
         DFHREGS                     REGISTER EQUATES
         EJECT
***********************************************************************
*        WORKING STORAGE                                              *
***********************************************************************
DFHEISTG DSECT
*
         COPY  DFH$IG1               MAPSET DEFINITIONS
*
RESP     DS    F                     RESPONSES TO CICS COMMANDS
*
DATAOUT  DS    0CL14                 DATA TO BE PASSED TO IQRR
KEYOUT   DS    CL6                   RECORD KEY TO BE PASSED
APPLID   DS    CL8                   THIS SYSTEMS APPLID
*
EMLEN    DS    H                     EXIT MESSAGE LENGTH
EXITMSG  DS    0CL72                 EXIT MESSAGE AREA
EXITMSG1 DS    CL27                  COMMON MESSAGE
EXITMSG2 DS    CL45                  VARIABLE MESSAGE
*
MSGWA    DS    0CL36                 WORK AREA FOR ERROR MSGS.
         DS    CL8
SYSWA1   DS    CL4
SYSWA2   DS    CL4
         DS    CL20
*
         END
```

Figure 125 (Part 4 of 4). Sample 3: temporary storage record retrieval — local (request) transaction (DFH$IQRL)

Source listing of sample 3 remote retrieve transaction (DFH$IQRR)

```
DFH$IQRR TITLE 'CICS INTERCOMMUNICATION SAMPLE - TS RECORD RETRIEVAL - *
               REMOTE READ TS'
***********************************************************************
*                                                                     *
*                      CICS SAMPLE PROGRAM IQRR                       *
*                      ASYNCHRONOUS  PROCESSING                       *
*                         'RECORD RETRIEVAL'                          *
*                         REMOTE TRANSACTION                          *
*                                                                     *
***********************************************************************
*                                                                     *
*        INPUTS TO THIS PROGRAM        KEY OF RECORD REQUIRED         *
*        ( RECEIVED FROM IQRL )        TS Q NAME TO SEARCH            *
*                                      DRIVER TRANSACTIONS APPLID     *
*                                      USERS TERMINAL ID              *
*                                      RETURN TRANSACTION ID          *
*                                                                     *
*        OUTPUTS FROM THIS PROGRAM     REQUIRED RECORD ( + KEY )      *
*        ( PASSED TO PROGRAM IQRD )      OR ERROR MESSAGE             *
*                                      TS Q NAME REQUESTED            *
*                                                                     *
***********************************************************************
*
DFH$IQRR CSECT
*
***********************************************************************
*        RETRIEVE THE DATA PASSED BY START FROM IQRL                  *
***********************************************************************
         EXEC CICS RETRIEVE INTO(DATAIN) RTRANSID(TRANIN)              *
               RTERMID(TERMIN) QUEUE(QNAMEIN)
*
***********************************************************************
*        TO OBTAIN THE SYSID TO RETURN TO, WE WILL USE THE 'EXTRACT   *
*        TCT' COMMAND. WE SPECIFY THE NETNAME AS BEING THE APPLID     *
*        THAT WAS PASSED TO US. THIS WILL YIELD THE SYSID THAT WE     *
*        ASSOCIATE WITH THE RECEIVED APPLID.                          *
***********************************************************************
*
         EXEC CICS EXTRACT TCT NETNAME(APPLIN) SYSID(SYSBK)
*
***********************************************************************
*        FIND THE REQUIRED RECORD BY READING THROUGH THE QUEUE,       *
*        AND COMPARING THE RECORD'S KEY WITH THE KEY SUPPLIED.        *
***********************************************************************
         XC    QRECNUM,QRECNUM         INITIALIZE TS Q ITEM NO.
*
READNEXT DS    0H
         LH    R5,QRECNUM              INCREMENT TS Q ITEM NO.
         LA    R5,1(R5)
         STH   R5,QRECNUM
*
         EXEC CICS READQ TS QUEUE(QNAMEIN) SET(R6) LENGTH(RECLEN)      *
               ITEM(QRECNUM) RESP(RESP)
*
         CLC   RESP,DFHRESP(NORMAL)    GOT A RECORD ?
         BNE   CHKITEM                 NO, GO CHECK REASON ?
*
```

Figure 126 (Part 1 of 3). Sample 3: remote file browse — remote (retrieve) transaction (DFH$IQRR)

```
         CLC   KEYIN,0(R6)             IS THIS THE RIGHT RECORD ?
         BNE   READNEXT                NO, SO GO READ THE NEXT RECORD.
         B     STARTBK                 YES, SO GO START TRANS.
*
**********************************************************************
*        HANDLE NON NORMAL RESPONSE FROM READQ.                      *
**********************************************************************
CHKITEM  DS    0H
         CLC   RESP,DFHRESP(ITEMERR)   GOOD Q ID, BUT REC. NOT FOUND ?
         BNE   CHKQID                  NO, GO CHECK FOR BAD Q ID
         MVC   EMESSAGE(L'NOTFDMSG),NOTFDMSG   SET UP ERROR MESSAGE
         MVC   EMNFKEY,KEYIN                   INSERT KEY INTO MSG.
         MVC   RECLEN,NFMLEN                   SET UP MSG. LENGTH
         B     SENDERR
*
CHKQID   DS    0H
         CLC   RESP,DFHRESP(QIDERR)           UNKNOWN QUEUE NAME ?
         BNE   READQERR                       NO, GO SET BAD READQ MSG.
         MVC   EMESSAGE(L'QIDERMSG),QIDERMSG   SET UP ERROR MESSAGE
         MVC   EMBADQ,QNAMEIN                  INSERT Q NAME IN MSG.
         MVC   RECLEN,QEMLEN                   SET UP MSG. LENGTH
         B     SENDERR
*
READQERR DS    0H
         MVC   EMESSAGE(L'RDQERMSG),RDQERMSG   SET UP ERROR MESSAGE
         MVC   RECLEN,RDQMLEN                  SET UP MSG. LENGTH
*
SENDERR  DS    0H
         MVC   EMKEY,KEYIN             MOVE KEY
         LA    R6,ERRORMSG             ADDRESS DATA TO BE PASSED
*
**********************************************************************
*        ISSUE START BACK TO USERS SYSTEM PASSING THE                *
*        REQUIRED RECORD OR EXPLANATORY ERROR MESSAGE.               *
**********************************************************************
STARTBK  DS    0H
         EXEC CICS START TRANSID(TRANIN) TERMID(TERMIN) SYSID(SYSBK)   *
               FROM(0(R6)) LENGTH(RECLEN) QUEUE(QNAMEIN)
*
**********************************************************************
*        TERMINATE TRANSACTION.                                      *
**********************************************************************
         EXEC CICS RETURN
         EJECT
**********************************************************************
*        CONSTANTS                                                   *
**********************************************************************
NOTFDMSG DC    C'RECORD WITH KEY ...... NOT FOUND'
QIDERMSG DC    C'QUEUE NAME ........ IS NOT KNOWN'
RDQERMSG DC    C'BAD RESPONSE TO READQ TS IN REMOTE IQRR TRANSACTION'
NFMLEN   DC    AL2(L'EMKEY+L'NOTFDMSG)
QEMLEN   DC    AL2(L'EMKEY+L'QIDERMSG)
RDQMLEN  DC    AL2(L'EMKEY+L'RDQERMSG)
*
         LTORG
         EJECT
```

Figure 126 (Part 2 of 3). Sample 3: remote file browse – remote (retrieve) transaction (DFH$IQRR)

```
***********************************************************************
*        EQUATES                                                      *
***********************************************************************
         DFHREGS                      REGISTER EQUATES
         EJECT
***********************************************************************
*        WORKING STORAGE                                              *
***********************************************************************
DFHEISTG DSECT
RESP     DS    F                      RESPONSES TO CICS COMMANDS
*
DATAIN   DS    0CL14                  DATA RECEIVED FROM IQRL
KEYIN    DS    CL6                    RECORD KEY RECEIVED
APPLIN   DS    CL8                    APPLID PASSED TO FIND
*
TRANIN   DS    CL4                    TRANID TO START BACK TO
TERMIN   DS    CL4                    TERMID TO PASS BACK
QNAMEIN  DS    CL8                    QUEUE NAME TO BE SEARCHED
SYSBK    DS    CL4                    SYSID TO RETURN TO
*
QRECNUM  DS    H                      NUMBER OF RECORD WITHIN QUEUE
RECLEN   DS    H                      LENGTH OF DATA TO SEND
*
ERRORMSG DS    0CL80                  ERROR MESSAGE
EMKEY    DS    CL6                    RECORD KEY
EMESSAGE DS    CL74                   MESSAGE TEXT
         ORG   EMESSAGE+16
EMNFKEY  DS    CL6                    KEY OF NOT FOUND RECORD
         ORG   EMESSAGE+11
EMBADQ   DS    CL8                    NAME OF BAD Q ID
         ORG
*
         END
```

Figure 126 (Part 3 of 3). Sample 3: remote file browse — remote (retrieve) transaction (DFH$IQRR)

Source listing of sample 3 local display transaction (DFH$IQRD)

```
DFH$IQRD TITLE 'CICS INTERCOMMUNICATION SAMPLE - TS RECORD RETRIEVAL - *
               LOCAL DISPLAY'
***********************************************************************
*                                                                     *
*                     CICS SAMPLE PROGRAM IQRD                        *
*                     ASYNCHRONOUS  PROCESSING                        *
*                        'RECORD RETRIEVAL'                           *
*                        DISPLAY TRANSACTION                          *
*                                                                     *
***********************************************************************
*        INPUTS TO THIS PROGRAM         REQUIRED RECORD ( + KEY )     *
*        ( RECEIVED FROM IQRR )           OR ERROR MESSAGE            *
*                                       QUEUE NAME REQUESTED          *
*                                                                     *
*        OUTPUTS FROM THIS PROGRAM      KEY OF REQUIRED RECORD        *
*        (DISPLAYED VIA MAP DFH$IG2)    QUEUE NAME REQUESTED          *
*                                       REQUIRED RECORD OR ERROR MSG  *
*                                                                     *
***********************************************************************
*
DFH$IQRD CSECT
*
         XC    SPMAPAO(SPMAPAL),SPMAPAO     CLEAR MAP STORAGE
*
***********************************************************************
*        RETRIEVE THE DATA PASSED BY START FROM 'IQRR'                *
***********************************************************************
*
         EXEC CICS RETRIEVE SET(R6) LENGTH(DATALEN) QUEUE(RQNAMO)
*
***********************************************************************
*        MOVE DATA TO MAP                                             *
***********************************************************************
*
         MVC   RECKEYO,0(R6)              MOVE KEY OF RECORD.
         CLC   DATALEN,=AL2(L'RECKEYO)    ANY OTHER DATA ?
         BNH   SENDMAP                    NO, BYPASS MOVE
         LH    R5,DATALEN                 CALCULATE LENGTH OF...
         SH    R5,=AL2(L'RECKEYO+1)       ...DATA FOR MVC
         EX    R5,MVCRECO                 MOVE DATA
         B     SENDMAP
MVCRECO  MVC   RECORDO(0),L'RECKEYO(R6)      MOVE DATA TO MAP
*
***********************************************************************
*        SEND MAP TO SCREEN                                           *
***********************************************************************
SENDMAP  DS    0H
         EXEC CICS SEND MAP('SPMAPA') MAPSET('DFH$IG2') ERASE
*
***********************************************************************
*        TERMINATE TRANSACTION                                        *
***********************************************************************
         EXEC CICS RETURN
*
*
         LTORG
*
```

Figure 127 (Part 1 of 2). Sample 3: remote record retrieval — local (display) transaction (DFH$IQRD)

```
         DFHREGS                  REGISTER EQUATES
*
***********************************************************************
*        WORKING STORAGE                                              *
***********************************************************************
DFHEISTG DSECT
         COPY  DFH$IG2            MAPSET DEFINITIONS
*
RESP     DS    F                  RESPONSES TO CICS COMMANDS
DATALEN  DS    H                  LENGTH OF DATA TO SEND TO SCREEN.
*
         END
```

Figure 127 (Part 2 of 2). Sample 3: remote record retrieval — local (display) transaction (DFH$IQRD)

BMS mapset 1 for sample 3 (DFH$IM1)

```
DFH$IM1  TITLE 'INTERCOMMUNICATION SAMPLE - TS RECORD RETRIEVAL - MAPSE*
               T 1'
         AIF   ('&SYSPARM' EQ 'DSECT').SKIPSD - OS SMP REQUIRES CSECT
DFH$IG1C CSECT
.SKIPSD  ANOP ,
DFH$IG1  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=(ALARM,FREEKB),          *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES,EXTATT=NO
*
SPMAPA   DFHMDI SIZE=(24,80)
         DFHMDF POS=(4,16),LENGTH=26,ATTRB=(ASKIP,BRT),                *
               INITIAL='CICS-CICS RECORD RETRIEVAL'
         DFHMDF POS=(5,18),LENGTH=25,ATTRB=(ASKIP,BRT),                *
               INITIAL=' SAMPLE PROGRAM MAP   '
         DFHMDF POS=(6,16),LENGTH=29,ATTRB=(ASKIP),                    *
               INITIAL='*****************************'
         DFHMDF POS=(10,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='KEY OF REC. REQD. .'
KEYVAL   DFHMDF POS=(10,25),LENGTH=6,ATTRB=(UNPROT,BRT,IC,FSET)
         DFHMDF POS=(10,32),ATTRB=(ASKIP),LENGTH=1
         DFHMDF POS=(12,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='REMOTE TS Q NAME ..'
RQNAM    DFHMDF POS=(12,25),LENGTH=8,ATTRB=(UNPROT,BRT,FSET)
         DFHMDF POS=(12,34),ATTRB=(ASKIP),LENGTH=1
         DFHMDF POS=(14,4),LENGTH=19,ATTRB=(ASKIP),                    *
               INITIAL='REMOTE SYSTEM ID ..'
RSYS     DFHMDF POS=(14,25),LENGTH=4,ATTRB=(UNPROT,BRT,FSET)
         DFHMDF POS=(14,30),ATTRB=(PROT),LENGTH=1
ERRMSG   DFHMDF POS=(18,4),LENGTH=36,ATTRB=(ASKIP,BRT),                *
               INITIAL=' '
         DFHMDF POS=(20,4),LENGTH=32,ATTRB=(ASKIP),                    *
               INITIAL='TYPE IN VALUES, THEN PRESS ENTER'
         DFHMDF POS=(21,4),LENGTH=32,ATTRB=(ASKIP),                    *
               INITIAL='OR HIT "PF3" TO TERMINATE.'
*
         DFHMSD TYPE=FINAL
*
         END
```

Figure 128. Sample 3: remote record retrieval — BMS mapset 1 (DFH$IM1)

BMS mapset 2 for sample 3 (DFH$IM2)

```
DFH$IM2  TITLE 'INTERCOMMUNICATION SAMPLE - TS RECORD RETRIEVAL - MAPSE*
               T 2'
         AIF   ('&SYSPARM' EQ 'DSECT').SKIPSD - OS SMP REQUIRES CSECT
DFH$IG2C CSECT
.SKIPSD  ANOP ,
DFH$IG2  DFHMSD TYPE=&SYSPARM,MODE=OUT,CTRL=(ALARM,FREEKB),            *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES,EXTATT=NO
*
SPMAPA   DFHMDI SIZE=(24,80)
         DFHMDF POS=(1,1),LENGTH=28,ATTRB=(PROT,BRT),                  *
               INITIAL='REQUEST FOR RECORD PROCESSED'
         DFHMDF POS=(3,1),LENGTH=8,ATTRB=(PROT,BRT),                   *
               INITIAL='REC. KEY'
RECKEY   DFHMDF POS=(3,10),LENGTH=6,ATTRB=(PROT,NORM)
         DFHMDF POS=(3,20),LENGTH=4,ATTRB=(PROT,BRT),                  *
               INITIAL='TS Q'
RQNAM    DFHMDF POS=(3,25),LENGTH=8,ATTRB=(PROT,NORM)
RECORD   DFHMDF POS=(5,1),LENGTH=80,ATTRB=(PROT,NORM)
         DFHMDF POS=(7,1),LENGTH=20,ATTRB=(PROT,BRT),                  *
               INITIAL='PROCESSING COMPLETE.'
*
         DFHMSD TYPE=FINAL
*
         END
```

Figure 129. Sample 3: remote record retrieval — BMS mapset 2 (DFH$IM2)

Sample 4 — CICS to CICS or IMS conversation

The CICS to CICS synchronous sample application program allows a terminal operator to enter a command on the screen and have that command transmitted to a remote system for execution. If necessary, the remote system responds with a request for further details, and the operator is given the opportunity of replying.

The front-end transaction is invoked by the transaction code ICIC, and displays the following menu at the user's terminal:

```
TYPE REMOTE SYSTEM ID AND COMMAND

REMOTE SYSTEM ID
COMMAND

THEN PRESS ENTER TO CONTINUE, OR
CLEAR TO TERMINATE
```

The program is able to converse with any application on a remote system which sends output data either one line at a time or in multiple line format. The CICS supplied programs listed below have this capability, so the main purpose of this example is to provide the CICS system programmer with a simple test transaction that will show how to establish contact with a second, remote CICS system without the need for any application program coding. A successful test of this sample will indicate, to the extent of the features actually being tested, that the system network has been correctly set up and that the Inter-System components of CICS to allow distributed transaction processing are in order; failure will indicate errors in set up rather than in user programming.

At the start of the program, the operator is prompted to enter the name of the remote system to be attached, and the actual command to be executed on the remote system which is entered just as if it were a local command, for example, CSMT TAS. The program is able to handle both single line output from the remote system and also output which exceeds the terminal page size.

The message received from the remote system is assumed to be in SCS form, that is, containing printable characters and new line symbols only. This is the default output format for LU6 type terminals as produced by CICS supplied programs such as CSFE, CSMT, CSOT, CSST, or CSTT.

Source listing of sample 4 combined front-end and back-end transaction (DFH$ICIC)

```
* $SEG(DFH$ICIC),COMP(SAMPLES),PROD(CICS/VS):
         TITLE 'DFH$ICIC - INTERCOMMUNICATION SAMPLE - CICS TO CICS OR *
               IMS CONVERSATION'
DFHEISTG DSECT
*
*        STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
*
XSYNC    DS    C                       IF SET, SYNCPOINT MUST
*                                      BE EXECUTED
XFREE    DS    C                       IF SET, TERMINAL / LU
*                                      MUST BE FREED
XRECV    DS    C                       IF SET, RECEIVE MUST
*                                      BE EXECUTED
XSEND    DS    C                       RESERVED
*
XATT     DS    C                       IF SET, ATTACH HEADER
*                                      DATA EXISTS AND MAY BE
*                                      ACCESSED USING EXTRACT
XEOC     DS    C                       IF SET, END-OF-CHAIN
*                                      WAS RECEIVED WITH DATA
XFMH     DS    C                       IF SET, DATA PASSED TO
*                                      APPL'N CONTAINS FMH(S)
         COPY  DFH$IGC                   COPY MAP
REMDATA  DS    256D
ATCHSESS DS    CL4
CONTROL  DS    0CL60
SBA      DS    CL3
CDATA    DS    CL57
MESSAGE  DS    CL32
INLEN    DS    H
OUTLEN   DS    H
NEWLINE  EQU   X'15'
         EJECT
DFH$ICIC CSECT
  1)     EXEC CICS HANDLE CONDITION MAPFAIL(MAPFAIL)
         EXEC CICS HANDLE AID CLEAR(CLEAR)
MAPFAIL  XC    MAPAI(MAPAE-MAPAI),MAPAI    CLEAR MAP
  2)     EXEC CICS SEND MAP('MAPA') MAPSET('DFH$IGC')                  *
               ERASE MAPONLY WAIT
  3)     EXEC CICS RECEIVE MAP('MAPA') MAPSET('DFH$IGC')
         LA    8,DATAI
         MVC   DATABL(3+L'DATABO),DATAL
         MVC   OUTLEN,DATAL
  4)     EXEC CICS HANDLE CONDITION SYSIDERR(SYSERR)
*
  5)     EXEC CICS SEND MAP('MAPB') MAPSET('DFH$IGC')                  *
         WAIT ERASE.
  6)     EXEC CICS ALLOCATE SYSID(SYSIDI)
         MVC   ATCHSESS,EIBRSRCE
CONVERSE DS    0H
         MVC   INLEN,=H'2048'
  7)     EXEC CICS CONVERSE                                            *
               SESSION(ATCHSESS)                                       *
               FROM(0(8))                                              *
               FROMLENGTH(OUTLEN)                                      *
               INTO(REMDATA)                                           *
               TOLENGTH(INLEN)
         MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
```

Figure 130 (Part 1 of 3). Sample 4: CICS to CICS or IMS conversation — combined front-end and back-end transaction (DFH$ICIC)

```
DATASENT DS      0H
  8)     CLC    INLEN,=H'0'           IF NULL RU SENT
         BE     TESTSYNC              NOTHING TO SEND.
*
         LH     1,INLEN
         LA     2,REMDATA(1)          ADDR BYTE AFTER DATA
         MVI    0(2),X'13'            INSERT CURSOR HERE
         LA     1,1(,1)
         STH    1,INLEN
*
         EXEC CICS SEND TEXT FROM(REMDATA) LENGTH(INLEN)              *
                ACCUM.
TESTSYNC DS     0H
  9)     CLI    XSYNC,X'FF'
         BNE    TESTFREE
         EXEC CICS SYNCPOINT
TESTFREE DS     0H
 10)     CLI    XFREE,X'FF'
         BNE    TESTRECV
         EXEC CICS SEND PAGE RETAIN
         EXEC CICS RETURN
TESTRECV DS     0H
 11)     CLI    XRECV,X'FF'
         BNE    SEND
         MVC    INLEN,=H'2048'
         EXEC CICS RECEIVE SESSION(ATCHSESS) INTO(REMDATA)            *
                LENGTH(INLEN)
         MVC    XDFEIFLG,EIBSYNC      SAVE EIB VALUES
         B      DATASENT
SEND     DS     0H
 12)     EXEC CICS SEND PAGE RETAIN
         MVC    OUTLEN,=H'60'
         EXEC CICS RECEIVE INTO(CONTROL) LENGTH(OUTLEN)
         LH     0,OUTLEN
         SH     0,=H'3'               FOR LENGTH OF SBA
         LA     8,CDATA
         B      CONVERSE
*
SYSERR   DS     0H
 13)     CLI    EIBRCODE+1,12
         BE     UNKNOWN
         CLI    EIBRCODE+1,8
         BE     OUTSERV
         CLI    EIBRCODE+1,4
         BE     NOTCTSE
NOLINK   DS     0H
 14)     MVC    MESSAGE,LINKMSG
         MVC    MESSAGE+28(4),SYSIDI
         B      EXPLAIN
LINKMSG  DC     CL32'UNABLE TO ESTABLISH LINK TO      '
*
UNKNOWN  DS     0H
 15)     MVC    MESSAGE,UNKMSG
         MVC    MESSAGE+12(4),SYSIDI
         B      EXPLAIN
UNKMSG   DC     CL32'SYSTEM NAME      IS NOT KNOWN    '
*
OUTSERV  DS     0H
 16)     MVC    MESSAGE,OUTSVMSG
         MVC    MESSAGE+8(4),SYSIDI
         B      EXPLAIN
OUTSVMSG DC     CL32'LINK TO      IS OUT OF SERVICE'
*
```

Figure 130 (Part 2 of 3). Sample 4: CICS to CICS or IMS conversation — combined front-end and back-end transaction (DFH$ICIC)

```
NOTCTSE  DS    0H
 17)     MVC   MESSAGE,TCTMSG
         MVC   MESSAGE(4),SYSIDI
         B     EXPLAIN
TCTMSG   DC    CL32'     IS NOT A SYSTEM NAME'
*
EXPLAIN  DS    0H
         EXEC CICS SEND FROM(MESSAGE) LENGTH(=H'32')                 *
               ERASE WAIT.
CLEAR    DS    0H
         EXEC CICS SEND CONTROL FREEKB
         EXEC CICS RETURN
         END
```

Figure 130 (Part 3 of 3). Sample 4: CICS to CICS or IMS conversation – combined front-end and back-end transaction (DFH$ICIC)

Program notes for DFH$ICIC

1. Set up exit for map errors and clear key.
2. The screen is erased, and the prompting map displayed at the terminal.
3. The remote system name and command to be transmitted are mapped in.
4. Set up exit for the error conditions which may arise while establishing connection to the remote system.
5. The screen is erased again and the command entered by the operator is displayed on the top line.
6. A session is now allocated naming the remote system only, and its name is obtained from EIBRSRCE.
7. A CONVERSE command is now issued which sends the data entered by the terminal operator to the remote system which he has specified, then receives the resulting response from that system. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block will have to be examined, thus the values therein must be retained. The SESSION option is used since the application is requesting that an alternate facility be made available to it. Note that, although it is permissible to build an attach header and transmit it using the CONVERSE command, this action does not need to be taken in this case since by default CICS will assume that the first four characters of the transmitted data contain the transaction code.
8. If the data length field for the RECEIVE component of the CONVERSE indicates that there is data to be handled, a logical message is built using the BMS TEXT facility for subsequent sending to the screen. To ensure that the terminal cursor is placed on the next available line for any further input, the "Insert Cursor" control character is appended to the data stream.
9. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The "SYNCPOINT required" indicator in the EXEC Interface Block is first tested and if need be the program issues its own SYNCPOINT.

10. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the built logical message is sent to the screen using the RELEASE option of the SEND PAGE command which returns control direct to CICS and thus frees the session.
11. If the EXEC Interface Block indicates that the application is to continue receiving data from across the session, a further RECEIVE command is issued.
12. The indicators SYNCPOINT, FREE session, or RECEIVE, do not apply, thus by default the remote application has requested a further transmission from this program. (In the case of the CICS supplied programs named in the description above this would imply the receipt of a prompting message.) The program therefore sends the logical message built to date, which will include the prompt, to the terminal operator and receives the operator's reply; a second CONVERSE can then be issued across the session. Note that the "Set Buffer Address" control and the two buffer address bytes received from the terminal must be bypassed before transmission across the link.
13. The SYSID error routine has been entered. To determine the exact cause of the error, EIBRCODE must be examined, and an appropriate informatory message sent to the operator.
14. Some kind of error exists which prevents the link between the two systems from being established.
15. The remote system name given by the operator is not recognized.
16. The link to the remote system is out of service.
17. The system name given is recognized, but is not that for a remote system.

BMS mapset for sample 4 (DFH$IMC)

```
* $SEG(DFH$IMC),COMP(SAMPLES),PROD(CICS/VS):
         TITLE 'DFH$IMC - INTERCOMMUNICATION SAMPLE - CICS TO CICS OR I*
               MS CONVERSATION - MAPSET'
DFH$IGC  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=FREEKB,                  *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES
MAPA     DFHMDI SIZE=(12,80)
         DFHMDF POS=(1,1),ATTRB=ASKIP,LENGTH=33,                       *
               INITIAL='TYPE REMOTE SYSTEM ID AND COMMAND'
         DFHMDF POS=(3,1),ATTRB=ASKIP,LENGTH=16,                       *
               INITIAL='REMOTE SYSTEM ID'
SYSID    DFHMDF POS=(3,20),ATTRB=(NORM,IC),LENGTH=4,                   *
               INITIAL='    '
         DFHMDF POS=(3,25),LENGTH=1
         DFHMDF POS=(4,1),ATTRB=ASKIP,LENGTH=07,                       *
               INITIAL='COMMAND'
DATA     DFHMDF POS=(4,10),ATTRB=(NORM),LENGTH=68,                     *
               INITIAL=' '
         DFHMDF POS=(6,1),ATTRB=ASKIP,LENGTH=29,                       *
               INITIAL='THEN PRESS ENTER TO CONTINUE, OR'
         DFHMDF POS=(7,1),ATTRB=ASKIP,LENGTH=18,                       *
               INITIAL='CLEAR TO TERMINATE'
MAPB     DFHMDI SIZE=(12,80)
DATAB    DFHMDF POS=(1,1),ATTRB=(NORM),LENGTH=68,                      *
               INITIAL=' '
         DFHMSD TYPE=FINAL
         END
```

Figure 131. Sample 4: CICS to CICS or IMS conversation – BMS mapset (DFH$IMC)

Sample 5 — CICS to IMS conversation

This sample is activated with the transaction code IMSN.

The CICS to IMS sample is intended to illustrate a conversation of a simple nature. It is designed to operate with a program similar to the IMS ECHO application.

At the start of the program, the operator is prompted to enter the name of the remote system and application on that system, together with the input data to be ECHOed back.

The response at the terminal consists of the reECHOed data only.

The message received from the remote system is assumed to contain printable characters only, and to be in variable length variable block format. Each logical record is treated as representing one screen line and, for the purposes of this sample, may not be greater than 79 characters in length.

Source listing of sample 5 CICS transaction (DFH$IMSN)

```
* $SEG(DFH$IMSN),COMP(SAMPLES),PROD(CICS/VS):
         TITLE 'DFH$IMSN - INTERCOMMUNICATION SAMPLE - CICS TO IMS CONV*
               ERSATION'
DFHEISTG DSECT
*
*        STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
*
DFHSYNC  DS    C                    IF SET, SYNCPOINT MUST
*                                   BE EXECUTED
DFHFREE  DS    C                    IF SET, TERMINAL / LU
*                                   MUST BE FREED
DFHRECV  DS    C                    IF SET, RECEIVE MUST
*                                   BE EXECUTED
DFHSEND  DS    C                    RESERVED
*
DFHATT   DS    C                    IF SET, ATTACH HEADER
*                                   DATA EXISTS AND MAY BE
*                                   ACCESSED USING EXTRACT
DFHEOC   DS    C                    IF SET, END-OF-CHAIN
*                                   WAS RECEIVED WITH DATA
DFHFMH   DS    C                    IF SET, DATA PASSED TO
*                                   APPL'N CONTAINS FMH(S)
         COPY  DFH$IGS
         COPY  DFHBMSCA             BMS ATTRIBUTES
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
R8       EQU   8
R9       EQU   9
REMSYS   DS    CL8
ATCHSESS DS    CL4
INLEN    DS    H
         EJECT
DFH$IMSN CSECT
         XC    MAPAI(MAPAE-MAPAI),MAPAI  CLEAR MAP
SENDMAP  DS    0H
  1)     EXEC CICS SEND MAP('MAPA') MAPSET('DFH$IGS') ERASE WAIT
  2)     XC    DATAI,DATAI          RE-CLEAR THE DATA AREA
  3)     EXEC CICS RECEIVE MAP('MAPA') MAPSET('DFH$IGS')
  4)     CLI   SYSIDI,0             REMOTE SYSTEM NAME GIVEN ?
         BE    REMAP                ..NO, SEND MSG TO OPERATOR
  5)     EXEC CICS ALLOCATE SYSID(SYSIDI)
  6)     MVC   ATCHSESS,EIBRSRCE
         B     BUILD
```

Figure 132 (Part 1 of 3). Sample 5: CICS to IMS conversation — CICS transaction (DFH$IMSN)

```
REMAP    DS    0H
  7)     MVC   ERROIO(L'SYSMSG),SYSMSG SET UP PROMPTING MSG
         MVI   ERROIA,DFHBMBRY     HIGHLIGHT MESSAGE
         B     SENDMAP             AND SEND IT.
BUILD    DS    0H
  8)     EXEC CICS BUILD ATTACH ATTACHID('TIMS')                      *
               RESOURCE(TRANI) IUTYPE(=H'1') .
  9)     EXEC CICS SEND SESSION(ATCHSESS) ATTACHID('TIMS') FROM(DATAI) *
               LENGTH(DATAL) INVITE
RECV     DS    0H
 10)     EXEC CICS RECEIVE SESSION(ATCHSESS)                          *
               SET(R9) LENGTH(INLEN)
DATASENT DS    0H
         MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
*
         LA    R4,MAPBI                START OF OUTPUT MAP
         LR    R6,R4
         LA    R5,MAPBE-MAPBI          LENGTH OF MAP
         XR    R7,R7
         MVCL  R4,R6                   CLEAN UP THE MAP
*
 11)     CLC   INLEN,=H'0'             IF NULL RU SENT THEN
         BE    TESTSYNC                NOTHING TO SEND TO TRM
*
         LA    R7,LINE10               ADDRESS 1ST OUTPUT LINE
         LH    R4,INLEN                LENGTH OF RECEIVED DATA
LRECL    DS    0H
         LH    R5,0(R9)                LOGICAL RECORD LENGTH
         SR    R4,R5                   REDUCE BLOCK LENGTH
         SH    R5,=H'3'                PREPARE FOR EX INSTR.
         EX    R5,SETLINE              MOVE LREC TO MAP
         LTR   R4,R4                   END OF BLOCK REACHED ?
         BZ    SENDMAPB                ..YES, SEND THE MAP
         AH    R9,0(R9)                ADVANCE TO NEXT RECORD
         LA    R7,LINE20-LINE10(R7)    ADDR NEXT OUTPUT LINE
         B     LRECL                   GO TO MOVE NEXT REC
SENDMAPB DS    0H
         EXEC CICS SEND MAP('MAPB') MAPSET('DFH$IGS') ERASE WAIT      *
               CURSOR(=H'1840')
*
TESTSYNC DS    0H
 12)     CLI   DFHSYNC,X'FF'
         BNE   TESTFREE
         EXEC CICS SYNCPOINT
TESTFREE DS    0H
 13)     CLI   DFHFREE,X'FF'
         BE    EXIT
 14)     EXEC CICS RECEIVE SET(R8) LENGTH(INLEN)
 15)     CLI   DFHRECV,X'FF'
         BE    RECV
```

Figure 132 (Part 2 of 3). Sample 5: CICS to IMS conversation – CICS transaction (DFH$IMSN)

```
 16)     EXEC CICS CONVERSE FROMLENGTH(INLEN) SESSION(ATCHSESS)          *
                SET(R9) TOLENGTH(INLEN) FROM(0(R8))
         B      DATASENT
*
SETLINE  MVC    0(0,R7),2(R9)        MOVE INPUT RECORD TO MAP
SYSMSG   DC     C'MUST SPECIFY REMOTE SYSID'
EXIT     DS     0H
         END
```

Figure 132 (Part 3 of 3). Sample 5: CICS to IMS conversation – CICS transaction (DFH$IMSN)

Program notes for DFH$IMSN

1. The screen is erased, and the prompting map displayed at the terminal.
2. The data area portion of the map is used to hold any error messages sent to the terminal; this area is cleared before a RECEIVE is issued.
3. The remote system name and data are mapped in.
4. The terminal operator now enters the remote system name.
5. If the remote system name is given, an ALLOCATE is performed on that system, and
6. The name of the actual session allocated is found in the EIBRSRCE field.
7. Use the input data area of the map to advise the operator to try again.
8. A transaction is to be initiated on a remote system which needs to know the transaction name. This is detailed in the attach header which is built at this point. For IMS, the "transaction name" must be entered as the resource name; the processing name being reserved for an attached system process (when used). Also, since IMS requires single-chain input, the IUTYPE option is set to binary halfword '1'.
9. The data entered by the terminal operator is now sent across the acquired session together with the previously built attach header. The presence of the INVITE option indicates that a RECEIVE will directly follow this SEND and improves performance across the session.
10. A RECEIVE is issued against the remote system to read back the echoed data. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block will have to be retained.
11. If the data length field for the previous RECEIVE indicates that there is data to be handled, it is sent to the requesting terminal.
12. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The SYNCPOINT required indicator in the EXEC Interface Block is first tested and if necessary the program issues its own SYNCPOINT.
13. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the program now exits causing an automatic freeing of the session.

14. The program receives further input data from the terminal operator. This allows for the remote program to send, for example, a request for further input. For simple autopaging through an output file, pressing ENTER is all that is required.
15. If the EXEC Interface Block indicates that the application is to continue receiving data from the session, a further RECEIVE command is issued.
16. A CONVERSE command is now issued which sends the data entered by the terminal operator to the remote system which he has specified, then receives the resulting response from that system. To enable the program to determine what action is next expected of it, the contents of the EXEC Interface Block must again be retained.

BMS mapset for sample 5 (DFH$IMS)

Note: This mapset is also used for sample 6.

```
* $SEG(DFH$IMS),COMP(SAMPLES),PROD(CICS/VS):
         TITLE 'DFH$IMS - INTERCOMMUNICATION SAMPLE - CICS TO IMS CONVE*
               RSATION/DEMAND PAGED OUTPUT - MAPSET'
DFH$IGS  DFHMSD TYPE=&SYSPARM,MODE=INOUT,CTRL=FREEKB,                  *
               LANG=ASM,STORAGE=AUTO,TIOAPFX=YES
MAPA     DFHMDI SIZE=(24,80)
         DFHMDF POS=(1,01),ATTRB=ASKIP,LENGTH=26,                      *
               INITIAL='INVOKE RETURN APPLICATION'
         DFHMDF POS=(3,01),ATTRB=ASKIP,LENGTH=25,                      *
               INITIAL='SUPPLY VALUES AS REQUIRED'
         DFHMDF POS=(4,01),ATTRB=(BRT,ASKIP),LENGTH=30,                *
               INITIAL='REMOTE TRANSACTION NAME......'
TRAN     DFHMDF POS=(4,32),ATTRB=(NORM,IC),LENGTH=8,                   *
               INITIAL= ' '
         DFHMDF POS=(4,41),LENGTH=1
         DFHMDF POS=(5,01),ATTRB=(BRT,ASKIP),LENGTH=30,                *
               INITIAL='REMOTE SYSTEM ID............'
SYSID    DFHMDF POS=(5,32),ATTRB=NORM,LENGTH=4,INITIAL='  '
         DFHMDF POS=(5,37),LENGTH=1
         DFHMDF POS=(8,01),ATTRB=(BRT,ASKIP),LENGTH=34,                *
               INITIAL='AND RETURN TRANSACTION INPUT DATA'
         DFHMDF POS=(8,36),LENGTH=1
DATA     DFHMDF POS=(10,1),ATTRB=NORM,LENGTH=79,INITIAL=' '
ERROI    DFHMDF POS=(11,1),ATTRB=NORM,LENGTH=79
MAPB     DFHMDI SIZE=(24,80)
LINE1    DFHMDF POS=(1,1),ATTRB=NORM,LENGTH=79
LINE2    DFHMDF POS=(2,1),ATTRB=NORM,LENGTH=79
LINE3    DFHMDF POS=(3,1),ATTRB=NORM,LENGTH=79
LINE4    DFHMDF POS=(4,1),ATTRB=NORM,LENGTH=79
LINE5    DFHMDF POS=(5,1),ATTRB=NORM,LENGTH=79
LINE6    DFHMDF POS=(6,1),ATTRB=NORM,LENGTH=79
LINE7    DFHMDF POS=(7,1),ATTRB=NORM,LENGTH=79
LINE8    DFHMDF POS=(8,1),ATTRB=NORM,LENGTH=79
LINE9    DFHMDF POS=(9,1),ATTRB=NORM,LENGTH=79
LINE10   DFHMDF POS=(10,1),ATTRB=NORM,LENGTH=79
ERROR    DFHMDF POS=(23,1),ATTRB=(BRT,ASKIP),LENGTH=70
         DFHMDF POS=(23,74),LENGTH=1,ATTRB=NORM
         DFHMSD TYPE=FINAL
         END
```

Figure 133. Samples 5 and 6: CICS to IMS — BMS mapset (DFH$IMS)

Sample 6 — CICS to IMS (demand paged output)

The following sample fulfills the same requirements as sample 4 except that provision is made for the operator to read IMS Demand Paged Output using Operator Logical Paging.

The sample automatically reads the first logical page of the IMS output and it is then the responsibility of the terminal operator to signify which logical page of the output message he now requires to see.

The message received from the remote system is assumed to contain printable characters only, and to be in VLVB (variable length variable block) format. Each logical record is treated as representing one screen line, and for the purposes of this example, may not be greater than 79 characters in length.

This sample is activated with the transaction code IMSO.

The following functions, available to the operator, are supported by the sample; they should be preceded by 'P/' as if normal CICS BMS paging were being performed.

- N = display the next logical page.
- P = display the previous logical page.
- C = redisplay the current logical page.
- Enter =n, =nn or =nnn to display a specific logical page of the message.
- +n, +nn or +nnn to display the nth logical page past the current position.
- -n, -nn or -nnn to display the nth logical page before the current position.
- Press CLEAR to delete the current message from the IMS system and CLEAR the user's screen.

Source listing of sample 6 CICS transaction (DFH$IMSO)

```
         TITLE 'DFH$IMSO - INTERCOMMUNICATION SAMPLE - CICS TO IMS DEMA*
               ND PAGED OUTPUT'
DFHEISTG DSECT
         COPY  DFH$IGS
*
*        STORAGE AREA FOR EIB SESSION AND STATUS FLAGS
*
XDFEIFLG DS    0CL7
*
DFHSYNC  DS    C                   IF SET, SYNCPOINT MUST
*                                  BE EXECUTED
DFHFREE  DS    C                   IF SET, TERMINAL / LU
*                                  MUST BE FREED
DFHRECV  DS    C                   IF SET, RECEIVE MUST
*                                  BE EXECUTED
DFHSEND  DS    C                   RESERVED
*
DFHATT   DS    C                   IF SET, ATTACH HEADER
*                                  DATA EXISTS AND MAY BE
*                                  ACCESSED USING EXTRACT
DFHEOC   DS    C                   IF SET, END-OF-CHAIN
*                                  WAS RECEIVED WITH DATA
DFHFMH   DS    C                   IF SET, DATA PASSED TO
*                                  APPL'N CONTAINS FMH(S)
         COPY  DFHBMSCA            BMS ATTRIBUTES
         COPY  DFHAID              ATTENTION IDS
R2       EQU   2
* R3 IS BASE REGISTER
R4       EQU   4
R5       EQU   5
R6       EQU   6
R7       EQU   7
DPAGEREG EQU   8
R9       EQU   9
R10      EQU   10
*
MESSAGE  DS    CL32
RESP     DS    F
DBLWORD  DS    D
INLEN    DS    H
REMSYS   DS    CL8
ATCHSESS DS    CL4
*
OLP      DSECT
OLPCODE  DS    CL2
OLPVAL   DS    CL3
         EJECT
DFH$IMSO CSECT
MAPFAIL  XC    MAPAI(MAPAE-MAPAI),MAPAI  CLEAR MAP
```

Figure 134 (Part 1 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
SENDMAP  DS    0C
 1)      EXEC CICS SEND MAP('MAPA') MAPSET('DFH$IGS') ERASE WAIT       *
               RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)    CHECK FOR NORMAL RESPONSE
         BNE   ERROR1
 2)      EXEC CICS RECEIVE MAP('MAPA') MAPSET('DFH$IGS') RESP(RESP)
         CLI   EIBAID,DFHCLEAR         WAS CLEAR KEY PRESSED ?
         BE    EXIT                    YES, EXIT
         CLC   RESP,DFHRESP(NORMAL)    WAS MAP RECEIVED OK?
         BE    RESPOK1                 ... YES, GO TO RESPOK1
         CLC   RESP,DFHRESP(MAPFAIL)
         BE    MAPFAIL                 ... NO, GO TO MAPFAIL
         CLC   RESP,DFHRESP(EOC)       ... EOC OK
         BNE   ERROR1
RESPOK1  DS    0H
         CLI   SYSIDI,0            REMOTE SYSTEM NAME GIVEN ?
         BE    REMAP               ..NO, SEND MSG TO OPERATOR
 3)      EXEC CICS ALLOCATE SYSID(SYSIDI) RESP(RESP)
         CLC   RESP,DFHRESP(SYSIDERR)  IS SYSTEM ID VALID?
         BE    SYSERR                  ... NO, GO TO SYSERR
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
 4)      MVC   ATCHSESS,EIBRSRCE
         B     BUILD
REMAP    DS    0H
 5)      MVC   ERROIO(L'SYSMSG),SYSMSG SET UP PROMPTING MSG
         MVI   ERROIA,DFHBMBRY     HIGHLIGHT MESSAGE
 6)      XC    MAPAI(MAPAE-MAPAI),MAPAI  RE-CLEAR MAP
         B     SENDMAP             AND SEND IT.
BUILD    DS    0H
 7)      EXEC CICS BUILD ATTACH                                        *
               ATTACHID('TIMS') RESOURCE(TRANI) IUTYPE(=H'1')          *
               RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
 8)      EXEC CICS SEND SESSION(ATCHSESS) ATTACHID('TIMS') FROM(DATAI) *
               LENGTH(DATAL) INVITE RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
         EXEC CICS SYNCPOINT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
 9)      EXEC CICS RECEIVE SESSION(ATCHSESS)                           *
               SET(R9) LENGTH(INLEN) RESP(RESP)
         BAL   R2,TESTRESP             CHECK RESPONSES FROM COMMAND
 10)     MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
 11)     CLI   DFHATT,X'FF'            IF NO HEADER SENT,
         BNE   ABEND                   REMOTE SYSTEM ERROR.
*
```

Figure 134 (Part 2 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
 12)     EXEC CICS EXTRACT ATTACH SESSION(ATCHSESS)                    *
               QUEUE(QGETQNAM) RESP(RESP).  GET REMOTE QUEUE NAME.
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
 13)     MVC   QGETNQNM,QGETQNAM
         MVC   QPURGENM,QGETQNAM
*
 14)     EXEC CICS BUILD ATTACH                                        *
               ATTACHID('QMOD') PROCESS(QMODEL) IUTYPE(=H'1')          *
               RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
*
RECV     DS    0H
 15)     LA    DPAGEREG,1              1ST LOGICAL PAGE.
 16)     EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGETN)              *
               FROMLENGTH(QGETNLEN) TOLENGTH(INLEN) SET(R9)            *
               ATTACHID('QMOD') FMH RESP(RESP)
         BAL   R2,TESTRESP
*
UNPICK   DS    0H
         MVC   XDFEIFLG,EIBSYNC        SAVE EIB VALUES
 17)     LA    R4,MAPBI                START OF OUTPUT MAP
         LR    R6,R4
         LA    R5,ERRORL-MAPBI         LENGTH OF MAP
         XR    R7,R7
         MVCL  R4,R6                   CLEAN UP THE MAP
*
         LH    R4,INLEN                LENGTH OF REC'D DATA
*
         XR    R5,R5
CATFMH   DS    0H
         IC    R5,0(R9)                FMH LENGTH.
         LR    R6,R9
 18)     AR    R9,R5                   POINT BEYOND FMH.
         SR    R4,R5                   LENGTH OF ACTUAL DATA.
 19)     BZ    QSTATUS                 QSTATUS IF NO DATA.
         TM    1(R6),X'80'             ANY CONCATENATED FMHS ?
         BO    CATFMH                  YES - AGAIN.
*
         LA    R7,LINE10               ADDRESS 1ST OUTPUT LINE
LRECL    DS    0H
 20)     LH    R5,0(R9)                LOGICAL RECORD LENGTH
         SR    R4,R5                   REDUCE BLOCK LENGTH
         SH    R5,=H'3'                PREPARE FOR EX INSTR.
 21)     EX    R5,SETLINE              MOVE LREC TO MAP
         LTR   R4,R4                   END OF BLOCK REACHED ?
         BZ    SENDMAPB                ..YES, SEND THE MAP
         AH    R9,0(R9)                ADVANCE TO NEXT RECORD
         LA    R7,LINE20-LINE10(R7)    ADDR NEXT OUTPUT LINE
         B     LRECL                   GO TO MOVE NEXT REC
```

Figure 134 (Part 3 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
SENDMAPB DS    0H
 22)     EXEC CICS SEND MAP('MAPB') MAPSET('DFH$IGS') ERASE WAIT      *
               CURSOR(=H'1841') RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
         XC    ERRORO,ERRORO            CLEAR ERROR LINE.
*
TESTSYNC DS    0H
 23)     CLI   DFHSYNC,X'FF'
         BNE   TESTFREE
         EXEC CICS SYNCPOINT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
TESTFREE DS    0H
 24)     CLI   DFHFREE,X'FF'
         BE    EXIT
 25)     CLI   DFHRECV,X'FF'
         BE    ABEND
*
GETOLP   DS    0H
 26)     EXEC CICS RECEIVE SET(R9) LENGTH(INLEN) RESP(RESP)
         CLI   EIBAID,DFHCLEAR        WAS CLEAR KEY PRESSED ?
         BE    CLEAR                  YES, EXIT
         BAL   R2,TESTRESP
*
         LA    R9,3(R9)                BYPASS SBA BYTES.
         USING OLP,R9
 27)     CLC   OLPCODE,=C'P/'          'P/' REQUIRED TO START.
         BNE   OLPERR
 28)     CLI   OLPVAL,C'C'             CURRENT PAGE AGAIN ?
         BE    READQ                   YES.
         L     R5,=F'1'                SET PAGE NUMBER INCREMENT +1
 29)     CLI   OLPVAL,C'N'             NEXT PAGE REQUIRED ?
         BE    ADDINCR                 YES
 30)     CLI   OLPVAL,C'P'             PREVIOUS PAGE REQ'D ?
         BE    SUBINCR
*
 31)     LH    R4,INLEN
         SH    R4,=H'6'                SBA +  3 CHARS. R4 IS LEN OF   X
                                       NUMBER IF A SIGN IS PRESENT
         BM    OLPERR                  MUST BE SBA + 'P/' + MIN 1 CHAR
*
         LA    R5,OLPVAL+1             POINT R5 AFTER SIGN
         CLI   OLPVAL,C'+'             POSITIVE INCREMENT ?
         BE    PACKINST
         CLI   OLPVAL,C'-'             NEGATIVE INCREMENT ?
         BE    PACKINST
         LA    R5,OLPVAL               NO SIGN SO POINT R5 TO 1ST DIG
         LA    R4,1(,R4)               NO SIGN SO LEN IS 1 GREATER
*
```

Figure 134 (Part 4 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
PACKINST DS    0H
         LR    R10,R5                  R5 POINTS TO 1ST DIGIT
         LR    R6,R5
         AR    R6,R4                   R6 POINTS AFTER LAST DIGIT
CHECKNUM DS    0H
         CLI   0(R10),C'0'             IF DIGIT < '0' THEN ...
         BL    NONUM                   ... IT IS AN ERROR
         CLI   0(R10),C'9'             IF DIGIT > '9' THEN ...
         BH    NONUM                   ... IT IS AN ERROR
         LA    R10,1(,R10)             ADDRESS NEXT DIGIT
         CR    R10,R6                  HAVE WE REACHED THE MAX LENGTH?
         BL    CHECKNUM                NO, CHECK NEXT DIGIT
         S     R4,=F'1'                DECREMENT LENGTH FOR EXECUTE
         EX    R4,PACK                 PACK PAGE NO. AND
         CVB   R5,DBLWORD              CONVERT TO BINARY VALUE. R5 NOWX
                                       CONTAINS THE NUMBER ENTERED
*
         CLI   OLPVAL,C'+'             POSITIVE INCREMENT ?
         BE    ADDINCR
         CLI   OLPVAL,C'-'             NEGATIVE INCREMENT ?
         BE    SUBINCR
*
         LR    DPAGEREG,R5             NO INCR - ABSOLUTE PAGE
         B     READQ
*
SUBINCR  DS    0H
         SR    DPAGEREG,R5             SET TS ITEM NO.
         B     READQ
ADDINCR  DS    0H
         AR    DPAGEREG,R5             SET TS ITEM NO.
*
READQ    DS    0H
         LTR   DPAGEREG,DPAGEREG      IF PAGE NO. IS NOT POSITIVE
         BNP   OLPERR                 THIS IS AN ERROR.
 32)     STCM  DPAGEREG,3,DPAGENO     STORE QUEUE RECORD NO.
         EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGET)               *
               FROMLENGTH(QGETLEN) TOLENGTH(INLEN) SET(R9) FMH         *
               RESP(RESP)
         BAL   R2,TESTRESP            TEST RESPONSES FROM COMMAND
 33)     CLC   INLEN,=H'0'            IF NULL RU SENT,THEN
         BNE   UNPICK                 ANALYSE INPUT.
         MVC   XDFEIFLG,EIBSYNC       SAVE EIB VALUES
         B     TESTSYNC               NOTHING TO SEND.
*
PACK     PACK  DBLWORD,0(0,R5)
*
OLPERR   DS    0H
 34)     MVC   ERRORO(L'OLPERMSG),OLPERMSG  SET UP MSG.
         B     SENDMAPB
*
```

Figure 134 (Part 5 of 8). Sample 6: CICS-to-IMS demand paged output – CICS transaction (DFH$IMSO)

```
QSTATUS  DS    0H
 35)     MVC   ERRORO(L'QSTAMSG),QSTAMSG    SET UP MSG.
         LA    DPAGEREG,1              1ST LOGICAL PAGE.
         EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QGETN)             *
               FROMLENGTH(QGETNLEN) TOLENGTH(INLEN) SET(R9) FMH       *
               RESP(RESP)
         BAL   R2,TESTRESP             CHECK RESPONSES FROM COMMAND
*
         B     UNPICK
*
CLEAR    DS    0H
 36)     EXEC CICS CONVERSE SESSION(ATCHSESS) FROM(QPURGE)            *
               FROMLENGTH(QPURGELN) TOLENGTH(INLEN) SET(R9) FMH       *
               RESP(RESP)
         BAL   R2,TESTRESP             CHECK RESPONSES FROM COMMAND
         B     EXIT
*
ABEND    DS    0H
         MVC   ERRORO(L'ABENDMSG),ABENDMSG SET UP ERROR MSG.
         EXEC CICS SEND MAP('MAPB') MAPSET('DFH$IGS')                 *
               WAIT RESP(RESP)
         CLC   RESP,DFHRESP(NORMAL)
         BNE   ERROR1
         B     EXIT
*
SETLINE  MVC   0(0,R7),2(R9)           MOVE LOG.REC. TO MAP.
*
SYSMSG   DC    C'MUST SPECIFY REMOTE SYSID'
OLPERMSG DC    C'OPERATOR LOGICAL PAGING ERROR - RE-TYPE'
ABENDMSG DC    C'PROCESSING ERROR IN REMOTE SYSTEM'
QSTAMSG  DC    C'PAGE NO. EXCEEDS QUEUE SIZE'
*
*        QGETN
*
QGETN    DS    0H
         DC    X'10060A1000010208'
QGETQNAM DC    CL8' '
*
QGETNLEN DC    AL2(*-QGETN)        LENGTH.
*
*        QGET
*
QGET     DS    0H
         DC    X'13060A04000102'
         DC    X'08'
QGETNQNM DC    CL8' '
         DC    X'02'
DPAGENO  DS    CL2
*
QGETLEN  DC    AL2(*-QGET)         LENGTH.
*
*
```

Figure 134 (Part 6 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
*        QPURGE
*
QPURGE   DS    0H
         DC    X'10060A06000102'
         DC    X'08'
QPURGENM DC    CL8' '
*
QPURGELN DC    AL2(*-QPURGE)       LENGTH.
*
QMODEL   DS    0CL8
         DC    X'03'
         DC    CL7' '
*
TESTRESP DS    0H
         CLC   RESP,DFHRESP(INBFMH)  IF FMH PRESENT, THEN IGNORE
         BER   R2                      AND RETURN
         CLC   RESP,DFHRESP(EOC)     IF EOC PRESENT, THEN IGNORE
         BER   R2                      AND RETURN
         CLC   RESP,DFHRESP(NORMAL)    ELSE CHECK FOR NORMAL RESPONSE
         BER   R2
*
NONUM    DS    0H
         MVC   ERRORO(L'ERRNUM),ERRNUM     NON NUMERIC MESSAGE
         B     SENDMAPB
ERRNUM   DC    CL32'ERROR - VALUE IS NOT NUMERIC'
ERROR1   DS    0H
         MVC   MESSAGE,ERRMSG
         B     EXPLAIN
ERRMSG   DC    CL32'ERROR - TRANSACTION TERMINATED'
*
SYSERR   DS    0H
         CLI   EIBRCODE+1,12
         BE    UNKNOWN
         CLI   EIBRCODE+1,8
         BE    OUTSERV
         CLI   EIBRCODE+1,4
         BE    NOTCTSE
NOLINK   DS    0H
         MVC   MESSAGE,LINKMSG
         MVC   MESSAGE+28(4),SYSIDI
         B     EXPLAIN
LINKMSG  DC    CL32'UNABLE TO ESTABLISH LINK TO       '
*
UNKNOWN  DS    0H
         MVC   MESSAGE,UNKMSG
         MVC   MESSAGE+12(4),SYSIDI
         B     EXPLAIN
UNKMSG   DC    CL32'SYSTEM NAME      IS NOT KNOWN   '
*
```

Figure 134 (Part 7 of 8). Sample 6: CICS-to-IMS demand paged output — CICS transaction (DFH$IMSO)

```
OUTSERV  DS    0H
         MVC   MESSAGE,OUTSVMSG
         MVC   MESSAGE+8(4),SYSIDI
         B     EXPLAIN
OUTSVMSG DC    CL32'LINK TO      IS OUT OF SERVICE'
*
NOTCISE  DS    0H
         MVC   MESSAGE,TCTMSG
         MVC   MESSAGE(4),SYSIDI
         B     EXPLAIN
TCTMSG   DC    CL32'     IS NOT A SYSTEM NAME'
*
EXPLAIN  DS    0H
         EXEC CICS SEND FROM(MESSAGE) LENGTH(=H'32')                *
               ERASE WAIT RESP(RESP)
EXIT     DS    0H
         EXEC CICS SEND CONTROL FREEKB RESP(RESP)
         EXEC CICS RETURN
         END
```

Figure 134 (Part 8 of 8). Sample 6: CICS-to-IMS demand paged output – CICS transaction (DFH$IMSO)

Program notes for DFH$IMSO

1. The screen is erased, and the prompting map displayed at the terminal.
2. The remote system name and data are mapped in.
3. If the remote system name is given, an ALLOCATE is performed on that system
4. The name of the actual session allocated is found in the EIBRSRCE field.
5. Use the input data area of the map to advise the operator to reenter his data, correctly naming the remote system.
6. The map is recleared to ensure that all three fields are correctly reentered.
7. A transaction is to be initiated on a remote system; the name of the transaction on that system is supplied via the attach FMH, built at this point.
8. The data entered by the terminal operator is now sent across the acquired session together with the previously built attach header. The presence of the INVITE option indicates that a RECEIVE will directly follow this SEND and improves performance across the session.
9. A RECEIVE is issued against the remote system to read back the IMS reply. IMS will initially transmit an attach FMH to signify that the output will now be sent at the request of the CICS terminal operator; this header will be examined to enable the name of the IMS demand paged output queue to be found.

10. To enable the program to determine what action should next be performed on the session, the contents of the EXEC Interface Block, set by RECEIVE, will have to be retained for future reference.

11. For IMS demand paging output queues, IMS sends as its initial output an attach header. The absence of this header indicates an error on the remote system.

12. The IMS queue name is extracted. (The SESSION option is required in this instance, since the EXTRACT relates to data sent by this sample's alternate facility; without this option, the principal facility, that is, the operator terminal, would be addressed.)

13. The sample will issue three types of request to the IMS queue.

    a. Get Next
    b. Get (specific)
    c. Purge

    The queue model FMHs required to perform these functions must be completed so as to contain the appropriate IMS queue name.

14. Preceding each queue model FMH, IMS needs an attach FMH, which must contain the queue model function as its destination process name. The FMH is built at this point and will be used in conjunction with all remaining commands across the session.

15. The program has to keep a note of the page number of the logical page being currently accessed on the IMS queue; this is to enable the new page number to be correctly calculated each time a logical paging command is entered by the operator. To do this, the register "DPAGEREG" is used to hold the current number.

16. In order to open the IMS queue for output a GET NEXT command has first to be issued; this will cause the first logical page of the message to be returned to CICS. Thereafter, GET commands will be issued. It will be seen that the command is sent as a text string containing an attach FMH together with the queue model FMH. The use of the ATTACHID and FMH options should be noted.

17. The record sent by IMS (that is, a logical page) is now prepared for writing to the operator's terminal.

18. The output record received from IMS will contain the requested page record preceded by a QXFR FMH; this FMH is not required in this sample and is bypassed.

19. The presence of a FMH but no accompanying data in the message returned from IMS indicates that a request has been made for a logical page outside the dimensions of the queue size. In such instances, IMS sends a QSTATUS FMH with the QINVCUR (invalid cursor) flag set.

20. The output from IMS is in VLVB format; the IMS mapping function sets one screen output line as one logical record. The following lines of code unpack the physical record received to obtain single logical records for transmission to the terminal via BMS.

21. One logical record is inserted and a check made for further logical records.
22. The whole logical message is now sent.
23. The session-oriented information transmitted across the LU6 session by the remote transaction must now be examined to determine what action should be taken next. The syncpoint required indicator in the EXEC Interface Block is tested and if necessary the program issues its own SYNCPOINT.
24. If the EXEC Interface Block indicates that the program should now free the session, thereby denoting that the remote transaction has completed successfully and has terminated the conversation, the program now exits causing an automatic freeing of the session.
25. If the EXEC Interface Block indicates that a further RECEIVE should be made over the session, some kind of error has occurred, since normally IMS will be awaiting a paging command at this point; thus the program should be in "send" state.
26. The operator now enters a paging command as if this were a normal CICS application.
27. The command must begin with 'P/'.
28. If the current page ('P/C') is required again, go back to reuse the current page number in the GET command.
29. If the next page ('P/N') is required, add 1 to "DPAGEREG" and issue the GET command.
30. If the previous page ('P/P') is required, subtract 1 from "DPAGEREG" and issue the GET command.
31. The presence of a '+' or '-' sign is now detected, in which case the increment or decrement is found and either added to or subtracted from the logical page number "DPAGEREG". If no sign is found, the actual value typed in is the new logical page number required.
32. The page number of the logical record to be read next, held in "DPAGEREG" is stored into DPAGENO and a GET command issued.
33. If the data length field indicates that no data has been sent, the session status must be tested to determine what to do next; otherwise, the new data will be unpacked.
34. If any error is detected in the paging command entered by the operator, an error message is sent to him to prompt for the command to be reentered correctly.

35. The QSTATUS FMH indicates that IMS has detected an invalid paging request. Having sent the QSTATUS, IMS relocks the queue in question, and it is the responsibility of the queue owner, in this case the sample program, to open the queue again for further processing; this is done by issuing the original GET NEXT which will unlock the queue and resend the first logical page.

36. A queue purge request is sent to IMS to cause it to delete the demand paging queue. This command is sent using CONVERSE since IMS will respond to the purge request by returning a QSTATUS FMH and the program must allow for its receipt.

BMS mapset for sample 6 (DFH$IMS)

See Figure 133 on page 400.

Appendix C. CICS mapping to the LUTYPE6.2 architecture

This appendix shows how the LUTYPE6.2 programming language (described in the *SNA Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084) is implemented by CICS.

The appendix contains four main sections:

1. Supported option sets

   This is a table showing which APPC option sets are supported by CICS and which are not.

2. Command-mapping for basic conversations

   The CICS application programming interface for basic, or unmapped, conversations is described in "Chapter 4.6. CICS applications for logical unit type 6.2 unmapped conversations" on page 221. These tables show how the LUTYPE6.2 verbs map to the EXEC CICS commands.

3. Command-mapping for mapped conversations.

   The CICS application programming interface for mapped conversations is described in "Chapter 4.5. CICS applications for logical unit type 6.2 mapped conversations" on page 171 The full syntax of EXEC CICS commands for LUTYPE6.2 mapped conversations is given in the *CICS/MVS Application Programmer's Reference* manual. These tables show how the LUTYPE6.2 verbs map to the EXEC CICS commands.

4. CICS implementation of control operator verbs.

   This section describes how CICS implements the LUTYPE6.2 control operator verbs. It includes tables showing how these verbs map to CICS commands.

Supported option sets

Table 13 (Page 1 of 2). CICS support of APPC options sets

Set #	Set Name	Supported
101	Flush the LU's send buffer	Yes
102	Get attributes	Yes
103	Post on receipt with test for posting	No
104	Post on receipt with wait	No
105	Prepare to receive	Yes
106	Receive immediate	No
107	Program reconnect	No
108	Syncpoint services	Yes
109	Get TP properties	No
110	Get conversation type	Yes
111	Recovery from program errors detected during syncpoint	No
201	Queued allocation of a contention-winner session	No
202	Queued allocation of a conversation-group session	No
203	Immediate allocation of a session	Yes
204	Conversations between two programs located at the same LU	No
211	Session-level LU-LU verification	Yes
212	User ID verification	Yes

Table 13 (Page 2 of 2). CICS support of APPC options sets

Set #	Set Name	Supported
213	Program-supplied user ID and password	No
214	User ID authorization	Yes
215	Profile verification and authorization	No
216	Origin LU authorization	No
217	Profile passthrough	No
218	Program-supplied profile	No
219	Send persistent verification	No
220	Receive persistent verification	No
241	Send PIP data	Yes
242	Receive PIP data	Yes
243	Accounting	Yes
244	Long locks	No
245	Test for request-to-send received	Yes
246	Data mapping	No
247	FMH data	Yes
248	Syncpoint restart	No
249	Vote read-only response to a syncpoint operation	No
250	Performance improvement for syncpoint	No
290	Logging of data in a system log	No
301	Mapped conversation LU services component	No
501	CHANGE_SESSION_LIMIT verb	Yes
502	ACTIVATE_SESSION verb	Yes
504	DEACTIVATE_SESSION verb	No
505	LU-definition verbs	Yes
601	MIN_CONWINNERS_TARGET parameter	No
602	RESPONSIBLE(TARGET) parameter	No
603	DRAIN_TARGET(NO) parameter	No
604	FORCE parameter	No
605	LU-LU session limit	No
606	Locally known LU names	Yes
607	Uninterpreted LU names	No
608	Single-session reinitiation	No
610	Maximum RU size bounds	Yes
611	Session-level mandatory cryptography	No
612	Contention winner automatic activation limit	No

Command-mapping for basic conversations

The following tables show the mapping between LUTYPE6.2 verbs and CICS commands for basic conversations. See "Return codes for basic conversations" on page 422 for details of the corresponding return code mapping.

ALLOCATE	**EXEC CICS GDS ALLOCATE + EXEC CICS GDS CONNECT PROCESS**
LU_NAME(OWN)	Not supported
LU_NAME(OTHER(vble))	SYSID on ALLOCATE
MODE_NAME(vble)	MODENAME on ALLOCATE
MODE_NAME('SNASVCMG')	MODENAME on ALLOCATE
TPN(vble)	PROCNAME on CONNECT PROCESS (with PROCLENGTH)
TYPE(BASIC_CONVERSATION)	Supported by GDS
TYPE(MAPPED_CONVERSATION)	Not supported
RETURN_CONTROL(WHEN_SESSION_ALLOCATED)	Default on ALLOCATE
RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED)	Not supported
RETURN_CONTROL(IMMEDIATE)	NOQUEUE on ALLOCATE
SYNC_LEVEL	SYNCLEVEL on CONNECT PROCESS 0 - none 1 - Confirm 2 - Syncpt
SECURITY(SAME)	Default on ALLOCATE
SECURITY(PGM(USED_ID(vble)	Not supported
(PASSWORD(vble)))	Not supported
PIP(NO)	Supported by PIPLENGTH(0)
PIP(YES(vble1,vble2 ... vblen))	Supported by PIPLIST+PIPLENGTH
RESOURCE	Returned in CONVID field
RETURN_CODE	Supported

BACKOUT	**EXEC CICS SYNCPOINT ROLLBACK**

CONFIRM	**EXEC CICS GDS CONFIRM**
RESOURCE	CONVID
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG

CONFIRMED	**EXEC CICS GDS ISSUE CONFIRMATION**
RESOURCE	CONVID

DEALLOCATE		**EXEC CICS GDS SEND LAST** **+ EXEC CICS SYNCPOINT** **+ EXEC CICS GDS FREE**
TYPE=SYNC_LEVEL	None	EXEC CICS GDS SEND LAST WAIT + EXEC CICS GDS FREE
TYPE=SYNC_LEVEL	Confirm	EXEC CICS GDS SEND LAST CONFIRM + EXEC CICS GDS FREE
TYPE=SYNC_LEVEL	Syncpt	EXEC CICS GDS SEND LAST + EXEC CICS SYNCPOINT + EXEC CICS GDS FREE
TYPE=FLUSH		EXEC CICS GDS SEND LAST WAIT + EXEC CICS GDS FREE
TYPE=ABEND_PROG	Depends on setting of CDBFREE by previous command:	
	CDBFREE = X'00'	EXEC CICS GDS ISSUE ABEND + EXEC CICS GDS FREE
	CDBFREE = X'FF'	EXEC CICS GDS FREE

TYPE=ABEND_SVC
TYPE=ABEND_TIMER
		Not supported at API | Option
		Not supported at API | Set 11

TYPE=LOCAL		EXEC CICS GDS FREE
LOG_DATA(vble)		Not available at API. CICS inserts the appropriate values.
RETURN_CODE		Supported

FLUSH	**EXEC CICS GDS WAIT**

GET_ATTRIBUTES	**EXEC CICS GDS EXTRACT PROCESS** or **EXEC CICS GDS ASSIGN** or **EXEC CICS ASSIGN**
RESOURCE	CONVID
SYNC_LEVEL	SYNCLEVEL on GDS EXTRACT PROCESS 0 - none 1 - Confirm 2 - Syncpt
UOW_IDENTIFIER	See note
OWN_FULLY_QUALIFIED_LU_NAME	See note
PARTNER_LU_NAME	GDS ASSIGN PRINSYSID
PARTNER_FULLY_QUALIFIED_LU_NAME	See note
MODE_NAME	See note
USERID	ASSIGN USERID
	Note: These values are not normally required in CICS applications and are not available at the API.

GET_TYPE	**EXEC CICS GDS ASSIGN** (+ return code test)
RESOURCE	CONVID
TYPE(vble)	RETCODE clear = GDS (BASE) 03 04 = wrong conversation level

POST_ON_RECEIPT	**Not Supported**

PREPARE_TO_RECEIVE	EXEC CICS GDS SEND INVITE
TYPE=SYNC_LEVEL none	EXEC CICS GDS SEND INVITE WAIT
TYPE=SYNC_LEVEL confirm	EXEC CICS GDS SEND INVITE CONFIRM
TYPE=SYNC_LEVEL syncpt	EXEC CICS GDS SEND INVITE + EXEC CICS SYNCPOINT
TYPE=FLUSH	EXEC CICS GDS SEND INVITE WAIT
LOCKS(SHORT)	Defaulted
LOCKS(LONG)	Not Supported
RETURN_CODE	Supported

RECEIVE_AND_WAIT	**EXEC CICS GDS RECEIVE** (for both LL and BUFFER)
RESOURCE	CONVID field
FILL(BUFFER)	BUFFER option
FILL(LL)	LLID option
LENGTH(vble) Input	MAXFLENGTH option
LENGTH(vble) Output	FLENGTH option
RETURN CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG
DATA	INTO or SET option
WHAT_RECEIVED	CICS Settings
CONFIRM	CDBCONF + CDBRECV
CONFIRM_DEALLOCATE	CDBCONF + CDBFREE [+ CDBRECV]
CONFIRM_SEND	CDBCONF
DATA	FLENGTH field ¬= 0 [+ CDBRECV]
DATA_COMPLETE	CDBCOMPL [+ CDBRECV]
DATA_INCOMPLETE	¬CDBCOMPL [+ CDBRECV]
LL_TRUNCATED	RETCODE = X'0310....'
SEND	¬CDBRECV
TAKE_SYNCPT	CDBSYNC + CDBRECV
TAKE_SYNCPT_DEALLOCATE	CDBSYNC + CDBFREE [+ CDBRECV]
TAKE_SYNCPT_SEND	CDBSYNC

Notes:

1. The mapping of RECEIVE_AND_WAIT to EXEC CICS RECEIVE is not always one-to-one.

   When a CICS RECEIVE command is issued, CICS returns all the information and data (the DATA, the WHAT_RECEIVED flags, and the RETURN_CODE) at once. On completion of a CICS command, more than one indicator may be set, as shown in the WHAT_RECEIVED mapping above. It may be necessary to perform more than one subsequent command to honor the actions required by the indicators. For this reason, the action flags must be saved when they are received, and then acted on one by one. If the same data area is used for CONVDATA on successive GDS commands, the flags will be overwritten and lost.

   LU6.2 does not work this way; a RECEIVE_AND_WAIT verb returns either data or information about the conversation state (as indicated by WHAT_RECEIVED), but never both.

   It is necessary to program round this difference in philosophy when translating LU6.2 verbs into CICS commands.

2. LU6.2 allows a RECEIVE_AND_WAIT to be issued immediately after an ALLOCATE verb. When you are writing base conversations in CICS, however, you must supply the PREPARE_TO_RECEIVE explicitly, as follows:

```
ALLOCATE                EXEC CICS GDS ALLOCATE
                       +EXEC CICS CONNECT PROCESS
(Required by CICS)      EXEC CICS GDS SEND INVITE WAIT
RECEIVE_AND_WAIT        EXEC CICS GDS RECEIVE
```

REQUEST_TO_SEND	**EXEC CICS GDS ISSUE SIGNAL**
RESOURCE	CONVID field

SEND_DATA	**EXEC CICS GDS SEND**
RESOURCE	CONVID field
DATA	FROM option
LENGTH	FLENGTH option
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG

SEND_ERROR	**EXEC CICS GDS ISSUE ERROR**
RESOURCE	CONVID field
TYPE(PROG)	Default
TYPE(SVC)	See note
LOG_DATA	See note
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG

Note:

TYPE(SVC) and LOG_DATA are not available at the API. CICS inserts appropriate values.

SYNCPT	**EXEC CICS SYNCPOINT**
RETURN_CODE	Zero - Control returned to program. Non-zero - CICS takes action; normally to abend the task and backout the UOW.
REQUEST_TO_SEND_RECEIVED	Returned in CDBSIG on next command.

Note:

EXEC CICS SYNCPOINT is not a GDS command.

For certain specialized applications, the PREPARE flow (the first flow in SYNCPT exchanges) may be sent for a particular conversation by using the command:

```
EXEC CICS GDS ISSUE PREPARE
```

This enables any outstanding messages in the network (for example, SEND ERROR) to be received before proceeding, or deciding not to proceed, with the full SYNCPT.

WAIT	Not Supported

Return codes for basic conversations

LU6.2 RETURN_CODE	**CICS Return Codes**
OK	CDBERR and RETCODE are zero
ALLOCATION_ERROR	
Local Allocation Failures:	CICS is unable to allocate a session for an ALLOCATE command
ALLOCATION_FAILURE_NO_RETRY	RETCODE = 01.... The second and subsequent bytes give further information.
ALLOCATION_FAILURE_RETRY	For temporary problems, CICS will wait in the ALLOCATE command until the problem has cleared and then continue. See also the UNSUCCESSFUL return code, which relates to the NOQUEUE option on the CICS ALLOCATE command.
Remote Allocation Failures:	These will be returned to the program after the CONNECT PROCESS command has been issued, and the remote system has been unable to start the requested task. They may be returned on any subsequent command that relates to the session in use.
CONVERSATION_TYPE_MISMATCH	CDBERRCD = 10086034
PIP_NOT_ALLOWED	CDBERRCD = 10086031
PIP_NOT_SPECIFIED_CORRECTLY	CDBERRCD = 10086032
SECURITY_NOT_VALID	CDBERRCD = 080F6051
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	CDBERRCD = 10086041
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	RETCODE = 030C Note: CICS remembers SYNC_LEVEL negotiated at Bind time and does not permit a request to be sent for a Sync Level not supported by the remote LU.
TPN_NOT_RECOGNIZED	CDBERRCD = 10086021
TRANS_PGM_NOT_AVAIL_NO_RETRY	CDBERRCD = 084C0000
TRANS_PGM_NOT_AVAIL_RETRY	CDBERRCD = 084B6031
BACKED_OUT	CDBERRCD = 08240000
DEALLOCATE_ABEND_PROG	CDBERRCD = 08640000
DEALLOCATE_ABEND_SVC	CDBERRCD = 08640001
DEALLOCATE_ABEND_TIMER	CDBERRCD = 08640002
DEALLOCATE_NORMAL	CDBFREE + ¬CDBERR

LU6.2 RETURN_CODE	CICS Return Codes
PARAMETER_ERROR	RETCODE = 01 0C .. This return code relates ONLY to the ALLOCATE command (for CICS). It is given when an invalid LU name or MODE name has been specified. The third byte gives additional information.

PROG_ERROR_NO_TRUNC
PROG_ERROR_TRUNC
PROG_ERROR_PURGING
	CDBERRCD = 08890000 (RECEIVE Only)
	CDBERRCD = 08890001
	CDBERRCD = 08890000

RESOURCE_FAILURE_RETRY RESOURCE_FAILURE_NO_RETRY	CDBERRCD = A000 CDBERRCD = A000

SVC_ERROR_NO_TRUNC
SVC_ERROR_TRUNC
SVC_ERROR_PURGING
	CDBERRCD = 08890100 (RECEIVE Only)
	CDBERRCD = 08890101
	CDBERRCD = 08890100

UNSUCCESSFUL This return code relates ONLY to the LU6.2 ALLOCATE verb with RETURN_CONTROL(IMMEDIATE) specified. This is implemented in CICS with the NOQUEUE option on the ALLOCATE command	RETCODE = 01 04 04 Control returned to the program because a session was not immediately available.
Note: In all cases, where a value for CDBERRCD is given, CDBERR will be set to X'FF'. It is intended that the program should first test CDBERR and then examine CDBERRCD if additional information is required.	

Command-mapping for mapped conversations

The following tables show the mapping between LUTYPE6.2 verbs and CICS commands for mapped conversations. See "Return codes for mapped conversations" on page 430 for details of the corresponding return code mapping.

MC_ALLOCATE	**EXEC CICS ALLOCATE** **+ EXEC CICS CONNECT PROCESS**
LU_NAME(OWN)	Not supported
LU_NAME(OTHER(vble))	SYSID on ALLOCATE
MODE_NAME(vble)	MODENAME on ALLOCATE
TPN(vble)	PROCNAME on CONNECT PROCESS (with PROCLENGTH)
RETURN_CONTROL(WHEN_SESSION_ALLOCATED)	Default on ALLOCATE
RETURN_CONTROL(DELAYED_ALLOCATION_PERMITTED)	Not supported
RETURN_CONTROL(IMMEDIATE)	NOQUEUE on ALLOCATE
SYNC_LEVEL	SYNC_LEVEL on CONNECT PROCESS 0 -- none 1 - Confirm 2 -- Syncpt
SECURITY(SAME)	Default on ALLOCATE
SECURITY(PGM(USED_ID(vble)	Not supported
(PASSWORD(vble)))	Not supported
PIP(NO)	Supported by PIPLENGTH(0)
PIP(YES(vble1,vble2 ... vblen))	Supported by PIPLIST+PIPLENGTH
RESOURCE	Returned in CONVID field
RETURN_CODE	Supported

BACKOUT	**EXEC CICS SYNCPOINT ROLLBACK**

MC_CONFIRM	**EXEC CICS SEND CONFIRM**
RESOURCE	CONVID
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

MC_CONFIRMED	**EXEC CICS ISSUE CONFIRMATION**
RESOURCE	CONVID

MC_DEALLOCATE		**EXEC CICS SEND LAST** + **EXEC CICS SYNCPOINT** + **EXEC CICS FREE**
TYPE=SYNC_LEVEL	None	EXEC CICS SEND LAST WAIT + EXEC CICS FREE
TYPE=SYNC_LEVEL	Confirm	EXEC CICS SEND LAST CONFIRM + EXEC CICS FREE
TYPE=SYNC_LEVEL	Syncpt	EXEC CICS SEND LAST + EXEC CICS SYNCPOINT + EXEC CICS FREE
TYPE=FLUSH		EXEC CICS SEND LAST WAIT + EXEC CICS FREE
TYPE=ABEND_PROG	Depends on setting of EIBFREE by previous command:	
	EIBFREE = X'00'	EXEC CICS ISSUE ABEND + EXEC CICS FREE
	EIBFREE = X'FF'	EXEC CICS FREE
TYPE=LOCAL		EXEC CICS FREE
RETURN_CODE		Supported

MC_FLUSH	**EXEC CICS WAIT** or **EXEC CICS SEND WAIT**

MC_GET_ATTRIBUTES	**EXEC CICS EXTRACT PROCESS** or **EXEC CICS ASSIGN**
RESOURCE	CONVID on EXTRACT PROCESS
SYNC_LEVEL	SYNCLEVEL on EXTRACT PROCESS 0 - none 1 - Confirm 2 - Syncpt
UOW_IDENTIFIER	See note
OWN_FULLY_QUALIFIED_LU_NAME	See note
PARTNER_LU_NAME	ASSIGN PRINSYSID
PARTNER_FULLY_QUALIFIED_LU_NAME	See note
MODE_NAME	See note
USERID	ASSIGN USERID
	Note: These values are not normally required in CICS applications and are not available at the API.

MC_GET_TYPE	**(Examine EIBRSRCE)**
RESOURCE	EIBRSRCE
TYPE(vble)	EIBRSRCE set - mapped EIBRSRCE not set - not mapped

MC_POST_ON_RECEIPT	**Not Supported**

MC_PREPARE_TO_RECEIVE	**EXEC CICS SEND INVITE**
TYPE=SYNC_LEVEL none	EXEC CICS SEND INVITE WAIT
TYPE=SYNC_LEVEL confirm	EXEC CICS SEND INVITE CONFIRM
TYPE=SYNC_LEVEL syncpt	EXEC CICS SEND INVITE + EXEC CICS SYNCPOINT
TYPE=FLUSH	EXEC CICS SEND INVITE WAIT
LOCKS(SHORT)	Defaulted
LOCKS(LONG)	Not Supported
RETURN_CODE	Supported

MC_RECEIVE_AND_WAIT		EXEC CICS RECEIVE [NOTRUNCATE]
RESOURCE		CONVID field
FILL(BUFFER)		BUFFER option
FILL(LL)		LLID option
LENGTH(vble) Input		MAXFLENGTH option
LENGTH(vble) Output		FLENGTH option
RETURN_CODE		Supported
REQUEST_TO_SEND_RECEIVED		Returned in EIBSIG
DATA		INTO or SET option
MAP_NAME		Not supported
WHAT_RECEIVED		CICS Settings
	CONFIRM	EIBCONF + EIBRECV
	CONFIRM_DEALLOCATE	EIBCONF + EIBFREE [+ EIBRECV]
	CONFIRM_SEND	EIBCONF
	DATA_COMPLETE	EIBCOMPL [+ EIBRECV]
	DATA_INCOMPLETE	¬EIBCOMPL [+ EIBRECV]
	FMH_DATA_COMPLETE	EIBFMH + EIBCOMPL [+ EIBRECV]
	FMH_DATA_INCOMPLETE	EIBFMH + ¬EIBCOMPL [+ EIBRECV]
	SEND	¬EIBRECV + no other flags
	TAKE_SYNCPT	EIBSYNC + EIBRECV
	TAKE_SYNCPT_DEALLOCATE	EIBSYNC + EIBFREE [+ EIBRECV]
	TAKE_SYNCPT_SEND	EIBSYNC

Notes:

1. The mapping of MC_RECEIVE_AND_WAIT to EXEC CICS RECEIVE is not always one-to-one.

   When a CICS RECEIVE command is issued, CICS returns all the information and data (the DATA, the WHAT_RECEIVED flags, and the RETURN_CODE) at once. On completion of a CICS command, more than one indicator may be set, as shown in the WHAT_RECEIVED mapping above. It may be necessary to perform more than one subsequent command to honor the actions required by the indicators. For this reason, the action flags must be saved when they are received (because the EIB can be overwritten by subsequent CICS commands) and then acted on one-by-one.

   LU6.2 does not work this way; a MC_RECEIVE_AND_WAIT verb returns either data or information about the conversation state (as indicated by WHAT_RECEIVED), but never both.

   It is necessary to program round this difference in philosophy when translating LU6.2 verbs into CICS commands.

2. CICS EIBCOMPL settings are applicable only if NOTRUNCATE is specified on the CICS RECEIVE command.

   If NOTRUNCATE is specified, DATA_INCOMPLETE is indicated by a zero value in EIBCOMPL. CICS will save the remaining data for retrieval by subsequent RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back.

   If the NOTRUNCATE option is not specified, DATA_INCOMPLETE is indicated by the CICS LENGERR condition, and the data remaining after the RECEIVE is discarded.

MC_REQUEST_TO_SEND	**EXEC CICS ISSUE SIGNAL**
RESOURCE	CONVID field

MC_SEND_DATA	**EXEC CICS SEND**
RESOURCE	CONVID field
DATA	FROM option
LENGTH	LENGTH option
FMH_DATA(NO)	Default
FMH_DATA(YES)	See note
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

Note:
FMH_DATA(YES) permits the sending of LU6.1 FMHs within an LU6.2 conversation (for example, when running a CICS program which was originally written for use on LU6.1). An LU6.1 FMH may be built either by using the EXEC CICS BUILD ATTACH command, prior to issuing the EXEC CICS SEND command, or by building the FMH within the program, putting it the output area, and specifying the FMH option on the SEND command. Either of these two actions is equivalent to specifying FMH_DATA(YES).

MC_SEND_ERROR	**EXEC CICS ISSUE ERROR**
RESOURCE	CONVID field
RETURN_CODE	Supported
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG

SYNCPT	**EXEC CICS SYNCPOINT**
RETURN_CODE	Zero - Control returned to program. Non-zero - CICS takes action; normally to abend the task and backout the UOW.
REQUEST_TO_SEND_RECEIVED	Returned in EIBSIG on next command.

Note:

For certain specialized applications, the PREPARE flow (the first flow in SYNCPT exchanges) may be sent for a particular conversation by using the command:

```
EXEC CICS ISSUE PREPARE
```

This enables any outstanding messages in the network (for example, SEND ERROR) to be received before proceeding, or deciding not to proceed, with the full SYNCPT.

WAIT	**Not Supported**

Return codes for mapped conversations

LU6.2 RETURN_CODE	**CICS Return Codes**
OK	EIBERR zero + INVREQ not raised
ALLOCATION_ERROR	
Local Allocation Failures:	CICS is unable to allocate a session for an ALLOCATE command
ALLOCATION_FAILURE_NO_RETRY	SYSIDERR raised The second and subsequent bytes of EIBRCODE give further information.
ALLOCATION_FAILURE_RETRY	SYSBUSY raised if there is a HANDLE for it. Otherwise, CICS queues the request until a session is available. See also the UNSUCCESSFUL return code, which relates to the NOQUEUE option on the CICS ALLOCATE command.
Remote Allocation Failures:	These will be returned to the program after the CONNECT PROCESS command has been issued, and the remote system has been unable to start the requested task. They may be returned on any subsequent command that relates to the session in use.
CONVERSATION_TYPE_MISMATCH	TERMERR (EIBERRCD = 10086034)
PIP_NOT_ALLOWED	TERMERR (EIBERRCD = 10086031)
PIP_NOT_SPECIFIED_CORRECTLY	TERMERR (EIBERRCD = 10086032)
SECURITY_NOT_VALID	TERMERR (EIBERRCD = 080F6051)
SYNC_LEVEL_NOT_SUPPORTED_BY_PGM	TERMERR (EIBERRCD = 10086041)
SYNC_LEVEL_NOT_SUPPORTED_BY_LU	INVREQ (EIBRCODE = E0 00 0C) Note: CICS remembers SYNC_LEVEL negotiated at Bind time and does not permit a request to be sent for a Sync Level not supported by the remote LU.
TPN_NOT_RECOGNIZED	TERMERR (EIBERRCD = 10086021)
TRANS_PGM_NOT_AVAIL_NO_RETRY	TERMERR (EIBERRCD = 084C0000)
TRANS_PGM_NOT_AVAIL_RETRY	TERMERR (EIBERRCD = 084B6031)
BACKED_OUT	EIBSYNRB (EIBERRCD = 08240000)
DEALLOCATE_ABEND	TERMERR (EIBERRCD = 08640000)
DEALLOCATE_NORMAL	EIBFREE + ¬EIBERR
FMH_DATA_NOT_SUPPORTED	TERMERR (EIBERRCD = 08890100)
MAP_EXECUTION_FAILURE MAP_NOT_FOUND MAPPING_NOT_SUPPORTED	Not applicable. Map requests are not sent because the option is not supported.

LU6.2 RETURN_CODE	**CICS Return Codes**
PARAMETER_ERROR	This return code relates ONLY to the CICS ALLOCATE command. It is given when an invalid LU name or mode name has been specified.
PARAMETER_ERROR (Invalid LU name)	SYSIDERR (EIBRCODE = D0 04 .. or D0 0C ..
PARAMETER_ERROR (Invalid mode name)	CBIDERR raised for invalid PROFILE on ALLOCATE command

PROG_ERROR_NO_TRUNC
PROG_ERROR_PURGING
	EIBERRCD = 08890000 (RECEIVE Only)
	EIBERRCD = 08890000

RESOURCE_FAILURE_RETRY RESOURCE_FAILURE_NO_RETRY	EIBERRCD = A000 EIBERRCD = A000
UNSUCCESSFUL This return code relates ONLY to the LU6.2 ALLOCATE verb with RETURN_CONTROL(IMMEDIATE) specified. This is implemented in CICS with the NOQUEUE option on the ALLOCATE command	SYSBUSY (EIBRCODE = D3) Control returned to the program because a session was not immediately available.
Note: In all cases, where a value for EIBERRCD is given, EIBERR will be set to X'FF'. It is intended that the program should first test EIBERR and then examine EIBERRCD if additional information is required.	

CICS implementation of control operator verbs

CICS supports control operator verbs in a variety of ways.

Some verbs are supported by the CICS master terminal transaction CEMT. The relevant CEMT commands are:

```
CEMT INQUIRE CONNECTION
CEMT SET CONNECTION
CEMT INQUIRE MODENAME
CEMT SET MODENAME
```

CEMT is normally entered by an operator at a screen. It is described in the *CICS/MVS CICS-Supplied Transactions* manual.

The inquire and set operations for connections and modenames are also available at the CICS API, using the following commands:

```
EXEC CICS INQUIRE CONNECTION
EXEC CICS SET CONNECTION
EXEC CICS INQUIRE MODENAME
EXEC CICS SET MODENAME
```

Details of these commands are given in the *CICS/MVS Customization Guide*.

Some control operator verbs are supported by CICS resource definition. The definition of LUTYPE6.2 links is described in "Defining logical unit type 6.2 links" on page 116. Full details of the resource-definition syntax are given in the *CICS/MVS Resource Definition (Online)* manual and the *CICS/MVS Resource Definition (Macro)* manual.

With resource definition online, the CEDA transaction can be used to change some CONNECTION and SESSION parameters while CICS is running. With macro-level definition, the corresponding parameters are fixed for the duration of the CICS run.

The following tables show how LUTYPE6.2 control operator verbs are implemented by CICS. See "Return codes for control operator verbs" on page 439 for details of the corresponding return code mapping

Note: Wherever CEMT is shown, the equivalent form of EXEC CICS command can be used.

CHANGE_SESSION_LIMIT	**CEMT SET MODENAME**
LU_NAME(OWN)	CONNECTION( )
MODE_NAME(vble)	MODENAME( )
LU_MODE_SESSION_LIMIT(vble)	AVAILABLE( )
MIN_CONWINNERS_SOURCE(vble)	CICS negotiates a revised value, based on the AVAILABLE request and the MAXIMUM value on the DEFINE SESSIONS for the group.
MIN_CONWINNERS_TARGET(vble)	Not supported
RESPONSIBLE(SOURCE)	Yes
RESPONSIBLE(TARGET)	Not supported. CICS does not support receipt of RESP(TARGET).
RETURN_CODE	Supported

INITIALIZE_SESSION_LIMIT	**DEFINE SESSIONS** (CICS resource definition)
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( )
LU_MODE_SESSION_LIMIT(vble)	MAXIMUM( ) parameter 1
MIN_CONWINNERS_SOURCE(vble)	MAXIMUM( ) parameter 2
MIN_CONWINNERS_TARGET(vble)	Not supported
RETURN_CODE	Supported

PROCESS_SESSION_LIMIT	Automatic action by CICS-supplied transaction CLS1 when CNOS is received by a target CICS system.

LU_NAME(vble)
MODE_NAME(vble)
RETURN_CODE
	Passed internally
	Passed internally
	Supported

RESET_SESSION_LIMIT	**CEMT SET MODENAME** (for individual modegroups) or **CEMT SET CONNECTION RELEASED** (to reset all modegroups)
LU_NAME(vble)	CONNECTION( )
MODE_NAME(ALL)	SET CONNECTION( ) RELEASED
MODE_NAME(ONE(vble))	MODENAME( ) AVAILABLE(0)
MODE_NAME(ONE('SNASVCMG'))	SET CONNECTION( ) RELEASED
RESPONSIBLE(SOURCE)	Yes
RESPONSIBLE(TARGET)	Not supported
DRAIN_SOURCE	CICS supports NO
DRAIN_TARGET	CICS supports NO
RETURN_CODE	Supported

ACTIVATE_SESSION	**CEMT SET MODENAME ACQUIRED** (for individual modegroups) or **CEMT SET CONNECTION ACQUIRED** (for SNASVCMG sessions)
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( ) ACQUIRED
MODE_NAME('SNASVCMG')	Activated when CEMT SET CONNECTION ACQUIRED is issued.
RETURN_CODE	Supported

DEACTIVATE_SESSION	**Not supported**

DEFINE_LOCAL_LU	**DEFINE SESSIONS** + **DFHSNT TYPE=ENTRY** macro + **DFHSIT** macro (CICS resource definition)
FULLY_QUALIFIED_LU_NAME(vble)	cannot be specified. CICS uses the network LU name (APPLID on DFHSIT)
LU_SESSION_LIMIT NONE	not supported
LU_SESSION_LIMIT VALUE(vble)	MAXIMUM on SESSIONS
SECURITY ADD USER_ID(vble)	USERID on DFHSNT
SECURITY ADD PASSWORD(vble)	PASSWRD on DFHSNT
SECURITY ADD PROFILE(vble)	RLSKEY on DFHSNT
SECURITY DELETE USER_ID(vble)	supported by redefining DFHSNT
SECURITY DELETE PASSWORD(vble)	
MAP_NAME ADD(vble)	not supported
MAP_NAME DELETE(vble)	

DEFINE_REMOTE_LU	**DEFINE CONNECTION** (CICS resource definition)
FULLY_QUALIFIED_LU_NAME(vble)	cannot be specified
LOCALLY_KNOWN_LU_NAME NONE	mandatory on CONNECTION
LOCALLY_KNOWN_LU_NAME(vble)	CONNECTION(name)
UNINTERPRETED_LU_NAME NONE	defaults to CONNECTION(name)
UNINTERPRETED_LU_NAME(vble)	NETNAME on CONNECTION
INITIATE_TYPE INITIATE_ONLY	not supported
INITIATE_TYPE INITIATE_OR_QUEUE	
PARALLEL_SESSION_SUPPORT (YES|NO)	SINGLESESS(NO|YES) on CONNECTION
CNOS_SUPPORT (YES|NO)	always YES
LU_LU_PASSWORD NONE	default on CONNECTION
LU_LU_PASSWORD VALUE(vble)	BINDPASSWORD on CONNECTION
SECURITY_ACCEPTANCE NONE	ATTACHSEC(LOCAL)
SECURITY_ACCEPTANCE CONVERSATION	ATTACHSEC(VERIFY)
SECURITY_ACCEPTANCE ALREADY_VERIFIED	ATTACHSEC(IDENTIFY)

DEFINE_MODE	**EXEC CICS CONNECT PROCESS** + **MODEENT** macro (ACF/VTAM systems definition) + **DEFINE SESSIONS** (CICS resource definition)
FULLY_QUALIFIED_LU_NAME (vble)	cannot be specified. LU identified via CONNECTION on SESSIONS
MODE_NAME (vble)	MODENAME on SESSIONS is mapped to LOGMODE on MODEENT
SEND_PACING_WINDOW (vble)	PSNDPAC on MODEENT
RECEIVE_PACING_WINDOW (vble)	SSNDPAC on MODEENT
SEND_MAX_RU_SIZE_LOWER_BOUND (vble)	fixed at 8
SEND_MAX_RU_SIZE_UPPER_BOUND (vble)	SENDSIZE on SESSIONS
RECEIVE_MAX_RU_SIZE_LOWER_BOUND (vble)	fixed at 256
RECEIVE_MAX_RU_SIZE_UPPER_BOUND (vble)	RECEIVESIZE on SESSIONS
SYNC_LEVEL_SUPPORT CONFIRM	is always
SYNC_LEVEL_SUPPORT CONFIRM_SYNCPT	CONFIRM_SYNCPT
SINGLE_SESSION_REINITIATION OPERATOR	
SINGLE_SESSION_REINITIATION PLU	not
SINGLE_SESSION_REINITIATION SLU	supported
SINGLE_SESSION_REINITIATION PLU_OR_SLU	
SESSION_LEVEL_CRYPTOGRAPHY (YES¬NO)	ENCR on MODEENT
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	MAXIMUM(,value2) on SESSIONS

DEFINE_TP	**DEFINE TRANSACTION** (CICS resource definition)
TP_NAME (vble)	TRANSACTION(name)
STATUS ENABLED	STATUS(ENABLED)
STATUS TEMP_DISABLED	not supported
STATUS PERM_DISABLED	STATUS(DISABLED)
CONVERSATION_TYPE (MAPPED|BASIC)	supported for all TPs (determined by choice of command)
SYNC_LEVEL (NONE|CONFIRM|SYNCPT)	SYNCPT for all TPs (actual level specified on CONNECT PROCESS)
SECURITY_REQUIRED NONE	RSL(PUBLIC) + TRANSEC(1)
SECURITY_REQUIRED CONVERSATION	TRANSEC(2-64)
SECURITY_REQUIRED ACCESS PROFILE	not supported
SECURITY_REQUIRED ACCESS USER_ID	RSL(1-24)
SECURITY_REQUIRED ACCESS USER_ID_PROFILE	not supported
SECURITY_ACCESS ADD USER_ID(vble)	Transaction can be redefined
SECURITY_ACCESS ADD PROFILE(vble)	
SECURITY_ACCESS DELETE USER_ID(vble)	
SECURITY_ACCESS DELETE PROFILE(vble)	
PIP NO	supported for all TPs (specified on CONNECT PROCESS)
PIP YES(vble)	
DATA_MAPPING (NO|YES)	DATA_MAPPING (NO) for all TPs
FMH_DATA (NO|YES)	FMH_DATA (YES) for all TPs
PRIVILEGE NONE	not supported
PRIVILEGE CNOS	
PRIVILEGE SESSION_CONTROL	
PRIVILEGE DEFINE	
PRIVILEGE DISPLAY	
PRIVILEGE ALLOCATE_SERVICE_TP	

DELETE	**not supported**

DISPLAY_LOCAL_LU	**CEMT INQUIRE CONNECTION** **+ CEMT INQUIRE MODENAME** **+ CEMT INQUIRE TRANSACTION**
FULLY_QUALIFIED_LU_NAME (vble)	cannot be specified in CICS. The APPLID on DFHSIT serves as identifier for the local LU. Specific information can be had by identifying the remote LU. Otherwise, the universal id * may be used.
LU_SESSION_LIMIT (vble) LU_SESSION_COUNT (vble) SECURITY (vble) MAP_NAMES (vble) REMOTE_LU_NAMES (vble) TP_NAMES (vble)	MAXIMUM on INQ MODENAME ACTIVE on INQ MODENAME not available not supported INQ CONNECTION(*) INQ TRANSACTION(*)

DISPLAY_REMOTE_LU	**CEMT INQUIRE CONNECTION** **+ CEMT INQUIRE MODENAME**
FULLY_QUALIFIED_LU_NAME (vble)	cannot be specified. CONNECTION or MODENAME may be used.
LOCALLY_KNOWN_LU_NAME (vble) UNINTERPRETED_LU_NAME (vble) INITIATE_TYPE (vble) PARALLEL_SESSION_SUPPORT (vble) CNOS_SUPPORT (vble) SECURITY_ACCEPTANCE_LOCAL_LU (vble) SECURITY_ACCEPTANCE_REMOTE_LU (vble) MODE_NAMES (vble)	This is CONNECTION name NETNAME on INQ CONNECTION not supported not available always YES not available not available

DISPLAY_MODE	CEMT INQUIRE MODENAME + CEMT INQUIRE TERMINAL
FULLY_QUALIFIED_LU_NAME (vble)	cannot be specified.
MODE_NAME (vble)	MODENAME
SEND_PACING_WINDOW (vble)	not available
RECEIVE_PACING_WINDOW (vble)	
SEND_MAX_RU_SIZE_LOWER_BOUND (vble)	fixed at 8
SEND_MAX_RU_SIZE_UPPER_BOUND (vble)	not available
RECEIVE_MAX_RU_SIZE_LOWER_BOUND (vble)	fixed at 256
RECEIVE_MAX_RU_SIZE_UPPER_BOUND (vble)	not available
SYNC_LEVEL_SUPPORT (vble)	always CONFIRM_SYNCPT
SINGLE_SESSION_REINITIATION (vble)	not supported
SESSION_LEVEL_CRYPTOGRAPHY (vble)	not available
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	not available
LU_MODE_SESSION_LIMIT (vble)	MAXIMUM on INQ MODENAME
MIN_CONWINNERS (vble)	not supported
MIN_CONLOSERS (vble)	
TERMINATION_COUNT (vble)	
DRAIN_LOCAL_LU (vble)	
DRAIN_REMOTE_LU (vble)	
LU_MODE_SESSION_COUNT (vble)	ACTIVE on INQ MODENAME
CONWINNERS_SESSION_COUNT (vble)	not available
CONLOSERS_SESSION_COUNT (vble)	
SESSION_IDS (vble)	INQ TERMINAL(*)

DISPLAY_TP	CEMT INQUIRE TRANSACTION
TP_NAME (vble)	TRANSACTION(tranid)
STATUS (vble)	ENABLED/DISABLED
CONVERSATION_TYPE (vble)	CICS TPs allow both types
SYNC_LEVEL (vble)	CICS TPs allow all sync levels
SECURITY_REQUIRED (vble)	not available
SECURITY_ACCESS (vble)	
PIP (vble)	CICS TPs allow PIP YES and NO
DATA_MAPPING (vble)	always NO
FMH_DATA (vble)	always YES
PRIVILEGE (vble)	not supported

Return codes for control operator verbs

The CEMT INQUIRE and SET CONNECTION or MODENAME, and the equivalent EXEC CICS commands, cause CICS to start up the LU Services Manager asynchronously.

Some of the errors that can occur are detected by CEMT, or the CICS API, and are passed back immediately. Other errors are not detected until a later time, when the LU Services Manager transaction (CLS1) actually runs.

If CLS1 detects errors, it causes messages to be written to the CSMT log, as shown in Figure 135 on page 440. In normal operation, the CICS master terminal operator may not wish to inspect the CSMT log after issuing a command. So in general, the operator, after issuing a command to change parameters (for example, SET MODENAME( ) ...) should wait for a few seconds for the request to be carried out and then reissue the INQUIRE version of the command to check that the requested change has been made. In the few cases when an error actually occurs, the operator can refer to the CSMT log.

If CEMT is driven from the menu panel, it is very simple to perform the above sequence of operations.

The message used to report the results of CLS1 execution is DFH4900. The explanatory text which accompanies the message varies and is summarized below. Refer to the *CICS/MVS Messages and Codes* manual for a full description of the message. In certain cases, DFH4901 is also issued to give further information, as shown in Figure 135 on page 440.

LU6.2 RETURN_CODE	CICS Message
OK	DFH4900 result = SUCCESSFUL
ACTIVATION_FAILURE_RETRY	DFH4900 result = VALUES AMENDED + DFH4901 MAX = 0
ACTIVATION_FAILURE_NO_RETRY	DFH4900 result = VALUES AMENDED + DFH4901 MAX = 0
ALLOCATION_ERROR	Checked by CEMT. If allocation fails, SYSTEM NOT ACQUIRED is returned to the operator.
COMMAND_RACE_REJECT	DFH4900 result = RACE DETECTED
LU_MODE_SESSION_LIMIT_CLOSED	DFH4900 result = VALUES AMENDED + DFH4901 MAX = 0
LU_MODE_SESSION_LIMIT_EXCEEDED	DFH4900 result = VALUES AMENDED + DFH4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT_NOT_ZERO	DFH4900 result = VALUES AMENDED + DFH4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT_ZERO	DFH4900 result = VALUES AMENDED + DFH4901 MAX = 0
LU_SESSION_LIMIT_EXCEEDED	DFH4900 result = VALUES AMENDED + DFH4901 MAX = (negotiated value)
PARAMETER_ERROR	Checked by CEMT
REQUEST_EXCEEDS_MAX_ALLOWED	Checked by CEMT
RESOURCE_FAILURE_NO_RETRY	The LU services manager transaction (CLS1) abends with abend code ATNI.
UNRECOGNIZED_MODE_NAME	DFH4900 result = MODENAME NOT RECOGNIZED

Figure 135. CLS1-detected errors written to CSMT log

CICS deviations from LUTYPE6.2 architecture

This section describes the way in which the CICS implementation of LUTYPE6.2 differs from the architecture described in the *SNA Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269.

There are four deviations:

1. **CICS Implementation**: CICS checks incoming BIND requests for valid combinations of the CNOS indicator (BIND RQ byte 24 bit 6) and the PARALLEL-SESSIONS indicator (BIND RQ byte 24 bit 7). If an invalid combination is found (that is, PARALLEL-SESSIONS specified but CNOS not specified), CICS sends a negative response to the BIND request.

LUTYPE6.2 Architecture: The secondary logical unit (SLU), or BIND request receiver, should negotiate the CNOS and PARALLEL-SESSIONS indicators to the supported level and return them in the BIND response. The SLU should not check for an invalid combination of these indicators.

2. **CICS Implementation**: If a transaction program (TP) issues ISSUE ERROR in RECEIVE state, at a time when CICS has received change direction (CD) on the conversation but has not yet passed it to the TP, CICS will process the ISSUE ERROR as though the TP was in SEND state. That is, CICS will not send a negative response (0846) before the FMH-7(0889), but will send the FMH-7(0889) alone.

   LUTYPE6.2 Architecture: Until the TP is passed the CD, the conversation should remain in RECEIVE state. If ISSUE ERROR is issued in this state, a negative response (0846) should precede the FMH-7(0889).

3. **CICS Implementation**: CICS does not allow a transaction program (TP) to distinguish between the return codes *_ERROR_NO_TRUNC and *_ERROR_PURGING. In both cases, the TP is returned a sense code value of 08890000.

   LUTYPE6.2 Architecture: The return code in both cases is 08890000. For *_ERROR_PURGING, however, the FMH-7(0889) is preceded by a negative response (0846), indicating that some data was not received by the partner TP. The transaction programs should be able to distinguish between the two cases and perform appropriate error processing.

4. **CICS Implementation:** When the parties to a distributed transaction at SYNCLEVEL 2 exchange SYNCPOINT ROLLBACK commands, CICS applies the following rules in determining the conversation state of each party after roll back has completed:

   - The transaction which first issues SYNCPOINT ROLLBACK is left in send state, while the partner which issues SYNCPOINT ROLLBACK in response is in receive state.
   - When two connected transactions — the one initially in send state and the other in receive state — issue SYNCPOINT ROLLBACK together, then each will change to the opposite state.

   LUTYPE6.2 Architecture: Both parties to an exchange of SYNCPOINT ROLLBACK commands will revert to the same conversation state as that following the last SYNCPOINT.

Effects of CICS deviations on the transaction programmer

Deviation 2 can have an effect on transaction programs running on products other than CICS that are in conversation with CICS transaction programs. Deviation 3 can have an effect on transaction programs running on CICS.

These effects can be avoided by using the following programming conventions (the verbs and return codes referred to here are described in *SNA Transaction Programmer's Reference Manual for LU Type 6.2*):

- When writing a transaction program that will converse with a CICS transaction program, do not use the verb PREPARE_TO_RECEIVE with the

TYPE(CONFIRM) and LOCKS(LONG) parameters, or with the TYPE(SYNC_LEVEL) and LOCKS(LONG) when the SYNC_LEVEL is CONFIRM. Instead, use the LOCKS(SHORT) parameter to achieve the same function. The LOCKS(LONG) parameter provides a line-flow optimization, only.

- When writing a transaction program that will converse with a CICS transaction program, do not depend on the distinction between the the return codes PROG_ERROR_PURGING and PROG_ERROR_NO_TRUNC, and between the return codes SVC_ERROR_PURGING and SVC_ERROR_NO_TRUNC. Instead, the CICS transaction program must be coded to send additional error information after it issues the EXEC CICS ISSUE ERROR in order to describe the reason for sending the error indication.
- When writing a transaction program that will run on CICS, do not depend on the receipt of the sense data X'08890000' or X'08890100' to indicate the state of the other end of the conversation when the partner transaction program sent the error indication. Instead, the partner transaction program must be coded to send additional error information after it sends the error indication in order to describe the reason for sending the error indication.
- Because CICS may omit the negative response before an FMH-7 (ALLOCATION_ERROR), a transaction program in conversation with CICS can receive an ALLOCATION_ERROR **after** the point where the partner transaction appears to have been successfully allocated. The transaction program must therefore be written to handle this possibility.

Glossary

This glossary contains definitions of those terms and abbreviations that relate specifically to the contents of this book.

If you do not find the term you are looking for, refer to the Index or to the IBM *Vocabulary for Data Processing, Telecommunications, and Office Systems*, GC20-1699.

ANSI definitions are preceded by an asterisk (*).

The symbol "(ISO)" at the beginning of a definition indicates that it has been discussed and agreed on at meetings of the International Organization for Standardization, Technical Committee 97/Subcommittee 1, and has been approved by ANSI for inclusion in the *American National Dictionary for Information Processing*.

ACB. Access method control block (VTAM).

ACF/NCP/VS. Advanced Communication Facilities/Network Control Program/Virtual Storage.

ACF/VTAM. Advanced Communication Facilities, Virtual Telecommunications Access Method. A set of programs that control communication between terminals and application programs running under VSE/AF, OS/VS1, and MVS.

Advanced Program-to-Program Communication (APPC). The general term chosen for the LUTYPE6.2 protocol under systems network architecture (SNA).

alternate facility. An MRO or SNA session that is obtained by a transaction by means of an ALLOCATE command. Compare principal facility.

AOR. Application-owning region.

APPC. Acronym for Advanced Program-to-Program Communication.

attach header. In SNA, a function management header that causes a remote process or transaction to be attached.

back-end transaction. In synchronous transaction-to-transaction communication, a transaction that is started by a front-end transaction.

backout. See dynamic transaction backout.

bind. In SNA products, a request to activate a session between two logical units.

conversation. A sequence of exchanges over a session, delimited by SNA brackets.

data link protocol. A set of rules for data communication over a data link in terms of a transmission code, a transmission mode, and control and recovery procedures.

data security. Prevention of access to or use of stored information without authorization.

destination control table. A table describing each of the transient data destinations used in the system, or in connected CICS systems.

distributed transaction processing. The distribution of processing between transactions that communicate synchronously with one another over intersystem or interregion links.

DL/I. Data Language/I. An IBM database management facility.

domain-remote. A term used in previous releases of CICS to refer to a system in another ACF/VTAM domain. If this term is encountered in the CICS library, it can be taken to refer to any system that is accessed via SNA LU6.1 or LU6.2 links, as opposed to CICS interregion communication.

DTP. Acronym for distributed transaction processing.

dynamic transaction backout. The process of canceling changes made to stored data by a transaction following the failure of that transaction for whatever reason.

EDF. Execution (command-level) diagnostic facility for testing command-level programs interactively at a terminal.

EIB. EXEC interface block.

FCT. File control table.

file control table. A table containing the characteristics of the files accessed by file control.

FMH. Function management header.

front-end transaction. In synchronous transaction-to-transaction communication, the transaction that acquires the session to a remote system and initiates a transaction on that system. Compare back-end transaction.

function management header (FMH). In SNA, one or more headers optionally present in the leading request unit (RU) of an RU chain. It allows one session partner in a LU-LU session to send function management information to the other.

function request shipping. The process, transparent to the application program, by which CICS accesses resources when those resources are actually held on another CICS system.

GDS. Generalized data stream.

generalized data stream. The data stream used for conversations on LUTYPE6.2 sessions.

host computer. The primary or controlling computer in a data communication system.

IMS. Information Management System.

inquiry. A request for information from storage.

installation. A particular computing system, in terms of the work it does and the people who manage it, operate it, apply it to problems, service it and use the work it produces.

interactive. Pertaining to an application in which each entry calls forth a response from a system or program, as in an inquiry system or an airline reservation system. An interactive system can also be conversational, implying a continuous dialog between the user and the system.

intercommunication facilities. A generic term covering intersystem communication (ISC) and multiregion operation (MRO).

interregion communication (IRC). The method by which CICS provides communication between a CICS region and another region in the same processor. Used for multiregion operation.

intersystem communication. Communication between separate systems by means of SNA networking facilities or by means of the application-to-application facilities of an SNA access method.

interval control. The CICS element that provides time-dependent facilities.

intrapartition destination. A queue of transient data used subsequently as input data to another task within the CICS partition or region.

IRC. Acronym for Interregion Communication.

ISC. Acronym for Intersystem Communication.

| **last agent.** One of the LUTYPE6.2 conversations
| selected by a syncpoint initiator to initiate the second
| phase of the two-phase commitment process.

| **limited resources.** This is a VTAM feature, indicated
| in the BIND. When a limited resource session is
| freed, CICS unbinds it if no other task requires it.

local resource. In CICS intercommunication, a resource that is owned by the local system.

local system. In CICS intercommunication, the CICS system from whose point of view intercommunication is being discussed.

logical unit. A port through which a user gains access to the services of a network.

logical unit of work. A unit of work that can be regarded as a logically-related sequence of actions for the purposes of CICS error recovery mechanisms.

LU. See logical unit.

LUW. See logical unit of work.

LU-LU session. A session between two logical units in a SNA network.

message performance option. The improvement of ISC performance by eliminating syncpoint coordination between the connected systems.

message switching. A telecommunication application in which a message received by a central system from one terminal is sent to one or more other terminals.

MRO. Multiregion operation – communication between CICS systems in the same processor without the use of SNA networking facilities.

multiprogramming. Concurrent execution of application programs across partitions.

multiregion operation. Communication between CICS systems in the same processor without the use of SNA networking facilities.

multitasking. Concurrent execution of application programs within a CICS partition or region.

multithreading. Use, by several transactions, of a single copy of an application program.

MVS. Multiple Virtual Storage. An alternative name for OS/VS2 Release 3.

network. A configuration connecting two or more terminal installations.

network configuration. In SNA, the group of links, nodes, machine features, devices, and programs that make up a data processing system, a network, or a communication system.

nonswitched connection. A connection that does not have to be established by dialing.

Operating System/Virtual Storage (OS/VS). A compatible extension of the IBM System/360* Operating System that supports relocation hardware and the extended control facilities of System/360.

OS/VS. Operating System/Virtual Storage.

partition. A fixed size subdivision of main storage, allocated to a system task.

| **partner**. The task at the other end of an LUTYPE6.2
| conversation.

PCT. Program control table.

piggy-backing. In CICS, the later addition of an SNA indicator, such as change-direction, to output data that has been created by a SEND command but whose transmission has been deferred.

PPT. Processing program table.

principal facility. The terminal or logical unit that is connected to a transaction at its initiation. Compare alternate facility.

processing program table. A table defining all application programs valid for processing under CICS. It also keeps track of whether an application program is in main storage or not.

processor. Host processing unit.

program control table. A table defining all transactions that may be processed by the system.

program isolation. Ensuring that only one task at a time can update a particular physical segment of a DL/I database.

pseudoconversational. CICS transactions designed to appear to the operator as a continuous conversation occurring as part of a single transaction.

queue. A line or list formed by items in a system waiting for service; for example, tasks to be performed or messages to be transmitted in a message switching system.

RACF. The Resource Access Control Facility program product. An external security management facility.

region. A section of the dynamic area that is allocated to a job step or system task. In this manual, the term is used to cover partitions and address spaces as well as regions.

region-remote. A term used in previous releases of CICS to refer to a CICS system in another region of the same processor. If this term is encountered in the CICS library, it can be taken to refer to a system that is accessed via an MRO link, as opposed to an SNA LU6.1 or LU6.2 link.

remote. See remote system and remote resource.

remote resource. In CICS intercommunication, a resource that is owned by a remote system.

remote system. In CICS intercommunication, a system that the local CICS system accesses via intersystem communication or multiregion operation.

resource. Any facility of the computing system or operating system required by a job or task, and including main storage, input/output devices, the processing unit, data sets, and control or processing programs.

rollback. A programmed return to a prior checkpoint. In CICS, the cancelation by an application program of the changes it has made to all recoverable resources during the current logical unit of work.

routing transaction. A CICS-supplied transaction (CRTE) that enables an operator at a terminal owned by one CICS system to sign onto another CICS system connected by means of an MRO or LUTYPE6.2 link.

RU. request unit

SCS. SNA character stream.

SDLC. Synchronous data link control.

security. Prevention of access to or use of data or programs without authorization.

* IBM Trademark. For a list of trademarks, see page iii.

session. In CICS intersystem communication, an SNA LU-LU session. In multiregion operation, a connection between two CICS regions.

SIT. System initialization table.

SNA. Systems network architecture.

startup jobstream. A set of job control statements used to initialize CICS.

subsystem. A secondary or subordinate system.

surrogate TCTTE. In transaction routing, a TCTTE in the application-owning region that is used to represent the terminal that invoked or was acquired by the transaction.

switched connection. A connection that is established by dialing.

synchronization level. The level of synchronization (0, 1, or 2) established for an LUTYPE6.2 session.

syncpoint. Synchronization point. An intermediate point in an application program at which updates or modifications are logically complete.

| **syncpoint agent**. The end of an LUTYPE6.2 conversation that receives a syncpoint from a syncpoint initiator. The syncpoint agent can become a syncpoint initiator if it has additional LUTYPE6.2 conversations.

| **syncpoint initiator**. The end of an LUTYPE6.2 conversation that initiates a syncpoint.

system. In CICS, an assembly of hardware and software capable of providing the facilities of CICS for a particular installation.

system generation. The process of creating a particular system tailored to the requirements of a data processing installation.

system initialization table. A table containing user-specified data that will control a system initialization process.

systems network architecture (SNA). The total description of the logical structure, formats, protocols, and operational sequences for transmitting information units through a communication system. The structure of SNA allows the end users to be independent of, and unaffected by, the specific facilities used for information exchange.

task. (1) A unit of work for the processor; therefore the basic multiprogramming unit under the control program. (CICS runs as a task under VSE or OS/VS.) (2) Under CICS, the execution of a transaction for a particular user. Contrast with transaction.

task control. The CICS element that controls all CICS tasks.

TCAM. Telecommunications Access Method.

TCT. Terminal control table.

temporary storage control. The CICS element that provides temporary data storage facilities.

temporary storage table. A table describing temporary storage queues and queue prefixes for which CICS is to provide recovery.

terminal. In CICS, a device equipped with a keyboard and some kind of display, capable of sending and receiving information over a communication channel.

terminal control. The CICS element that controls all CICS terminal activity.

terminal control table. A table describing a configuration of terminals, logical units, or other CICS systems in a CICS network with which the CICS system can communicate.

terminal operator. The user of a terminal.

terminal paging. A set of commands for retrieving "pages" of an oversize output message in any order.

TIOA. Terminal input/output area.

TOR. Terminal-owning region

transaction. A transaction can be regarded as a unit of processing (consisting of one or more application programs) initiated by a single request, often from a terminal. A transaction may require the initiation of one or more tasks for its execution. Contrast with task.

transaction backout. The cancelation, as a result of a transaction failure, of all updates performed by a task.

transaction identifier. Synonym for transaction identifier. For example, a group of up to four characters entered by an operator when selecting a transaction.

transaction restart. The restart of a task after a transaction backout.

transaction routing. A CICS facility that allows terminals or logical units connected to one CICS region to initiate and to communicate with transactions in another CICS region within the same processor system or in another CICS system connected by an LUTYPE6.2 link.

transient data control. The CICS element that controls sequential data files and intrapartition data.

TST. Temporary storage table.

VSE. Virtual Storage Extended.

VTAM. See ACF/VTAM.

Index

Readers' Comments

CICS/MVS
Intercommunication Guide
Version 2 Release 1 Modification 2

Publication No. SC33-0519-02

Use this form to tell us what you think about this manual. If you have found errors in it, or you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use. To help us produce books that meet your needs, we have included a questionnaire at the front of the book. Whichever form you use, your comments will be sent to the author's department for review and appropriate action.

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.

Thank you for your time and effort. No postage stamp is necessary if mailed in USA. (Elsewhere, an IBM office or representative will be happy to forward your comments or you may mail directly to the address in the Edition Notice on the back of the title page.) Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Phone No.	

Readers' Comments
SC33-0519-02

IBM®

Cu
Alc

Fold and Tape **Please do not staple** Fold and Tape

NO POSTAGE
NECESSARY
IF MAILED IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS MAIL PERMIT NO. 40 ARMONK, NEW YORK

POSTAGE WILL BE PAID BY ADDRESSEE

International Business Machines Corporation
Department 6R1H
180 KOST ROAD
MECHANICSBURG PA 17055-0786

Fold and Tape **Please do not staple** Fold and Tape

Cut or
Along Lir

Intercommunication Guide
SC33-0519-02

Version 2.1.2

CICS/MVS

Program Number
5665-403

Printed in U.S.A.

SC33-0519-02