CICS® Transaction Server for VSE/ESA™

Intercommunication Guide

Release 1

SC33-1665-01

CICS® Transaction Server for VSE/ESA™

Intercommunication Guide

Release 1

SC33-1665-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page 287.

First Edition (June 1999)

This edition applies to Release 1 of CICS Transaction Server for VSE/ESA, program number 5648-054, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions. Make sure you are using the correct edition for the level of the product.

The CICS for VSE/ESA Version 2.3 edition remains applicable and current for users of CICS for VSE/ESA Version 2.3.

Order publications through your IBM representative or the IBM branch office serving your locality.

At the back of this publication is a page entitled “Sending your comments to IBM”. If you want to make any comments, please use one of the methods described there.

Contents

Preface

What this book is about

This book is about:

- Multiregion operation (MRO): communication between CICS® systems in the same operating system without the use of IBM® Systems Network Architecture (SNA) networking facilities[1].
- Intersystem communication (ISC): communication between a CICS Transaction Server for VSE/ESA™ system and other systems or terminals that support the logical unit type 6.1 or logical unit type 6.2 protocols of SNA. Logical unit type 6.2 protocols are also known as Advanced Program-to-Program Communication (APPC).

What is not covered by this book

The information in this book is predominantly, but not exclusively, about communication between CICS Transaction Server for VSE/ESA Release 1 and other mainframe CICS systems. If you are interested in communication between CICS Transaction Server for VSE/ESA Release 1 and non-mainframe CICS products, you will find the *CICS Family: Communicating from CICS on System/390* manual useful. For an overview of the intercommunication facilities provided on other CICS products, see the *CICS Family: Interproduct Communication* manual.

The CICS Front End Programming Interface is not described in this book, but in the *CICS Front End Programming Interface User's Guide*.

Who this book is for

This book is for customers 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.

It is also assumed that you understand SNA concepts and terminology.

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.

[1] The external CICS interface (EXCI) uses a specialized form of MRO link to support communication between VSE/ESA batch programs and CICS.

Road map

Table 1. Getting started road map

If you want to...	**Refer to...**
Read an overview of CICS intercommunication facilities	Part 1, “Concepts and facilities” on page 1
Install CICS so that it can use its intercommunication facilities	Part 2, “Installation and system definition” on page 81
Define remote systems, remote resources, and local resources that are required for intercommunication	Part 3, “Resource definition” on page 91
Write application programs that use intercommunication facilities	Part 4, “Application programming” on page 173
Tune the performance of interconnected systems	Part 5, “Performance” on page 219
Read about recovery in an intercommunication environment	Part 6, “Recovery and restart” on page 231
Refer to background technical information	“Appendixes” on page 257
Check technical terms used in this manual	“Glossary” on page 275

Notes on terminology

The terms listed in Table 2 are commonly used in the CICS Transaction Server for VSE/ESA Release 1 library. See the *CICS Glossary* for a comprehensive definition of terminology.

Table 2 (Page 1 of 2). Commonly used words and abbreviations in CICS Transaction Server for VSE/ESA Release 1

Term	**Definition (and abbreviation if appropriate)**
$(the dollar symbol)	In the programming examples in this book, the dollar symbol ($) is used as a national currency symbol. In countries where the dollar is not the national currency, the local currency should be used.
BSM	BSM is used to indicate the basic security management supplied as part of the VSE/ESA product. It is RACROUTE-compliant, and provides the following functions: • Signon security • Transaction attach security
C	The C programming language
CICSplex	A CICSplex consists of two or more regions that are linked using CICS intercommunication facilities. Typically, a CICSplex has at least one terminal-owning region (TOR), more than one application-owning region (AOR), and may have one or more regions that own the resources accessed by the AORs
CICS Data Management Facility	The new CICS Transaction Server for VSE/ESA Release 1 facility to which all statistics and monitoring data is written, generally referred to as “DMF”
CICS/VSE	The CICS product running under the VSE/ESA operating system, frequently referred to as simply “CICS”
COBOL	The COBOL programming language
DB2 for VSE/ESA	Database 2 for VSE/ESA which was previously known as “SQL/DS”.

Table 2 (Page 2 of 2). Commonly used words and abbreviations in CICS Transaction Server for VSE/ESA Release 1

Term	**Definition (and abbreviation if appropriate)**
ESM	ESM is used to indicate a RACROUTE-compliant external security manager that supports some or all of the following functions: • Signon security • Transaction attach security • Resource security • Command security • Non-terminal security • Surrogate user security • MRO/ISC security (MRO, LU6.1 or LU6.2) • FEPI security.
FOR (file-owning region)—also known as a DOR (data-owning region)	A CICS region whose primary purpose is to manage VSAM and DAM files, and VSAM data tables, through function provided by the CICS file control program.
IBM C for VSE/ESA	The Language Environment version of the C programming language compiler. Generally referred to as “C/VSE”.
IBM COBOL for VSE/ESA	The Language Environment version of the COBOL programming language compiler. Generally referred to as “COBOL/VSE”.
IBM PL/I for VSE/ESA	The Language Environment version of the PL/I programming language compiler. Generally referred to as “PL/I VSE”.
IBM Language Environment for VSE/ESA	The common runtime interface for all LE-conforming languages. Generally referred to as “LE/VSE”.
PL/I	The PL/I programming language
VSE/POWER	Priority Output Writers Execution processors and input Readers. The VSE/ESA spooling subsystem which is exploited by the report controller.
VSE/ESA System Authorization Facility	The new VSE facility which enables the new security mechanisms in CICS TS for VSE/ESA R1, generally referred to as “SAF”
VSE/ESA Central Functions component	The new name for the VSE Advanced Function (AF) component
VSE/VTAM	“VTAM”

Part 1. Concepts and facilities

Table 3. Concepts road map

If you want to...	Refer to...
Read an overview of CICS intercommunication methods and facilities	Chapter 1, “Introduction to CICS intercommunication” on page 3
Read about CICS-to-CICS communication within the same operating system	Chapter 2, “Multiregion operation” on page 11
Read about communication between CICS and non-CICS systems, or between CICS systems that are not in the same operating system	Chapter 3, “Intersystem communication” on page 15
Learn about function shipping	Chapter 4, “CICS function shipping” on page 21
Learn about DPL	Chapter 5, “CICS distributed program link” on page 33
Learn about the external CICS interface	Chapter 6, “The external CICS interface” on page 39
Learn about asynchronous processing	Chapter 7, “Asynchronous processing” on page 41
Learn about transaction routing	Chapter 8, “CICS transaction routing” on page 53
Learn about DTP	Chapter 9, “Distributed transaction processing” on page 71

Chapter 1. Introduction to CICS intercommunication

It is assumed that you are familiar with the use of CICS as a single system, with associated data resources and a network of terminals. In this book, 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 communication between a **local** CICS system and a **remote** system, which may or may not be another CICS system.

This chapter contains the following topics:

- "Intercommunication methods"
- "Intercommunication facilities" on page 4
- "Using CICS intercommunication" on page 7

Intercommunication methods

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

Multiregion operation

For CICS-to-CICS communication, CICS provides an **interregion communication** facility that is independent of SNA access methods. This form of communication is called **multiregion operation** (MRO). CICS Transaction Server for VSE/ESA systems can use MRO to communicate with other CICS systems that reside in the same host operating system.[2] Thus, CICS Transaction Server for VSE/ESA Release 1 can use MRO to communicate with:

- Other CICS Transaction Server for VSE/ESA Release 1 systems
- CICS/VSE® Version 2 Release 3 systems

Intersystem communication

For communication between CICS and non-CICS systems, or between CICS systems that are not in the same operating system, you normally require an SNA access method, such as ACF/VTAM®, to provide the necessary communication protocols.[2] Communication between systems through SNA is called **intersystem communication (ISC)**.

Note: This form of communication can also be used between CICS systems in the same operating system, but MRO provides a more efficient alternative.

The SNA protocols that CICS uses for intersystem communication are those of Logical Unit Type 6 (otherwise known as LUTYPE 6.1) and of Advanced Program-to-Program Communication (APPC, otherwise known as LUTYPE 6.2). Additional information on this topic is given in Chapter 3, "Intersystem communication" on page 15.

[2] The external CICS interface (EXCI) uses a specialized form of MRO link to support communication between VSE/ESA batch programs and CICS.

CICS Transaction Server for VSE/ESA Release 1 can use ISC to communicate with:

- Other CICS Transaction Server for VSE/ESA Release 1 systems
- CICS/VSE Version 2
- CICS Transaction Server for OS/390®
- CICS/ESA® Version 4
- CICS/ESA Version 3
- CICS/MVS® Version 2
- CICS on Open Systems
- CICS for OS/2®
- CICS for Windows NT®
- CICS/400®
- IMS/ESA® Version 5
- IMS/ESA Version 6
- Any system that supports **Advanced Program-to-Program Communication** (APPC) protocols (LU6.2).

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. It can also establish links to other systems, and thereby gain access to remote resources. This mechanism allows resources to be distributed among and shared by the participating systems.

CICS intercommunication provides these basic types of facility:

- Function shipping
- Distributed program link (DPL)
- The external CICS interface
- Asynchronous processing
- 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 4 on page 5.

Table 4. CICS provision of intercommunication facilities

Facility	**Intercommunication**				
	ISC Intersystem (via ACF/VTAM)				**IRC Interregion**
	LUTYPE6.2 (APPC)		**LUTYPE6.1**		**MRO**
	CICS	**non-CICS**	**CICS**	**IMS™**	**CICS**
Function Shipping	Yes	No	Yes	No	Yes
Distributed program link	Yes	No	No	No	Yes
External CICS interface	No	No	No	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 shipping

CICS function shipping lets an application program access a resource owned by, or accessible to, another CICS system. Both read and write access are permitted, and facilities for exclusive control and recovery and restart are provided.

The remote resource can be:

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

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.

Distributed program link (DPL)

CICS distributed program link enables a CICS program (the client program) to call another CICS program (the server program) in a remote CICS region. Here are some of the reasons you might want to design your application to use DPL:

- To separate the end-user interface (for example, BMS screen handling) from the application business logic, such as accessing and processing data, to enable parts of the applications to be ported from host to workstation more readily.
- To obtain performance benefits from running programs closer to the resources they access, and thus reduce the need for repeated function shipping requests.
- In many cases, DPL offers a simple alternative to writing distributed transaction processing (DTP) applications.

The external CICS interface

The external CICS interface is an application programming interface (API) that enables a VSE/ESA application program (running in a VSE/ESA address space) to call a CICS application program (running in a CICS Transaction Server for VSE/ESA Release 1 address space) and to pass and receive data using a communications area. The CICS program is invoked as if linked-to by another CICS program.

You can think of the external CICS interface as a specialized form of distributed program link.

Asynchronous processing

Asynchronous processing allows a CICS transaction to initiate a transaction in a remote system and to pass data to it. The remote transaction can then initiate a transaction in the local system to receive the reply.

The reply is not necessarily returned to the **task** that initiated the remote transaction, and no direct tie-in between requests and replies is possible (other than that provided by user-defined fields in the data). The processing is therefore called **asynchronous**.

CICS transaction routing

CICS transaction routing permits a transaction and an associated terminal to be owned by different CICS systems. Transaction routing can take the following forms:

- A terminal that is owned by one CICS system can run a transaction owned by another CICS system.
- A transaction that is started by automatic transaction initiation (ATI) can acquire a terminal owned by another CICS system.
- A transaction that is running in one CICS system can allocate a session to an APPC device owned by another CICS system.

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

Distributed transaction processing (DTP)

When CICS arranges function shipping, distributed program link, asynchronous transaction processing, or transaction routing for you, it establishes a logical data link with a remote system. A data exchange between the two systems then follows. This data exchange is controlled by CICS-supplied programs, using APPC, LUTYPE6.1, or MRO protocols. The CICS-supplied programs issue commands to allocate conversations, and send and receive data between the systems. Equivalent commands are available to application programs, to allow applications to converse with CICS or non-CICS applications. The technique of distributing the functions of a transaction over several transaction programs within a network is called **distributed transaction processing (DTP)**.

DTP allows a CICS transaction to communicate with a transaction running in another system. The transactions are designed and coded specifically to communicate with each other, and thereby to use the intersystem link with maximum efficiency.

The communication in DTP 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 associated. This contrasts with the asynchronous processing described previously.

Using CICS intercommunication

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

Multiregion operation makes it possible for 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 errors in other regions. Various possible applications of MRO are described in Chapter 2, “Multiregion operation” on page 11.

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

Figure 1 on page 8 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 toward the activities of that area, with its own network of terminals able to inquire on 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.

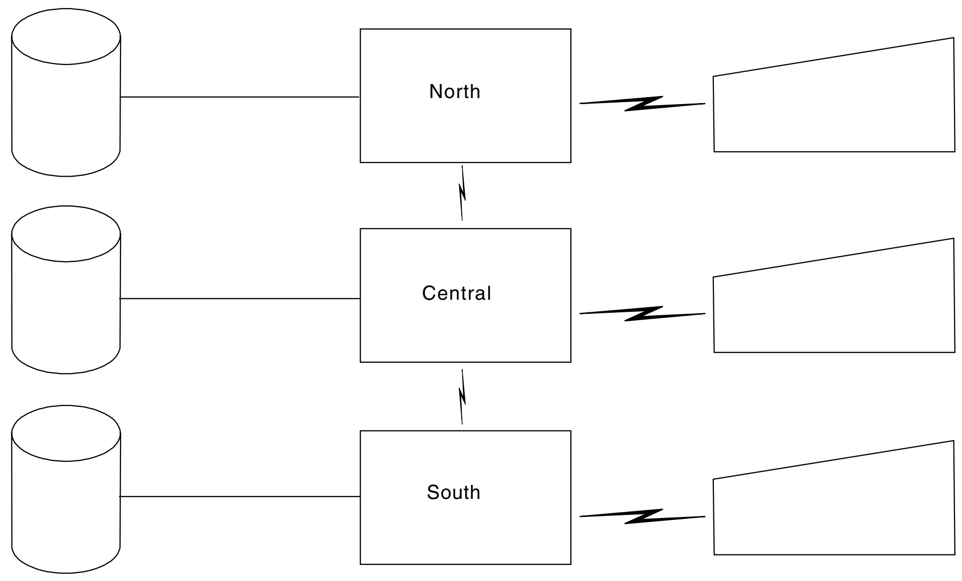
Using CICS function shipping, application programs can be written to be independent of the actual location of the data, and able to run in any of the regional centers. An example of this type of application is the verification of credit against customer accounts.

Connecting divisions within an organization

Some users are organized by division, 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.

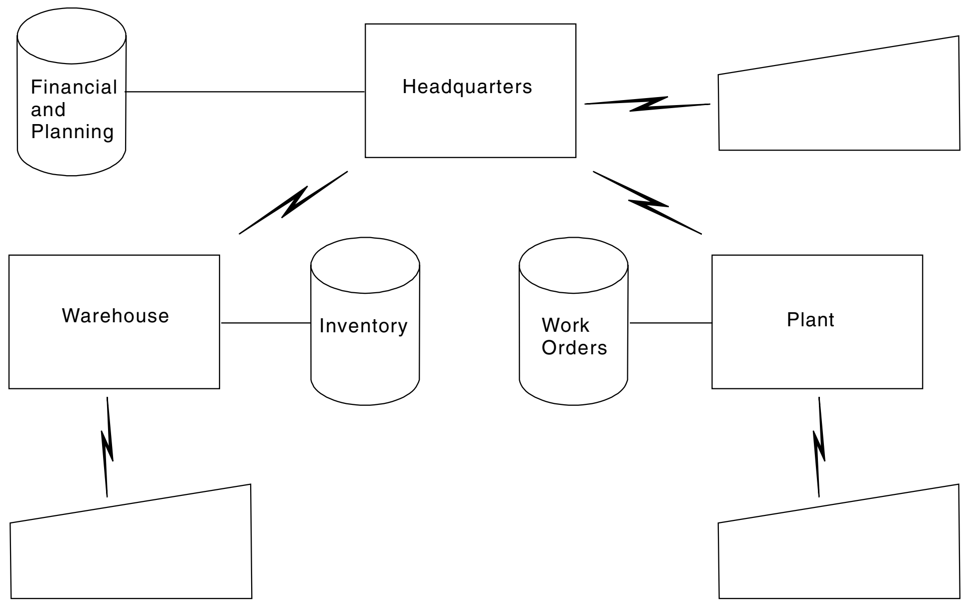
The applications and data can be hierarchically organized, with summary and central data at the headquarters site and detail data at plant sites. Alternatively, the applications and data can be distributed across the divisional locations, with planning and financial data and applications at the headquarters 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



- Database 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

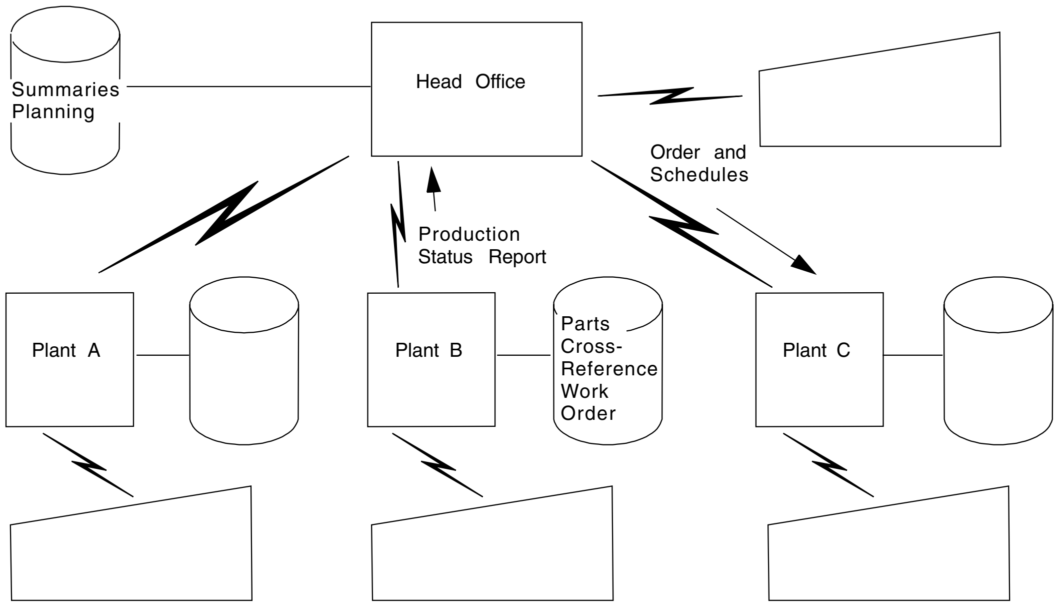
Connecting divisions: distributed applications and data



- Database 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

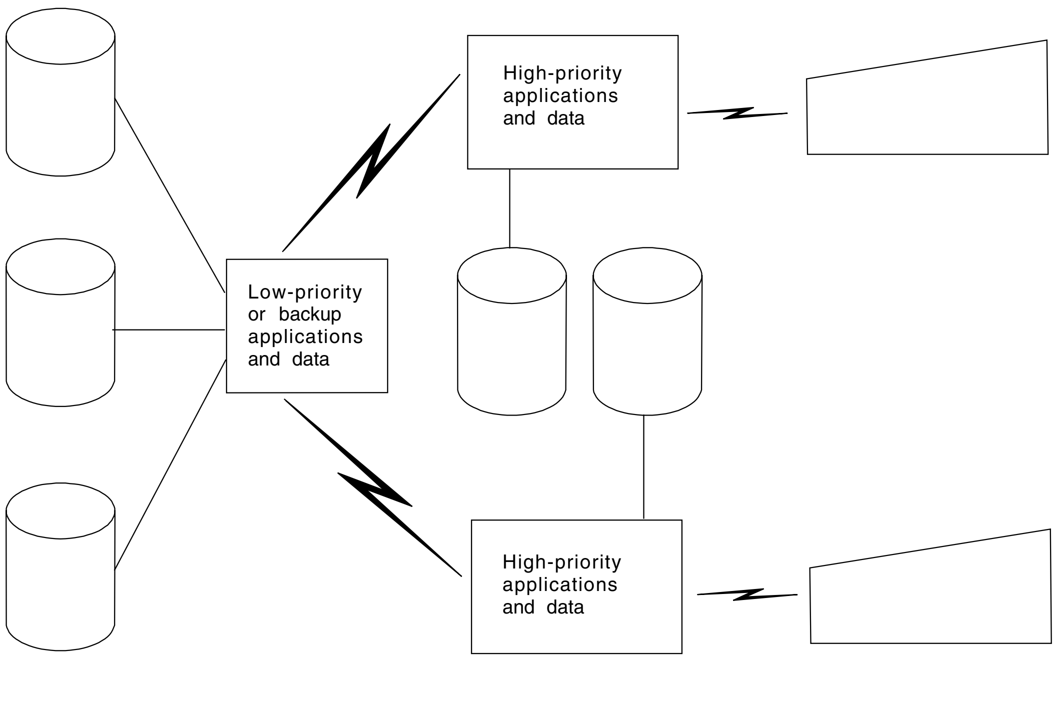
Figure 1 (Part 1 of 2). Examples of distributed resources

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



- Improved response through distributed processing

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

Chapter 2. Multiregion operation

This chapter contains the following topics:

- “Overview of MRO”
- “Facilities available through MRO”
- “Applications of multiregion operation”
- “Conversion from single-region system” on page 13

Overview of MRO

CICS multiregion operation (MRO) enables CICS systems that are running in the same operating system to communicate with each other. MRO does not support communication between a CICS system and a non-CICS system such as IMS.[3]

ACF/VTAM and SNA networking facilities are not required for MRO. The support within CICS that enables 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 shared virtual area (SVA) of VSE/ESA.

Installation of CICS multiregion operation is described in Chapter 10, “Installation considerations for multiregion operation” on page 83.

Facilities available through MRO

The intercommunication facilities available through MRO are:

- Function shipping
- Asynchronous processing
- Distributed program link
- The external CICS interface (EXCI)
- Transaction routing
- Distributed transaction processing

These are described under “Intercommunication facilities” on page 4.

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

Applications of multiregion operation

This section describes some typical applications of multiregion operation.

[3] The external CICS interface (EXCI) uses a specialized form of MRO link to support communication between VSE/ESA batch programs and CICS.

Program development

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

By using function 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 started and ended as required, without interrupting production work. During the cutover of the new programs into production, 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, 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 partition 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

You can use MRO to guard against unreliable applications that might otherwise bring down the system or disable other applications.

For example, you could define two CICS regions, one of which owns applications that you have identified as unreliable, and the other the reliable applications and the database. The fewer the applications that run in the database-owning region, the more reliable this region will be. However, 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 enables you to create a **CICSplex** in which the various departments of an organization have their own CICS systems. Each can start and end its own system as it requires. At the same time, each can have access to other departments' data, with access 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, because, with transaction routing, any terminal could run a transaction on any system.

Multiprocessor performance

Using MRO, you can take advantage of a multiprocessor by linking several CICS systems into a CICSplex, and allowing any terminal to access the transactions and data resources of any of the systems. The system programmer can assign transactions and data resources to any of the connected systems to get optimum performance. Transaction routing presents the terminal user with a single system image; the user need not be aware that there is more than one CICS system.

Transaction routing is described in Chapter 8 on page 53.

Virtual storage constraint relief

In some large CICS systems, the amount of virtual storage available can become a limiting factor. In such 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.

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 shipping allows operators of terminals owned by 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 shipping must follow the rules given in Chapter 17, "Application programming for CICS function shipping" on page 177. To conform to these rules, it may sometimes be necessary to modify programs written for single-region CICS systems.

CICS transaction routing allows operators of terminals owned by one CICS region to run transactions in a connected CICS region. One use of this facility is to allow applications to continue to use function that has been discontinued in the current release of CICS. Such coexistence considerations are described in the *CICS Migration Guide*. In addition, the restrictions that apply are given in Chapter 21, "Application programming for CICS transaction routing" on page 193.

It is always 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" on page 91.

Chapter 3. Intersystem communication

The data formats and communication protocols required for communication between systems in a multiple-system environment are defined by the IBM Systems Network Architecture (SNA); CICS intersystem communication (ISC) implements this architecture.

It is assumed that you are familiar with the general concepts and terminology of SNA.

This chapter contains the following topics:

- "Connections between subsystems"
- "Intersystem sessions" on page 16
- "Establishing intersystem sessions" on page 19

Connections between subsystems

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

- ISC within a single host operating system
- ISC between physically adjacent operating systems
- ISC between physically remote operating systems

A possible configuration is shown in Figure 2.

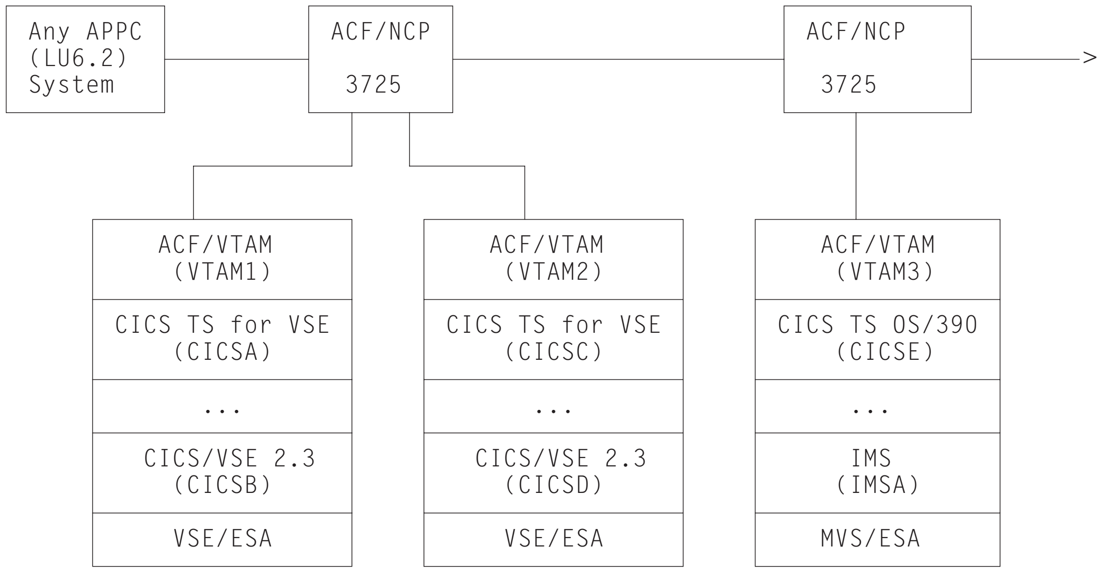


Figure 2. A possible configuration for intercommunicating systems

Single operating system

ISC within a single operating system (intrahost ISC) is possible through the application-to-application facilities of ACF/VTAM. In Figure 2 on page 15, these facilities can be used to communicate between CICSA and CICSB, between CICSC and CICSD, and between CICSE and IMSA.

In a VSE/ESA system, you can use intrahost ISC for communication between two or more CICS Transaction Server for VSE/ESA systems or between, for example, a CICS Transaction Server for VSE/ESA system and a CICS/VSE 2.3 system. (In both cases, MRO is a more efficient alternative.)

From the CICS point of view, intrahost ISC is the same as ISC between systems in different VTAM® domains.

Physically adjacent operating systems

An IBM 3725 can be configured with a multichannel adapter that permits you to connect two VTAM domains (for example, VTAM1 and VTAM2 in Figure 2 on page 15) through a single ACF/NCP/VS. This configuration may be useful for communication between:

- A production system and a local but separate test system
- Two production systems[4] with differing characteristics or requirements

Direct channel-to-channel communication is available between systems that have ACF/VTAM installed.

Remote operating systems

This is the most typical configuration for intersystem communication. For example, in Figure 2 on page 15, CICSE and IMSA can be connected to CICSA, CICSB, CICSC, and CICSD in this way. Each participating system is appropriately configured for its particular location, using Virtual Storage Extended (VSE) or Multiple Virtual Storage (MVS) CICS or IMS, and one of the ACF access methods, such as ACF/VTAM.

For a list of the CICS and non-CICS systems that CICS Transaction Server for VSE/ESA Release 1 can connect to via ISC, see page 4. For detailed information about using ISC to connect CICS Transaction Server for VSE/ESA Release 1 to other CICS products, see the *CICS Family: Communicating from CICS on System/390* manual.

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. A single logical connection can carry multiple independent sessions. Such sessions are called **parallel** sessions.

[4] The operating systems may or may not be located in the same physical box.

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

- LUTYPE6.1 sessions
- LUTYPE6.2 (APPC) sessions

Note that you must not have more than one APPC connection installed at the same time between an LU-LU pair. Nor should you have an APPC and an LUTYPE6.1 connection installed at the same time between an LU-LU pair.

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 (APPC), which was introduced later.

The characteristics of LUTYPE6 sessions are described in the Systems Network Architecture book *Sessions Between Logical Units*.

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)

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

In addition to enabling data communication between transaction-processing systems, the APPC architecture defines subsets that enable device-level products (APPC terminals) to communicate with host-level products and also with each other. APPC sessions can therefore be used for CICS-to-CICS communication, and for communication between CICS and other APPC systems or terminals.

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

Protocol boundary

The APPC protocol boundary is a generic interface between transactions and the SNA network. It is defined by formatted functions, called **verbs**, and protocols for using the verbs. Details of this SNA protocol boundary are given in the Systems Network Architecture publication *Transaction Programmer’s Reference Manual for LU Type 6.2*.

CICS provides a command-level language that maps to the protocol boundary and enables you to write application programs that hold APPC conversations. Alternatively, you may use the **Common Programming Interface Communications (CPI Communications)** of the Systems Application Architecture (SAA) environment.

Two types of APPC conversation are defined:

Mapped

In mapped conversations, the data passed to and received from the APPC application program interface is simply user data. The user is not concerned with the internal data formats demanded by the architecture.

Basic

In basic conversations, the data passed to and received from the APPC application program interface is prefixed with a header, called a GDS header. The user is responsible for building and interpreting this header. Basic 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.

Synchronization levels

The APPC architecture provides three levels of synchronization. In CICS, these levels are known as Levels 0, 1, and 2. In SNA terms, these correspond to NONE, CONFIRM, and SYNCPOINT, as follows:

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. CICS built-in synchronization does not occur at this level.

Level 2 (SYNCPOINT)

This level is the equivalent of full CICS syncpointing, including rollback. Level-1 synchronization requests can also be used.

All three levels are supported by both EXEC CICS commands and CPI Communications.

Program initialization parameter data

When a transaction initiates a remote transaction connected by an APPC session, it can send data to be received by the attached transaction. This data, called program initialization parameters (PIP), is formatted into one or more variable-length subfields according to the SNA architected rules. CPI Communications does not support PIP.

LU services manager

Multisession APPC 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; these are 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 ‘not available for allocation’ for users.

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

Class of service

The CICS implementation of APPC 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:

- Alternate routing–virtual routes for a given COS can be assigned to different physical paths (explicit routes).
- 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.
- Trunking–explicit routes can use parallel links between specific 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, APPC 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 “VTAM LOGMODE table entries for CICS” on page 86.)

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 conversation at any time. The contention loser is the LU that must use an SNA BID command (LUTYPE6.1) or LUSTATUS command (APPC) to request permission to begin a conversation.

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 APPC links, the number of contention-winning sessions is specified when the link is defined. (See “Defining APPC links” on page 101.) The contention-winning sessions are normally bound by CICS, but CICS also accepts 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.

A single session to an APPC terminal is normally defined as the contention winner, and is bound by CICS, but CICS can accept a negotiated bind in which the contention winner is changed to the loser.

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

- By CICS during CICS initialization for sessions for which AUTOCONNECT(YES) or AUTOCONNECT(ALL) has been specified. See Chapter 12, “Defining links to remote systems” on page 93.
- By a request from the CICS master terminal operator.
- By the remote system with which CICS is to communicate.

- 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 4. CICS function shipping

This chapter contains the following topics:

- "Overview of function shipping"
- "Design considerations" on page 22
- "The mirror transaction and transformer program" on page 25
- "Function shipping–examples" on page 28.

Overview of function shipping

CICS function shipping enables CICS application programs to:

- Access CICS files owned by other CICS systems by shipping file control requests.
- Access DL/I databases managed by or accessible to other CICS systems by shipping requests for 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 7, "Asynchronous processing" on page 41.

Applications can be written without regard for the location of the requested resources; they simply use file control commands, temporary storage commands, and other functions in the same 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 CICA 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 CICB. CICS changes the READ request into a suitable transmission format, and then ships it to CICB for execution.

In CICB, the request is passed to a special transaction known as the **mirror transaction**. The mirror transaction recreates the original request, issues it on CICB, and returns the acquired data to CICA.

The CICS recovery and restart facilities enable resources in remote systems to be updated, and 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 action can be taken manually or by user-written code.

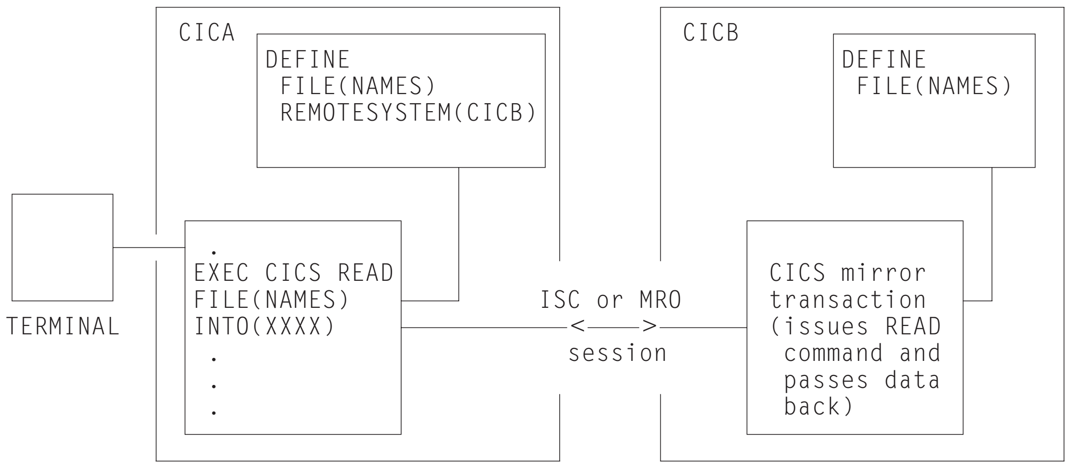

CICA
DEFINE FILE(NAMES) REMOTESYSTEM(CICB)
TERMINAL
. EXEC CICS READ FILE(NAMES) INTO(XXXX) . . .
ISC or MRO
<——>
session
CICB
DEFINE FILE(NAMES)
CICS mirror transaction (issues READ command and passes data back)

Figure 3. Function shipping

Design considerations

User application programs can run in a CICS intercommunication environment and use the intercommunication facilities without being aware of the location of the file or other resource being accessed. The location of the resource is specified in the resource definition—see Chapter 14, “Defining remote resources” on page 137.

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.

Although this may limit program independence, 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

Function shipping allows access to DAM or VSAM files located on a remote CICS system. Both read-only and update requests are allowed, and the files can be defined as recoverable in the system on which they reside. Updates to remote recoverable files are not 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.

Warning: Take care when designing systems in which remote file requests using physical record identifier values are employed, such as DAM, VSAM RBA, or files with keys not embedded in the record. You must 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 shared data tables. For information about shared data tables, see the *CICS Shared Data Tables Guide*.

DL/I

Function shipping allows a CICS transaction to access IMS/ESA DM and IMS/VS DB databases associated with a remote CICS Transaction Server for OS/390, CICS/ESA, CICS/MVS, or CICS/OS/VS system; or DL/I VSE databases associated with a remote CICS/VSE or CICS Transaction Server for VSE/ESA system. (See Chapter 1, "Introduction to CICS intercommunication" on page 3 for a list of systems with which CICS Transaction Server for VSE/ESA Release 1 can communicate.)

The IMS/ESA DM (DL/I) database associated with a remote CICS Transaction Server for OS/390 or CICS/ESA system can be a **local** database owned by the remote system, or a database accessed using IMS database control (DBCTL). To the CICS system that is doing the function shipping, this database is simply **remote**.

As with file control, updates to remote DL/I databases are not committed until the application reaches a syncpoint. With IMS/ESA DM, it is not possible to schedule more than one program specification block (PSB) for each logical unit of work, even when the PSBs are defined to be on different remote systems. Hence linked DL/I updates on different systems cannot be made in a single logical unit of work. DL/I VSE allows an application program using extended RPSB support simultaneous access to DL/I data bases located on a local system and on a remote system.

The PSB directory list (DLZACT) is used to define a PSB as being on a remote system. The remote system owns the database and the associated program communication block (PCB) definitions. When DL/I access requests are made to another processor system by a CICS Transaction Server for VSE/ESA system but no local requests are made, it is still necessary to install DL/I on the requesting system.

Temporary storage

Function shipping 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 an entry in the local temporary storage table (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 records of fixed or variable length.

Many of the uses currently made of transient data and temporary storage queues in a single CICS system can be extended to an interconnected processor 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 when it is ready, and delivered to the operator when there is a lull in entering transactions.

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

Intersystem queuing

Performance problems can occur when function shipping requests awaiting free sessions are queued in the issuing CICS region. Requests that are to be function shipped to a resource-owning region may be queued if all bound contention winner[5] sessions are busy, so that no sessions are immediately available. If the resource-owning region is unresponsive (if it is a file-owning region, it may, for example, be waiting for a system journal to be archived), the queue can become so long that the performance of the issuing region is severely impaired. Further, if the issuing CICS region is an application-owning region, its impaired performance can spread back to the terminal-owning region.

The symptoms of this impaired performance are:

- The system reaches its MXT limit, because many tasks have requests queued.
- The system becomes short-on-storage.

In either case, CICS is unable to start any new work.

CICS provides two methods of preventing these problems:

- The QUEUELIMIT and MAXQTIME options of CONNECTION definitions. You can use these to limit the number of requests that can be queued against particular remote systems, and the time that requests should wait for sessions on unresponsive connections.
- Two global user exits, XZIQUE and XISCONA. Your XZIQUE or XISCONA exit program is invoked if no contention winner session is immediately available, and can tell CICS to queue the request, or to return SYSIDERR to the application program. Its decision can be based on statistics accessible from the user exit parameter list. For programming information about writing XZIQUE and XISCONA exit programs, refer to the *CICS Customization Guide*. For information on the statistics records that are passed to your exit program, refer to the *CICS Performance Guide*.

  Note: It is recommended that you use the XZIQUE exit, rather than XISCONA. XZIQUE provides better functionality, and is of more general use than XISCONA: it is driven for transaction routing, DPL, and distributed transaction processing requests, as well as for function shipping, whereas XISCONA is driven only for function shipping. If you enable both exits, XZIQUE and

5 “Contention winner” is the terminology used for APPC connections. On MRO and LUTYPE6.1 connections, the SEND sessions (defined in the session definitions) are used for ALLOCATE requests; when all SEND sessions are in use, queuing starts.

XISCONA could both be driven for function shipping requests, which is not recommended.

For further information about controlling intersystem queues, see Chapter 23, "Intersystem session queue management" on page 221.

The mirror transaction and transformer program

CICS supplies a number of mirror transactions, some of which correspond to "architected processes" (see "Architected processes" on page 166). Details of the supplied mirror transactions are given in Chapter 15, "Defining local resources" on page 163. 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, some 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 27.

ISC function shipping

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

In the requesting system (CICA in Figure 4 on page 26), the command-level EXEC interface program (for all except DL/I requests) determines that the requested resource is on another system (CICB in the example). It therefore calls the function-shipping transformer program to transform the request into a form suitable for transmission (in the example, line 2 indicates this). The EXEC interface program then calls on the intercommunication component to send the transformed request to the appropriate connected system (line 3). 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 causes 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 the mirror transaction terminates, and reinvokes it as required.

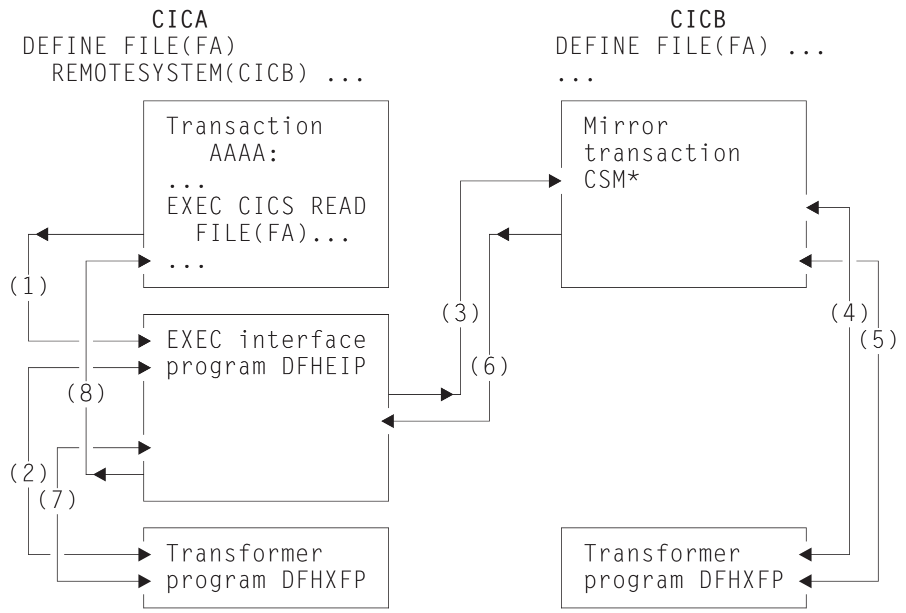


Figure 4. The transformer program and the mirror in function shipping

The mirror transaction uses the function-shipping transformer program, DFHXFP, to decode the formatted request (line 4 in Figure 4). The mirror then executes the corresponding command. On completion of the command, the mirror transaction uses the transformer program to construct a formatted reply (line 5). The mirror transaction returns this formatted reply to the requesting system, CICA (line 6). On CICA the reply is decoded, again using the transformer program (line 7), and used to complete the original request made by the application program (line 8).

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 terminates after sending its reply. However, if the request causes the mirror transaction to change or update a protected resource, or if the request is for any DL/I program specification block (PSB), it does not terminate until the requesting application program issues a synchronization point (syncpoint) request or terminates successfully. If a browse is in progress, the mirror transaction does not terminate until the browse is complete.

When the application program issues a syncpoint request, or terminates successfully, the intercommunication component sends a message to the mirror transaction that causes it also to issue a syncpoint request and terminate. The successful syncpoint by the mirror transaction is indicated in a response sent back to the requesting system, which then completes its syncpoint processing, so committing changes to any protected resources. If DL/I requests have been received from another system, CICS issues a DL/I TERM request as a part of the processing resulting from a syncpoint request made by the application program and executed by the mirror transaction.

The application program may access protected or unprotected resources in any order, and is not 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 for the application program. Each mirror transaction follows the above rules for termination, and when the application program reaches a syncpoint, the intercommunication component exchanges syncpoint messages with any mirror transactions that have not yet terminated. This is called the **multiple-mirror** situation.

The mirror transaction uses the CICS command-level interface to execute CICS requests, and the DL/I CALL or the EXEC DLI 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 is called a **chained-mirror** situation. To guard against possible threats to data integrity caused by session failures, it is strongly recommended that the system designer avoids defining a connected system in which chained mirror requests 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.

Long-running mirror tasks

Normally, MRO mirror tasks are terminated as soon as possible, in the same way as described for ISC mirrors (see page 26). This is 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 syncpoint, even though this is not required for data integrity. For example, a transaction that issues many 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, you can reduce the overheads of allocating sessions on the sending side and attaching mirror tasks on the receiving side.

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 runs*, by coding MROLRM=YES in the system initialization parameters. A long-running mirror is terminated by the next syncpoint (or RETURN) on the sending side.

For some applications, the performance benefits of using long-running mirrors can be significant.

Figures 6 and 7 in "Function shipping–examples" on page 28 show how the mirror acts for MROLRM=NO and MROLRM=YES respectively.

An additional system initialization parameter, MROFSE=YES, specified on the front-end region, extends the retention of the mirror task and the session from the next syncpoint to the end of the task. To achieve maximum benefit, MROFSE=YES should be used in conjunction with MROLRM=YES on the back-end region. However, MROFSE=YES applies even if the back-end region has MROLRM=NO, if requests are of the type which cause the mirror transaction to keep its inbound session.

Conceptually, MROLRM is specified on the back-end region and MROFSE is specified on the front-end region. However, if the distinction between “back end” and “front end” is not clear, it is safe to code both parameters on each region if necessary.

MROFSE=YES gives a performance improvement only if most applications initiated from the front-end region have multiple syncpoints and function shipping requests are issued between each syncpoint. For further information about the performance implications of using MROFSE=YES, see the *CICS Performance Guide*.

The short-path transformer

CICS uses a special transformer program (DFHXFX) for function shipping over MRO links. This **short-path transformer** 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 program (DFHXFX) for shipping file control, transient data, temporary storage, and interval control (asynchronous processing) requests. It 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 lifetime 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 unprotected resources on behalf of the application program, over MRO or ISC links, with and without MRO long-running mirror tasks.

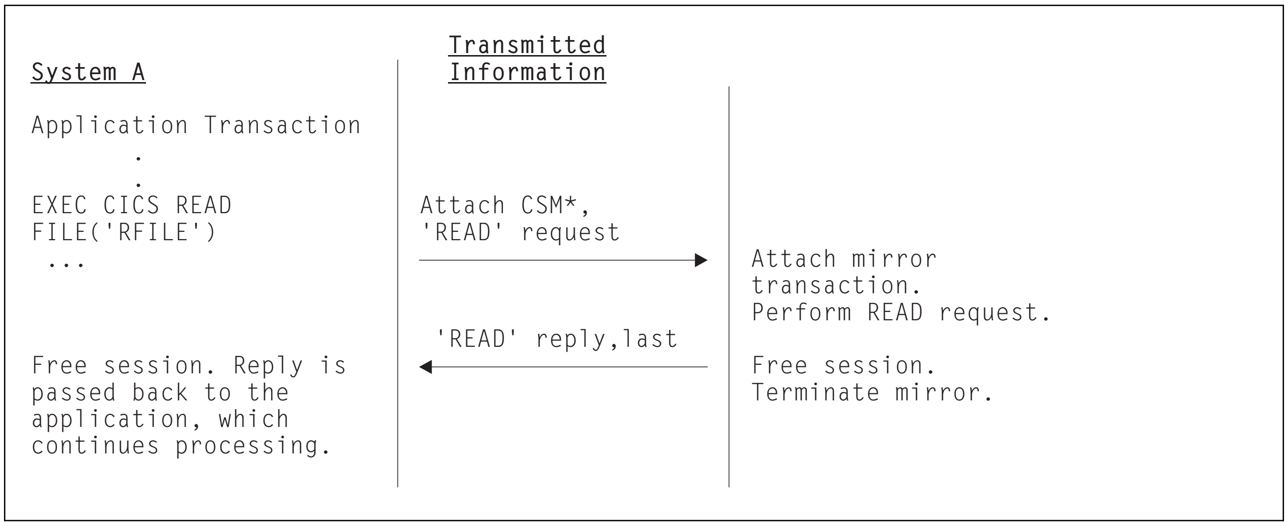
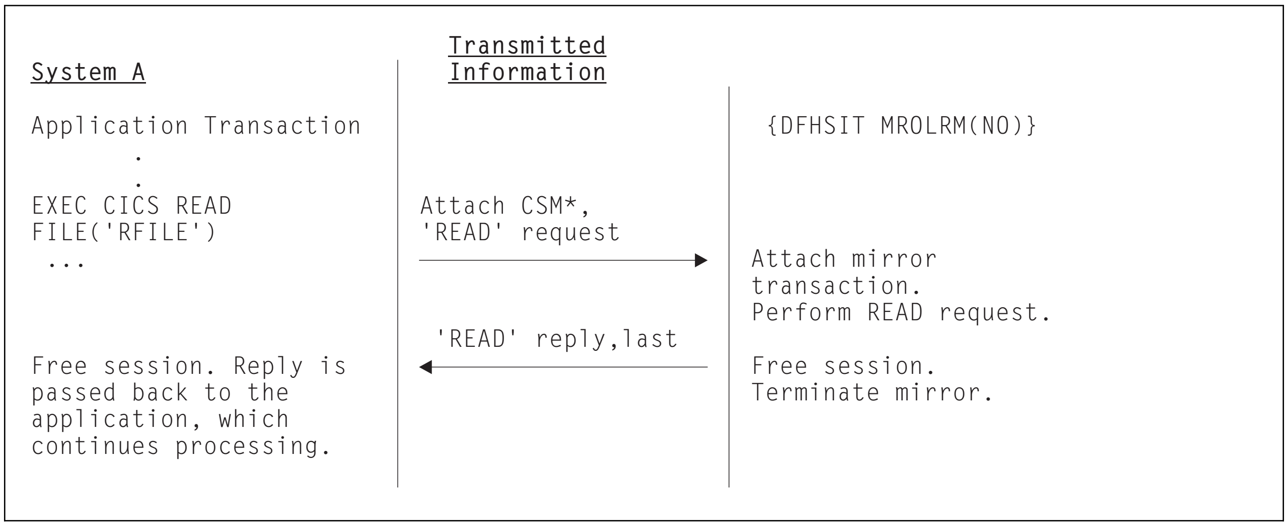

System A
Transmitted Information
Application Transaction
EXEC CICS READ FILE('RFILE') ...
Attach CSM*, 'READ' request
Attach mirror transaction. Perform READ request.
'READ' reply,last
Free session. Reply is passed back to the application, which continues processing.
Free session. Terminate mirror.

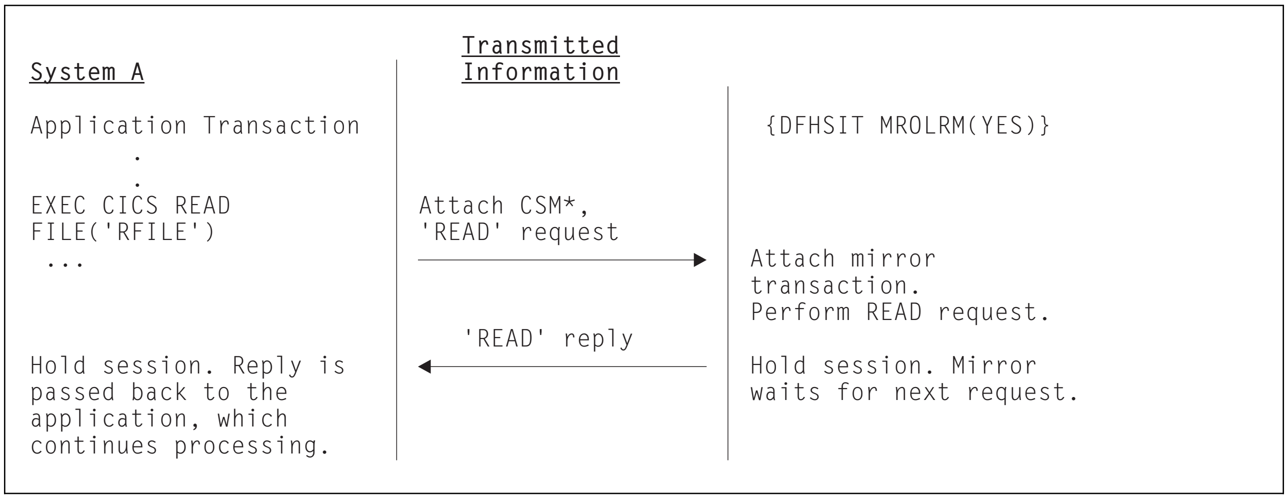
Figure 5. ISC function shipping—simple inquiry. Here no resource is being changed; the session is freed and the mirror task is terminated immediately.



```
                                Transmitted
System A                        Information

Application Transaction                                {DFHSIT MROLRM(NO)}
       .
       .
EXEC CICS READ                Attach CSM*,
FILE('RFILE')                 'READ' request
 ...                          ------------------->  Attach mirror
                                                    transaction.
                                                    Perform READ request.
                               'READ' reply,last
Free session. Reply is        <-------------------  Free session.
passed back to the                                  Terminate mirror.
application, which
continues processing.
```

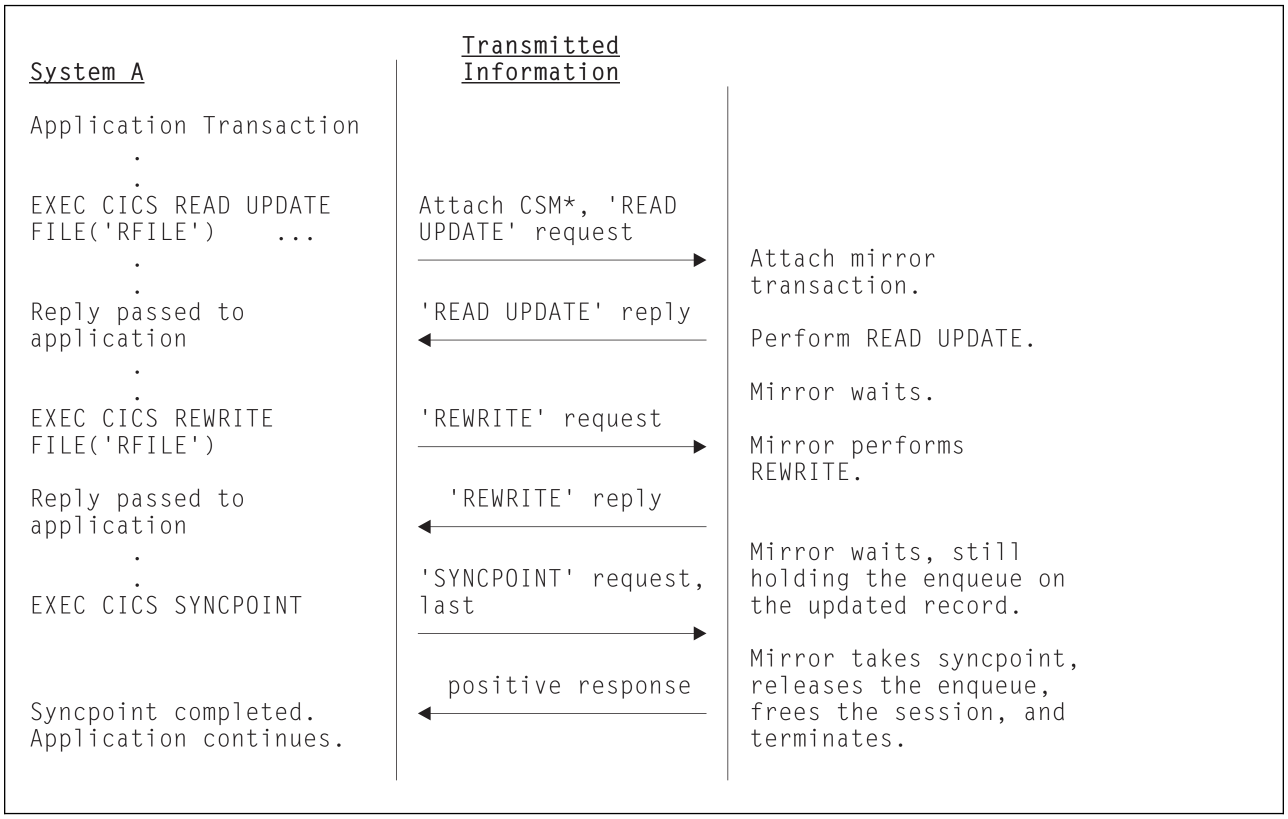
Figure 6. MRO function shipping—simple inquiry. Here no resource is being changed. Because long-running mirror tasks are not specified, the session is freed by System B and the mirror task is therefore terminated immediately.



```
                                 Transmitted
System A                         Information

Application Transaction                                {DFHSIT MROLRM(YES)}
       .
       .
EXEC CICS READ                Attach CSM*,
FILE('RFILE')                 'READ' request
 ...                          ------------------->  Attach mirror
                                                    transaction.
                                                    Perform READ request.
                                 'READ' reply
Hold session. Reply is        <-------------------  Hold session. Mirror
passed back to the                                  waits for next request.
application, which
continues processing.
```

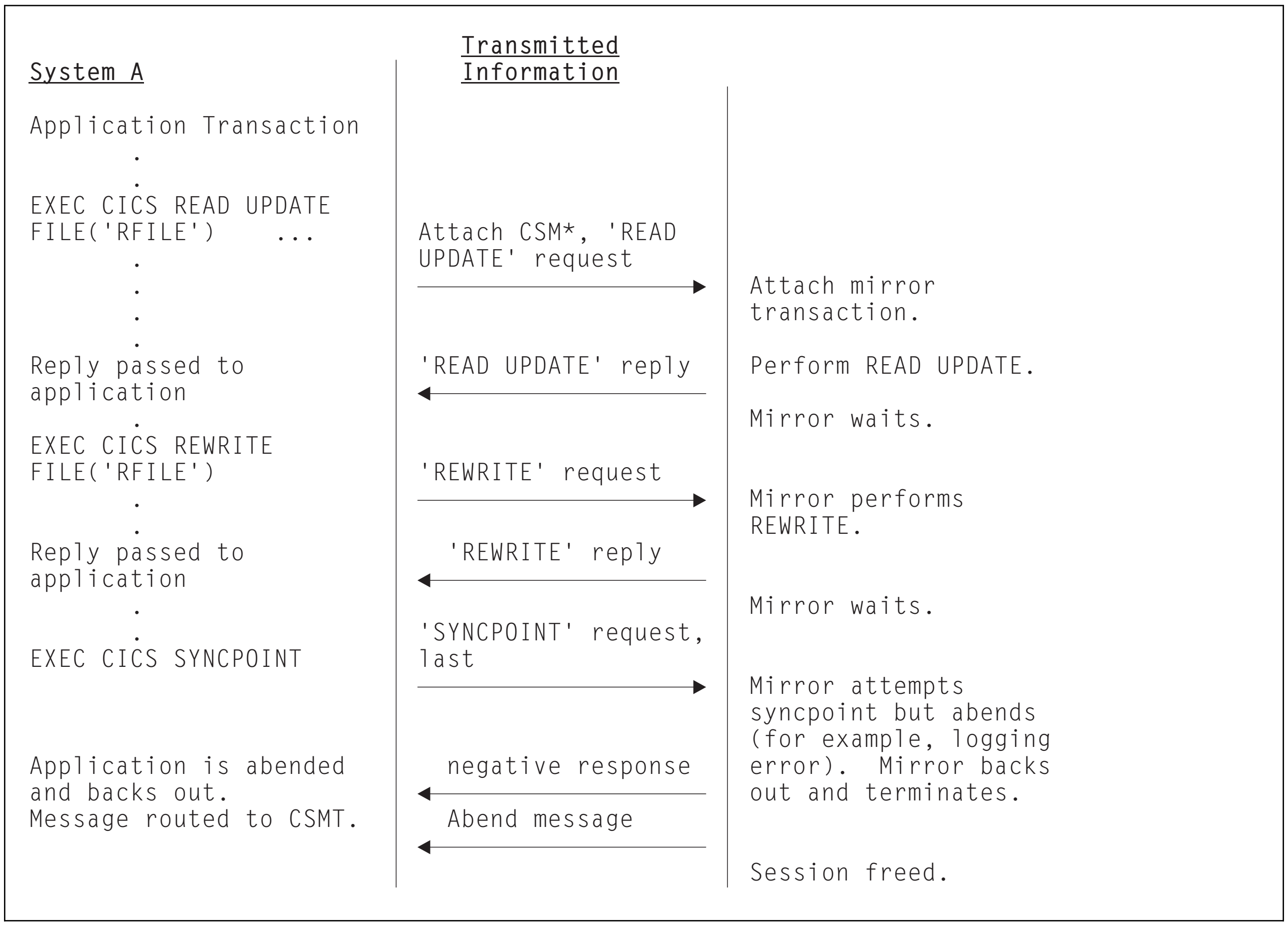
Figure 7. MRO function shipping—simple inquiry. Here no resource is being changed. However, because long-running mirror tasks are specified, the session is held by System B, and the mirror task waits for the next request.



```
System A                     Transmitted
                             Information

Application Transaction
       .
       .
EXEC CICS READ UPDATE        Attach CSM*, 'READ
FILE('RFILE')    ...         UPDATE' request
       .                     ------------------->  Attach mirror
       .                                           transaction.
Reply passed to              'READ UPDATE' reply
application                  <-------------------  Perform READ UPDATE.
       .
       .                                           Mirror waits.
EXEC CICS REWRITE            'REWRITE' request
FILE('RFILE')                ------------------->  Mirror performs
                                                   REWRITE.
Reply passed to                'REWRITE' reply
application                  <-------------------
       .                                           Mirror waits, still
       .                     'SYNCPOINT' request,  holding the enqueue on
EXEC CICS SYNCPOINT          last                  the updated record.
                             ------------------->
                                                   Mirror takes syncpoint,
                               positive response   releases the enqueue,
Syncpoint completed.         <-------------------  frees the session, and
Application continues.                             terminates.
```

Figure 8. ISC or MRO function shipping—update. Because the mirror must wait for the REWRITE, it becomes long-running and is not terminated 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.



```
                              Transmitted
System A                      Information

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

Figure 9. ISC or MRO function shipping—update with ABEND. This is similar to the previous example, except that an abend occurs during syncpoint processing.

Chapter 5. CICS distributed program link

This chapter contains the following topics:

- "Overview of DPL"
- "Design considerations" on page 34
- "Examples of DPL" on page 37

Overview of DPL

CICS distributed program link (DPL) enables CICS application programs to run programs residing in other CICS regions by shipping program-control LINK requests.

An application can be written without regard for the location of the requested programs; it simply uses program-control LINK commands in the usual way. Entries in the CICS program definition tables allow the system programmer to specify that the named program is not in the local CICS region (known as the **client region**) but in a remote CICS region (known as the **server region**).

An illustration of a DPL request is given in Figure 10. In this figure, a program (known as a **client program**) running in CICA issues a program-control LINK command for a program called PGA (the **server program**). From the installed program definitions, CICS discovers that this program is owned by a remote CICS system called CICB. CICS changes the LINK request into a suitable transmission format, and then ships it to CICB for execution.

In CICB, the mirror transaction (described in Chapter 4, "CICS function shipping" on page 21) is attached. The mirror program recreates the original request, issues it on CICB, and, when the server program has run to completion, returns any communication-area data to CICA.

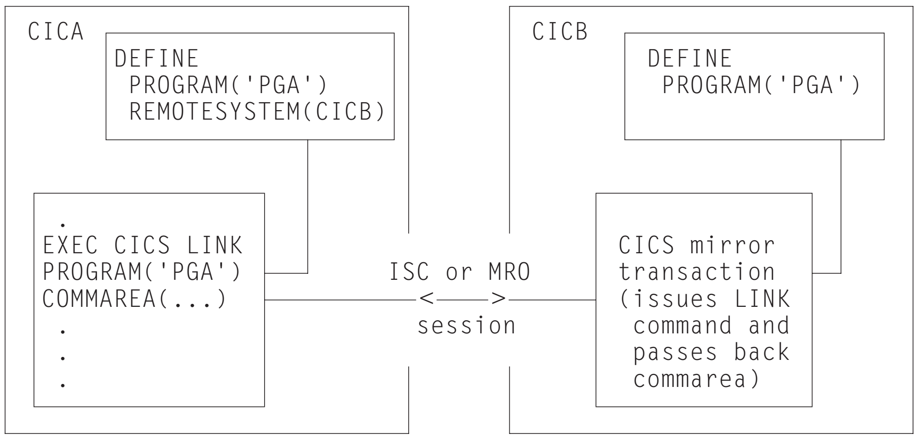

CICA
DEFINE PROGRAM('PGA') REMOTESYSTEM(CICB)
. EXEC CICS LINK PROGRAM('PGA') COMMAREA(...) . . .
ISC or MRO <——> session
CICB
DEFINE PROGRAM('PGA')
CICS mirror transaction (issues LINK command and passes back commarea)

Figure 10. Distributed program link

The CICS recovery and restart facilities enable resources in remote CICS regions to be updated, and ensure that when the client program reaches a syncpoint, any mirror transactions that are updating protected resources also take a syncpoint, so that changes to protected resources in remote and local systems are consistent.

The CSMT transient data queue is notified of any failures in this process, so that suitable corrective action can be taken, whether manually or by user-written code.

Design considerations

Client programs can run in a CICS intercommunication environment and use DPL without being aware of the location of the server program. The location of the server program is specified in the installed program resource definition. (Details are given in “CICS distributed program link (DPL)” on page 142.)

The program resource definition can also specify the name of the server program as it is known on the resource system, if it is different from the name by which it is known locally. When the server program is requested by its local name, CICS substitutes the remote name before sending the request. This facility is particularly useful when a server program exists with the same name on more than one system, but performs different functions depending on the system on which it is located. Consider, for example, a local system CICA and two remote systems CICB and CICC. A program named PG1 resides in both CICB and CICC. These two programs are to be defined in CICA, but they have the same name. Two definitions are needed, so a local alias and a REMOTENAME have to be defined for at least one of the programs. The definitions in CICA could look like this:

```
DEFINE PROGRAM(PG1) REMOTESYSTEM(CICB) ...
DEFINE PROGRAM(PG99) REMOTENAME(PG1) REMOTESYSTEM(CICC) ...
```

Although doing so may limit the client program’s independence, the client program can also name the remote system explicitly by using the SYSID option on the LINK command. If this option is specified, CICS routes the request directly to the named system without reference to the installed program resource definitions in the client region. The local system can also be specified on the SYSID option, so that the decision whether to link to a remote server program or a local one can be taken at execution time.

In the client region (CICA in Figure 11 on page 35), the command-level EXEC interface program determines that the requested server program is on another system (CICB in the example). It therefore calls the transformer program to transform the request into a form suitable for transmission (in the example, line (2) indicates this). As indicated by line (3) in the example, the EXEC interface program then calls on the intercommunication component to send the transformed request to the appropriate connected system.

Using the mirror transaction

The intercommunication component uses CICS terminal-control facilities to send the request to the mirror transaction. The request to a particular server region causes the communication component in the client region to precede the formatted request with the identifier of the appropriate mirror transaction to be attached in the server system.

Controlling access to resources, accounting for system usage, performance tuning, and establishing an audit trail can all be made easier if you use a user-specified name for the mirror transaction initiated by any given DPL request. This transaction name must be defined in the server region as a transaction that invokes the mirror program DFHMIRS. It is worth noting that defining user transactions to invoke the mirror program gives you the freedom to specify appropriate values for all the other options on the transaction resource definition. To initiate any user-defined mirror transaction, the client program specifies the transaction name on the LINK request. Alternatively, the transaction name can be specified on the TRANSID option of the program resource definition.

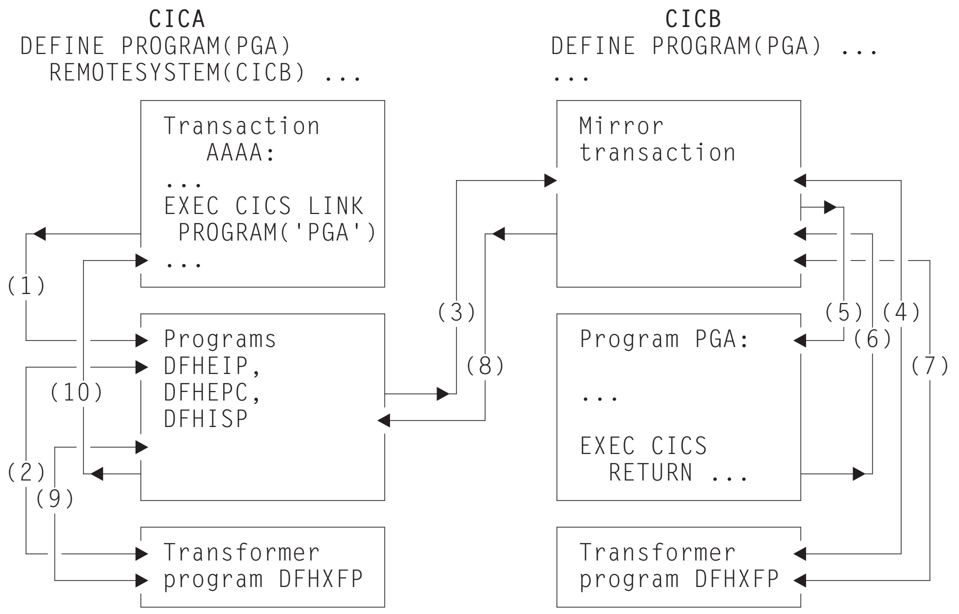


Figure 11. The transformer program and the mirror in DPL

As line (4) in Figure 11 shows, a mirror transaction uses the transformer program DFHXFP to decode the formatted link request. The mirror then executes the corresponding command, thereby linking to the server program PGA (5). When the server program issues the RETURN command (6), the mirror transaction uses the transformer program to construct a formatted reply (7). The mirror transaction returns this formatted reply to the client region (8). In that region (CICA in the example), the reply is decoded, again using the transformer program (9), and used to complete the original request made by the client program (10).

The mirror transaction, which is always long-running for DPL, suspends after sending its commarea. The mirror transaction does not terminate until the client program issues a syncpoint request or terminates successfully.

When the client program issues a syncpoint request, or terminates successfully, the intercommunication component sends a message to the mirror transaction that causes it also to issue a syncpoint request and terminate. The successful syncpoint by the mirror transaction is indicated in a response sent back to the client region, which then completes its syncpoint processing, so committing changes to any protected resources.

The client program may link to server programs in any order, without being affected by the location of server programs (they could all be in different server regions, for example). When the client program links to server programs in more than one server region, the intercommunication component invokes a mirror transaction in each server region to execute link requests for the client program. Each mirror transaction follows the above rules for termination, and when the application program reaches a syncpoint, the intercommunication component exchanges syncpoint messages with any mirror transactions that have not yet terminated.

Limitations on the server program

A server program cannot issue the following kinds of commands:

- Terminal-control commands referring to its principal facility
- Commands that set or inquire on terminal attributes
- BMS commands
- Signon and signoff commands
- Batch data interchange commands
- Commands addressing the TCTUA
- Syncpoint commands (except when the client program specifies the SYNCONRETURN option on the LINK request)

If the client specifies SYNCONRETURN:

- The server program can issue syncpoint requests.
- The mirror transaction requests a syncpoint when the server program completes processing.

Warning: Both these kinds of syncpoint commit only the work done by the server program. In applications where both the client program and the server program update recoverable resources, they could cause data-integrity problems if the client program fails after issuing the LINK request.

For further information about application programming for DPL, see Chapter 18, “Application programming for CICS DPL” on page 181.

Using global user exits to redirect DPL requests

Two global user exits can be invoked during DPL processing:

- If it is enabled, XPCREQ is invoked on entry to the CICS program control program, **before** a link request is processed. For DPL requests, it is invoked on both sides of the link; that is, in both the client and server regions.
- If it is enabled, XPCREQC is invoked **after** a link request has completed. For DPL requests, it is invoked in the client region only.

XPCREQ and XPCREQC can be used for a variety of purposes. You could, for example, use them to route DPL requests to different CICS regions, thereby providing a simple load balancing mechanism. For advice on how to do this, together with programming information about writing XPCREQ and XPCREQC global user exit programs, see the *CICS Customization Guide*.

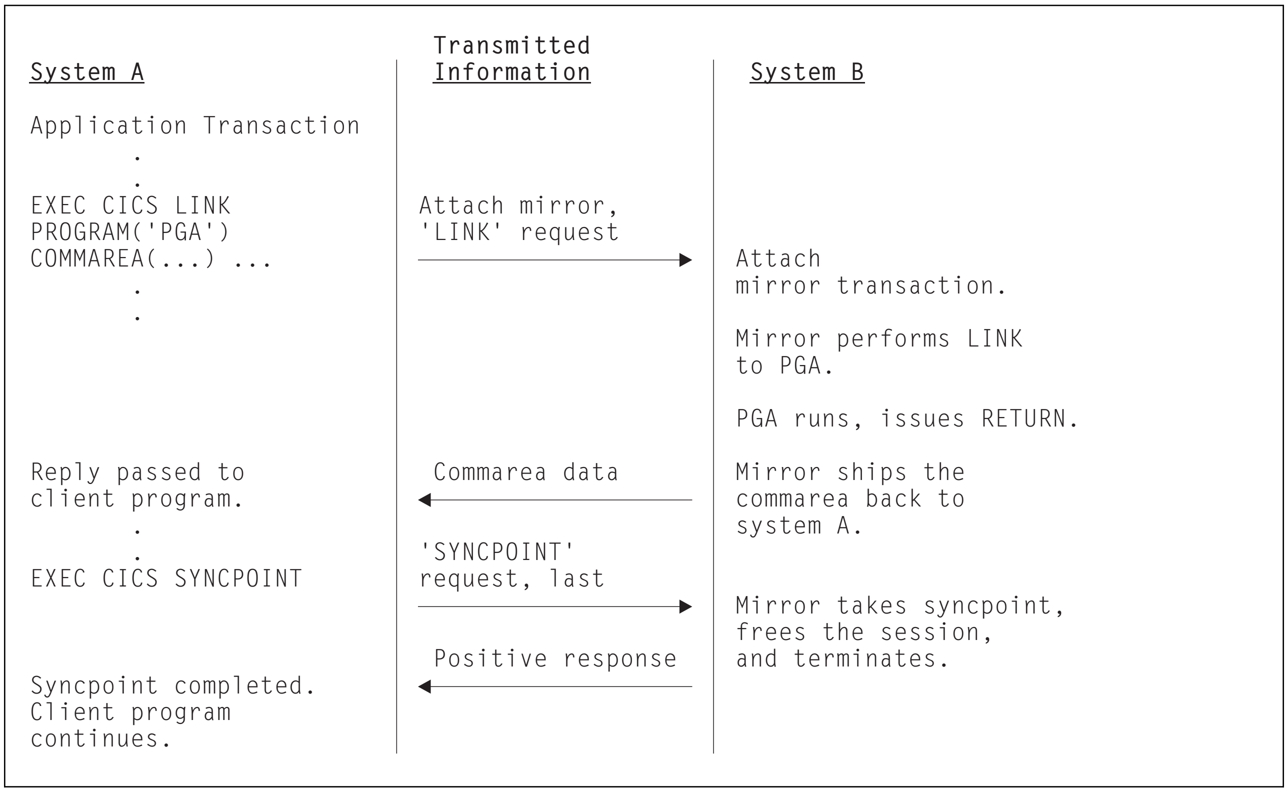
Intersystem queuing

If the link to a remote region is established, but there are no free sessions available, distributed program link requests may be queued in the issuing region. Performance problems can occur if the queue becomes excessively long.

For guidance information about controlling intersystem queues, see Chapter 23, “Intersystem session queue management” on page 221.

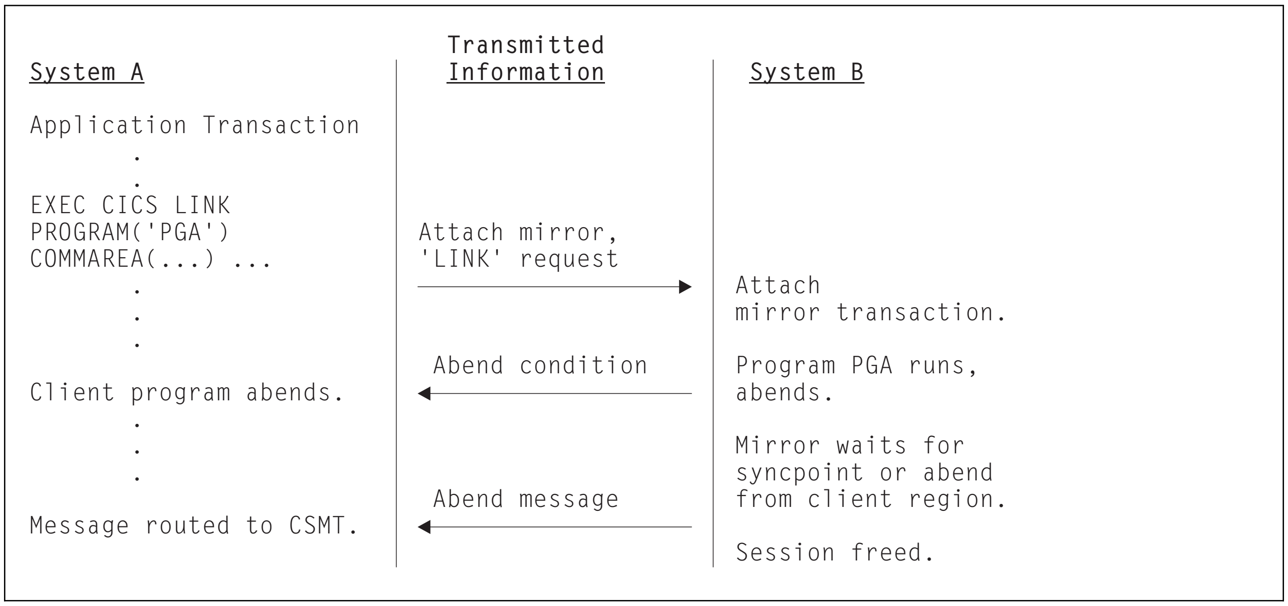
Examples of DPL

This section gives some examples to illustrate the lifetime of the mirror transaction and the information flowing between the client program and its mirror transaction.



System A
Transmitted Information
System B
Application Transaction
EXEC CICS LINK PROGRAM('PGA') COMMAREA(...) ...
Attach mirror, 'LINK' request
Attach mirror transaction.
Mirror performs LINK to PGA.
PGA runs, issues RETURN.
Reply passed to client program.
Commarea data
Mirror ships the commarea back to system A.
EXEC CICS SYNCPOINT
'SYNCPOINT' request, last
Mirror takes syncpoint, frees the session, and terminates.
Positive response
Syncpoint completed. Client program continues.

Figure 12. DPL with the client transaction issuing a syncpoint. Because the mirror is always long-running, it does not terminate before SYNCPOINT is received.



System A
Transmitted Information
System B
Application Transaction
EXEC CICS LINK PROGRAM('PGA') COMMAREA(...) ...
Attach mirror, 'LINK' request
Attach mirror transaction.
Abend condition
Program PGA runs, abends.
Client program abends.
Mirror waits for syncpoint or abend from client region.
Abend message
Message routed to CSMT.
Session freed.

Figure 13. DPL with the server program abending

Chapter 6. The external CICS interface

The external CICS interface (EXCI) is an application programming interface that enables a non-CICS client program running in a VSE/ESA address space—for example, a VSE/ESA batch program—to call a server program running in a CICS Transaction Server for VSE/ESA Release 1 system and to pass and receive data using a communications area. The CICS program is invoked as if linked-to by another CICS program.

This API allows a client program to allocate and open sessions (or *pipes*) to a CICS Transaction Server for VSE/ESA Release 1 system and to pass distributed program link (DPL) requests over them. IRC supports these requests and each pipe[6] maps onto one MRO session.

The client program and the server CICS system to which it passes the requests must be in the same VSE/ESA operating system. Once received by CICS, requests can be “daisy-chained” to other CICS systems, just like other DPL requests.

A client program that uses the external CICS interface can operate multiple sessions for different users.[7] All the sessions coexist in the same VSE/ESA address space without knowledge of, or interference from, each other.

Benefits of the external CICS interface

The external CICS interface makes CICS applications more easily accessible from non-CICS environments. VSE/ESA batch programs could, for example, be used to:

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

Implementing the external CICS interface

To use the external CICS interface, you must:

- Define the connections that your non-CICS programs will use to communicate with CICS. This is described on page 99.
- In your VSE/ESA client programs, use one of the external CICS interface APIs (two are provided) to allocate and open sessions to a CICS system, and to issue DPL requests on those sessions. The external CICS interface APIs are described in Chapter 19, “Application programming for the external CICS interface” on page 185.

6 Pipe. A one-way communication path between a sending process and a receiving process. In EXCI, each pipe maps on to one MRO session, where the client program represents the sending process and the CICS server region represents the receiving process.

7 The distinction between a VSE/ESA client program and a “user” is explained on page 185.

For programming information about using the external CICS interface, see the *CICS External Interfaces Guide* manual.

Chapter 7. Asynchronous processing

This chapter contains the following topics:

- "Overview of asynchronous processing"
- "Asynchronous processing methods" on page 42
- "Asynchronous processing using START and RETRIEVE commands" on page 43
- "System programming considerations" on page 48
- "Asynchronous processing—examples" on page 49

Overview of 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 14.

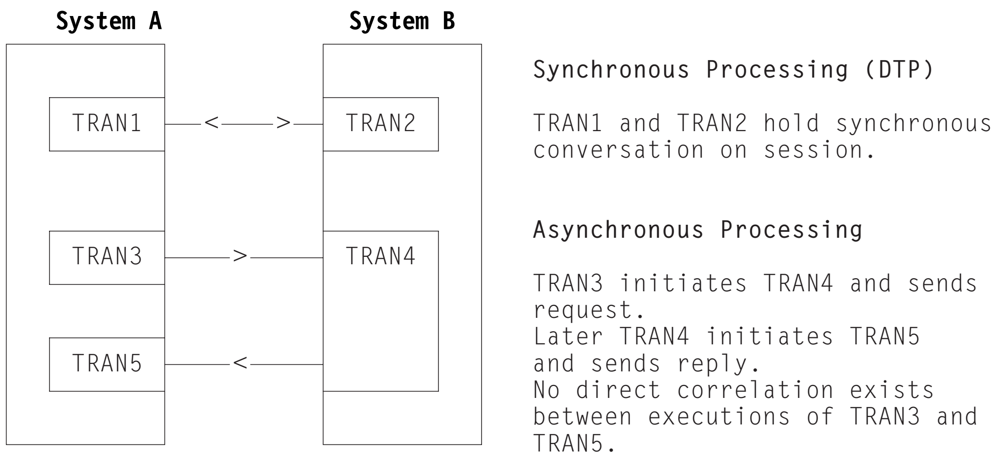

System A
System B
TRAN1
TRAN2
TRAN3
TRAN4
TRAN5
Synchronous Processing (DTP)
TRAN1 and TRAN2 hold synchronous conversation on session.
Asynchronous Processing
TRAN3 initiates TRAN4 and sends request.
Later TRAN4 initiates TRAN5 and sends reply.
No direct correlation exists between executions of TRAN3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

Figure 14. Synchronous and asynchronous processing compared

A typical application area for asynchronous processing is online inquiry on remote databases; for example, an application to check a credit rating. 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 one that initiated the transaction). The replies may not arrive in the same order as that in which 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 process 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 EXEC CICS START and EXEC CICS RETRIEVE.

   You can use the EXEC CICS 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 in effect 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 EXEC CICS 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 EXEC CICS RETRIEVE command to retrieve any data associated with the request. Data transfer is restricted to a single record passing from the initiating transaction to the transaction initiated.

2. By using distributed transaction processing (DTP).

   This is 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 EXEC CICS SEND commands to pass multirecord files.

   When you have exchanged data, you 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. APPC 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, it is full DTP.

In both forms of asynchronous processing (and also in synchronous processing), a CICS transaction can use the EXEC CICS ASSIGN STARTCODE command 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 EXEC CICS SEND LAST command answered by a single EXEC CICS RECEIVE, this book refers to it elsewhere as the SEND/RECEIVE interface. The circumstances under which it is used are described in Chapter 22, “CICS-to-IMS applications” on page 197.

The remainder of this chapter is devoted to asynchronous processing using EXEC CICS START and RETRIEVE commands. Distributed transaction processing is described in Chapter 9, “Distributed transaction processing” on page 71.

Asynchronous processing using START and RETRIEVE commands

For programming information about CICS interval control, see the *CICS Application Programming Reference* manual. The interval control commands that can be used for asynchronous processing are:

- EXEC CICS START
- EXEC CICS CANCEL
- EXEC CICS RETRIEVE

Starting and canceling remote transactions

The interval control EXEC CICS START command is used to schedule transactions asynchronously in remote CICS and IMS systems. The command is function shipped. If the remote system is CICS, the mirror transaction is invoked in the remote system to issue the EXEC CICS START command on that system.

For CICS-to-CICS communication, you can include time-control information on the shipped EXEC CICS START command in the normal way, by means of the INTERVAL or TIME options. A TIME specification is converted by CICS to a time interval, relative to the local clock, before the command is shipped. Because the ends of an intersystem link may be in different time zones, 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 an EXEC CICS 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.

An EXEC CICS START command shipped to a remote CICS system can be canceled at any time up to its expiration time by shipping an EXEC CICS CANCEL command to the same system. The particular START command has a unique 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 EXEC CICS START commands sent to IMS systems; INTERVAL(0) must be specified or allowed to take the default value.

Consequently, start requests for IMS transactions cannot be canceled after they have been issued.

Passing information with the EXEC CICS START command

The EXEC CICS 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, it obtains the information by issuing an EXEC CICS 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.

- The transaction and terminal names to be used for replies—specified in the RTRANSID and RTERMID options.

  These options, whose values are set by the local transaction, provide the means for the remote transaction to pass a reply to the local system. (That is, the TRANSID and TERMID specified by the remote transaction on its reply are the RTRANID and RTERMID specified by the local system on the initial request.)

- 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. It may be that the terminal is defined on the region that owns the remote transaction but is not owned by that region. If so, it is obtained by the automatic transaction initiation (ATI) facility of transaction routing. See “Automatic transaction initiation (ATI)” on page 57.

  The global user exits XICTENF and XALTENF can be coded to cover the case of the terminal that is *shippable* but not defined in the application-owning region—see “Shipping terminals for automatic transaction initiation” on page 58.

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

Passing a sysid or applid with the START command

If you have a transaction that can be started from several different systems, and which is required to issue an EXEC CICS START command to the system that initiated it, you can arrange for all of the invoking transactions to send their local system sysid or applid as part of the user data in the EXEC CICS START command. An initiating transaction can obtain its local sysid by using an EXEC CICS ASSIGN SYSID command, or its applid by using an EXEC CICS ASSIGN APPLID command.

If the name of the connection to the remote system matches the SYSIDNT system initialization parameter of the remote system (typical of MRO), then the started transaction can reply using an EXEC CICS START command specifying the passed sysid.

If the name of an APPC or LUTYPE6.1 connection to the remote system does not match the SYSIDNT system initialization parameter of the remote, then the started transaction can still determine the sysid to be responded to. It can do this by issuing an EXEC CICS EXTRACT TCT command on which the NETNAME option specifies the passed applid.

Improving performance of intersystem START requests

In many inquiry-only applications, sophisticated error-checking and recovery procedures are not justified. Where the transactions make inquiries only, the terminal operator can retry an operation if no reply is received within a specific time. In such a situation, the number of messages to and from the remote system can be substantially reduced by using the NOCHECK option of the EXEC CICS 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 some types of error in the START command.

A typical use for the START NOCHECK command is in the remote inquiry application described at the beginning of this chapter.

The transaction attached as a result of the terminal operator's inquiry issues an appropriate EXEC CICS 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 inquiry. 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 inquiry 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 the inquiry, and be prepared to receive duplicate replies. To aid the operator, 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 in Figure 16 on page 51.

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 of EXEC CICS START commands" on page 47), 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 EXEC CICS START command. The PROTECT option indicates that the remote transaction must not be scheduled until the local one has successfully completed a synchronization point (syncpoint). (It can take the syncpoint either by issuing an EXEC CICS 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 syncpoint. Both PROTECT and NOCHECK must be specified for all IMS recoverable transactions.

Deferred sending of START requests with NOCHECK option

For EXEC CICS START commands with the NOCHECK option, whether or not PROTECT is specified, CICS may defer transmission of the request to the remote system, depending on the environment.

For MRO links, START requests with NOCHECK are not deferred.

For ISC links, START requests with NOCHECK are deferred until one of the following events occurs:

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

For both the APPC and LUTYPE6.1 protocols, if the first START with NOCHECK is followed by a second, CICS transmits the first and defers the second.

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 in the logical unit of work (LUW) 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, no message must cross the link between a START request and the following syncpoint. Therefore, you cannot send multiple EXEC CICS START NOCHECK PROTECT requests to IMS. Each request must be followed by an EXEC CICS SYNCPOINT command, or by termination of the transaction.

Intersystem queuing

If the link to a remote region is established, but there are no free sessions available, function shipped EXEC CICS START requests used to schedule remote transactions may be queued in the issuing region. Performance problems can occur if the queue becomes excessively long. This problem is described on page 24.

For guidance information about controlling intersystem queues, see Chapter 23, "Intersystem session queue management" on page 221.

Local queuing of EXEC CICS START commands

If a remote system is unavailable, either because it is not active or because a connection cannot be established, an attempt to function ship a START request to it normally results in the SYSIDERR condition being returned to the application. This can happen too, when there *is* a connection to the remote system, but there are no sessions available and you have chosen not to queue the request in the issuing region. However, provided that the remote system is directly connected to this CICS, and that you specify the NOCHECK option on the EXEC CICS START command, you can arrange for the request to be queued locally, and forwarded when the required link is in service. You can do this in two ways:

1. Specify LOCALQ(YES) on the local definition of the remote transaction. The LOCALQ option specifies that local queuing is used, where necessary, for all requests from the local system for a particular remote transaction.

   For information about the LOCALQ option, see the *CICS Resource Definition Guide*.

2. Use an XISLCLQ global user exit program. XISLCLQ is invoked only for function shipped EXEC CICS START NOCHECK commands where:

   - The remote system is unavailable

     or

   - There is a connection to the remote system but there are no sessions available, and *either* the number of requests currently queued in the issuing region has reached the maximum specified on the QUEUELIMIT option of the CONNECTION definition *or* your XZIQUE or XISCONA global user exit program has specified that the request is not to be queued in the issuing region.

   Your user exit program can decide, on a request-by-request basis, whether to queue locally.

   For programming information about the XZIQUE, XISCONA, and XISLCLQ global user exits, see the *CICS Customization Guide*.

Data retrieval by a started transaction

A CICS transaction that is started by a start request can get the user data and other information associated with the request by using the EXEC CICS 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 is 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 using a further EXEC CICS RETRIEVE command. This has the effect of canceling the queued start request.

Thus, it is possible to design transactions that can handle the data associated with multiple start requests. Typically, a long-running local transaction could be designed to accept multiple inquiries 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 EXEC CICS 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. If this option is used in a task attached to an APPC device, CICS does not suspend the task, but instead raises the ENDDATA condition if no data is currently available. However, for tasks attached to non-APPC devices, you must make sure that your 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. It 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 EXEC CICS START command.

If CICS finds that the “terminal” named in a locally- or remotely-issued start request is a system, it selects a session available to that system and makes it the principal facility of the started transaction (see “Terminology” on page 175). If no session is available, the request is queued until there is one.

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

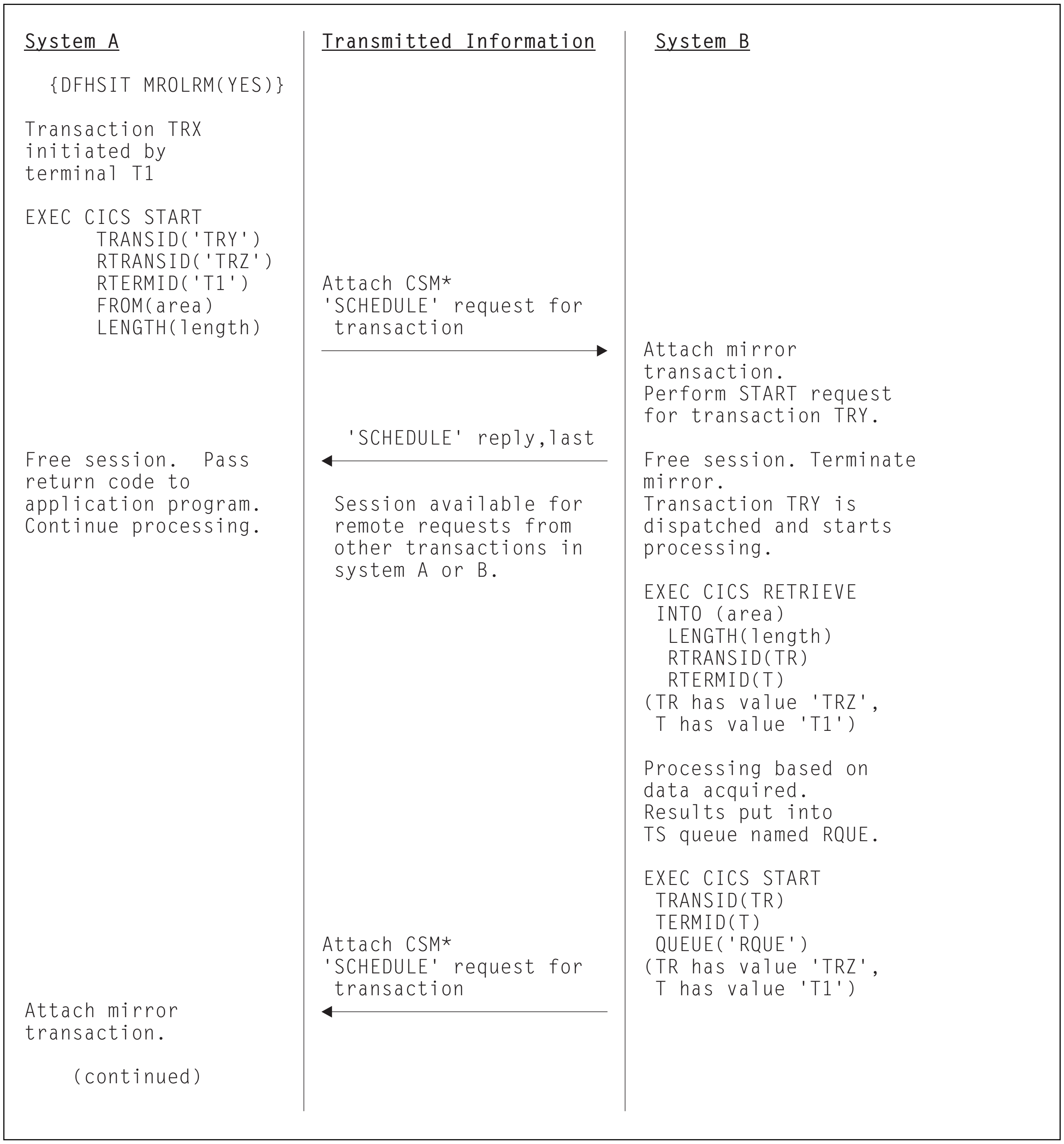
System programming considerations

This section discusses the CICS resources that must be defined for asynchronous processing. Information about how to define the resources is given in Part 3, “Resource definition” on page 91. Things to remember are:

- A link to a 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 EXEC CICS 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.

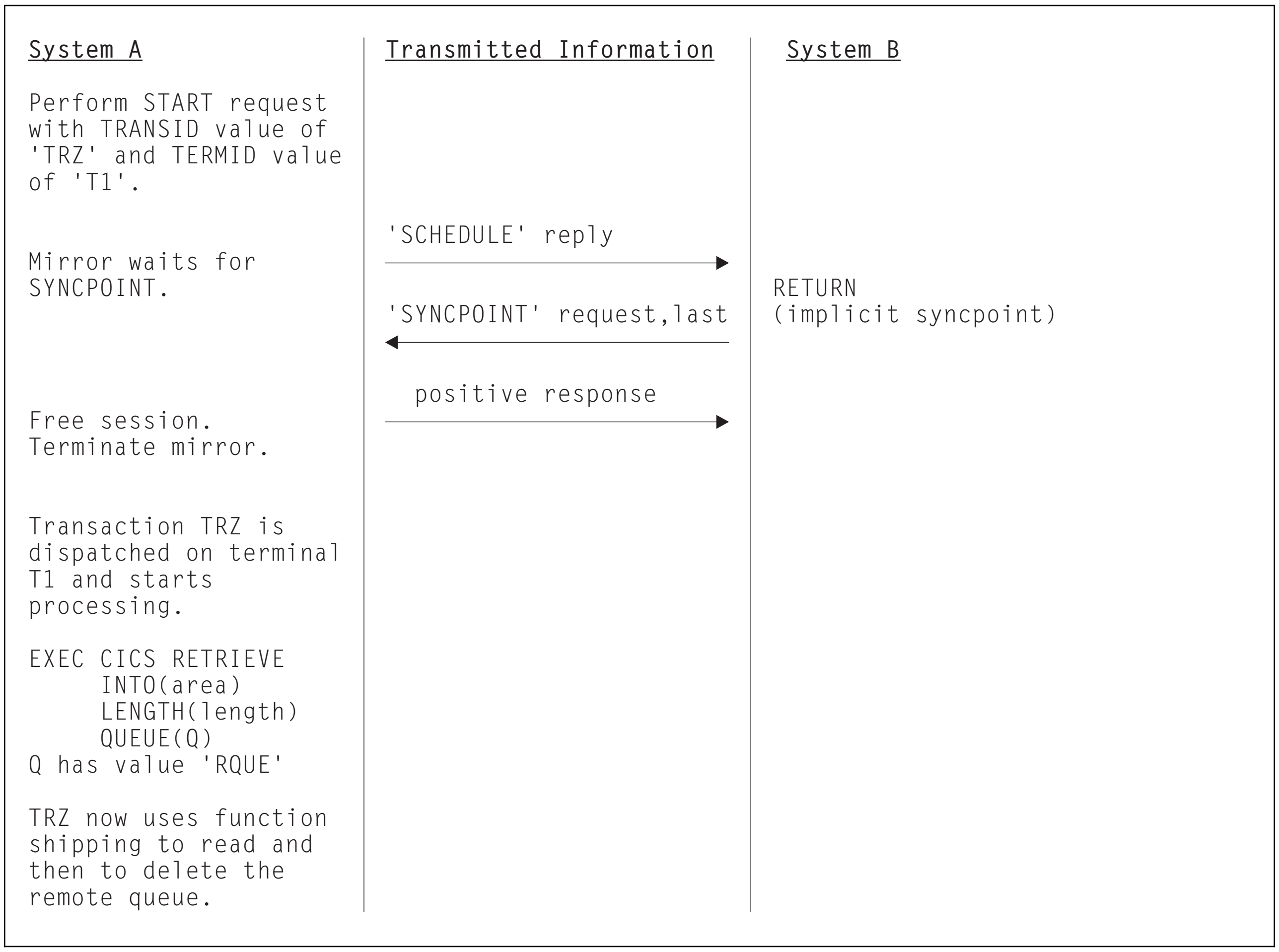
- If a START request names a “reply” transaction, that transaction must be defined on the system to which the start request is shipped.

Asynchronous processing—examples



```
System A                 | Transmitted Information    |  System B
                         |                            |
  {DFHSIT MROLRM(YES)}   |                            |
                         |                            |
Transaction TRX          |                            |
initiated by             |                            |
terminal T1              |                            |
                         |                            |
EXEC CICS START          |                            |
      TRANSID('TRY')     |                            |
      RTRANSID('TRZ')    |                            |
      RTERMID('T1')      | Attach CSM*                |
      FROM(area)         | 'SCHEDULE' request for     |
      LENGTH(length)     |  transaction               |
                         | -------------------------> | Attach mirror
                         |                            | transaction.
                         |                            | Perform START request
                         |                            | for transaction TRY.
                         |   'SCHEDULE' reply,last    |
Free session.  Pass      | <------------------------- | Free session. Terminate
return code to           |                            | mirror.
application program.     |  Session available for     | Transaction TRY is
Continue processing.     |  remote requests from      | dispatched and starts
                         |  other transactions in     | processing.
                         |  system A or B.            |
                         |                            | 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)
                         | Attach CSM*                |  QUEUE('RQUE')
                         | 'SCHEDULE' request for     | (TR has value 'TRZ',
                         |  transaction               |  T has value 'T1')
Attach mirror            | <------------------------- |
transaction.             |                            |
                         |                            |
    (continued)          |                            |
```

Figure 15 (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
                          'SYNCPOINT' request,last    (implicit syncpoint)
                          <-----------------------

                            positive response
Free session.             ----------------------->
Terminate mirror.


Transaction TRZ is
dispatched on terminal
T1 and starts
processing.

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

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

Figure 15 (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

                         Attach CSM*
Terminate, and free      'SCHEDULE' request for
terminal T1.  T1 could   trans, last (no reply)
now initiate another     ----------------------->  Attach mirror.
transaction, but TRZ                               Perform START
could not start until                              request for transaction
T1 became free again.       session available      TRY.  Free session.
                                                   Terminate mirror.
                                                   Transaction TRY is
                                                   dispatched and starts.
                                                   EXEC CICS RETRIEVE
                                                    INTO (area)
                                                     LENGTH(length)
                                                     RTRANSID(TR)
                                                     RTERMID(T)
                                                   (TR has value 'TRZ',
                                                    T has value 'T1')

                                                   Data determines
                                                   processing.  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')
   (continued)
```

Figure 16 (Part 1 of 2). Asynchronous processing—remote transaction initiation using NOCHECK

System A	Transmitted Information	System B
	Attach CSM* 'SCHEDULE' request for trans, last (no reply)	TRY terminates.
Attach mirror transaction.	←	
Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'.		
Free session.	session available	
Terminate mirror.		
Transaction TRZ is dispatched on terminal T1 and starts processing.		

Figure 16 (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 8. CICS transaction routing

This chapter contains the following topics:

Overview of transaction routing

CICS transaction routing allows terminals connected to one CICS system to run with transactions in another connected CICS system. This means that you can distribute terminals and transactions around your CICS systems and still have the ability to run any transaction with any terminal.

Figure 17 shows a terminal connected to one CICS system running with a user transaction in another CICS system. Communication between the terminal and the user transaction is handled by a CICS-supplied transaction called the **relay transaction**.

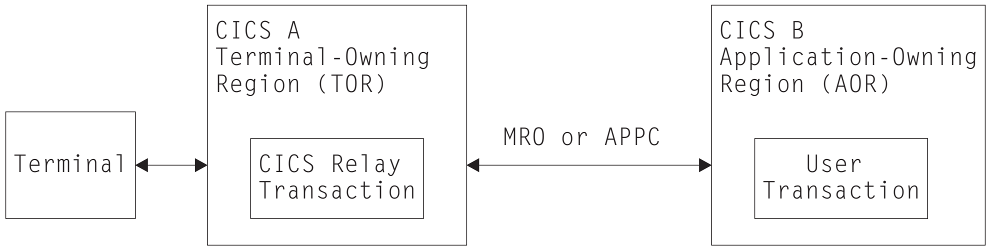


Figure 17. The elements of transaction routing

The CICS system that owns the terminal is called the **terminal-owning region** or **TOR**, and the CICS system that owns the transaction is called the **application-owning region** or **AOR**. These terms are not meant to imply that one system owns all the terminals and the other system all the transactions, although this is a possible configuration.

The terminal-owning region and the application-owning region must be connected by MRO or APPC links. Transaction routing over LUTYPE6.1 links is not supported.

In transaction routing, the term *terminal* is used in a general sense to mean such things as an IBM 3270, or a single-session APPC device, or an APPC session to another CICS system, and so on. **All** terminal and session types supported by CICS are eligible for transaction routing, **except** those given in the following list:

- LUTYPE6.1 connections and sessions
- MRO connections and sessions
- IBM 2260 terminals

- Pooled 3600 or 3650 pipeline logical units
- VSE system consoles

The user transaction can use the terminal control, BMS, or batch data interchange facilities of CICS to communicate with the terminal, as appropriate for the terminal or session type. Mapping and data interchange functions are performed in the application-owning region. BMS paging operations are performed in the terminal-owning region. (More information about BMS operations is given under "Basic mapping support (BMS)" on page 67.)

Pseudo-conversational transactions are supported (except when the "terminal" is an APPC session), and the various transactions that make up a pseudo-conversational transaction can be in different systems.

More information about writing transactions used in transaction routing is given in Chapter 21, "Application programming for CICS transaction routing" on page 193.

Initiating transaction routing

Transaction routing can be initiated in the following three ways:

1. A request to start a transaction can arrive from a terminal connected to the TOR. On the basis of an installed resource definition for the transaction, and possibly on decisions made in a user-written dynamic routing program, the request is routed to an appropriate AOR, and the transaction runs as if the terminal were attached to the same region.
2. A transaction can be started by automatic transaction initiation (ATI) and can acquire a terminal that is owned by another CICS system.
3. A transaction can issue an EXEC CICS ALLOCATE command to obtain a session to an APPC terminal or connection that is owned by another system.

In addition to these methods, CICS provides a special transaction (CRTE) that can be used for the occasional invocation of transactions in other systems. See "The routing transaction (CRTE)" on page 68.

Terminal-initiated transaction routing

When a request to start a transaction arrives at a CICS TOR, the TOR must find out on which system the transaction is to run. It does this by examining the installed transaction definition; in particular, the values of the DYNAMIC and REMOTESYSTEM options. See "Defining transactions for transaction routing" on page 155.

Terminal-initiated transaction routing can be either **static** or **dynamic**, depending upon the value of the DYNAMIC option.

Static transaction routing

Static transaction routing occurs when DYNAMIC(NO) is specified in the transaction definition. In this case, the request is routed to the system named in the REMOTESYSTEM option. (If REMOTESYSTEM is unspecified, or if it names the local CICS system, the transaction is a local transaction, and transaction routing is not involved.)

Dynamic transaction routing

Specifying DYNAMIC(YES) means that you want the chance to route the terminal data to an alternative transaction at the time the defined transaction is invoked. CICS manages this by allowing a user-replaceable program, called the **dynamic transaction routing program**, to intercept the terminal input data and specify that it be redirected to any transaction and system. The default dynamic transaction routing program, supplied with CICS, is named DFHDYP. You can modify the supplied program, or replace it with one that you write yourself. You can also use the DTRPGM system initialization parameter to specify the name of the program that is invoked for dynamic routing, if you want to name your program something other than DFHDYP. For programming information about user-replaceable programs in general, and about DFHDYP in particular, see the *CICS Customization Guide*. For information about system initialization parameters, see the *CICS System Definition Guide*.

When your routing program is invoked

CICS invokes the dynamic transaction routing program:

- When a transaction defined as DYNAMIC(YES) is initiated.
- When a transaction definition is not found, and CICS uses the common transaction definition specified on the DTRTRAN system initialization parameter. See "Using a single transaction definition in the TOR" on page 158.
- Before routing to a terminal-oriented, remote, automatically-initiated (by ATI), transaction. For example, when an ATI request on a remote system is associated with a terminal owned by this system[8]. (This case is described in "Automatic transaction initiation (ATI)" on page 57.)
- If an error occurs in route selection.
- At the end of a routed transaction, if the initial invocation requests re-invocation at termination.
- If a routed transaction abends, if the initial invocation requests re-invocation at termination.

Information passed to your routing program

Parameters are passed in a communications area between CICS and the dynamic routing program. The program may change some of these parameters to influence subsequent CICS action. The parameters include:

- The reason for the current invocation.
- Error information.
- The sysid of the target system. Initially, the one specified on the REMOTESYSTEM option of the installed transaction definition. If none was specified, the sysid passed is that of the local system.

  Note: The recommended method is to use a single, common definition for all remote transactions that are to be dynamically routed. See "Using a single transaction definition in the TOR" on page 158.

[8] In this case, your dynamic routing program cannot redirect requests, but it could, for example, update a count of requests routed to a particular system.

- The name of the target transaction. Initially, the name specified on the REMOTENAME option for the installed transaction definition. If none was specified, the name passed is the local name.
- The address of a buffer containing a copy of the data in the terminal input/output area (TIOA).
- The netname of the target system. Initially, it corresponds to the sysid specified on the REMOTESYSTEM option of the installed transaction definition.
- The address of the target transaction's communications area.
- A user area.

Using your routing program

Dynamic transaction routing enables you to make transaction routing decisions based on such factors as input to the transaction, available CICS systems, relative loading of the available systems, and so on. Note that your routing program can only reroute terminal-initiated requests, where DYNAMIC(YES) is specified on the transaction definition. It cannot reroute remote ATI requests. However, a routing program can perform other functions, besides redirecting transaction requests.

Your dynamic routing program could be used to:

- Perform work-load balancing. For example, in a CICSplex, your program could make intelligent choices between equivalent transactions on parallel AORs.
- Stipulate whether a request is to be queued if no sessions to a remote system are available. (For information about controlling the length of intersystem queues, see Chapter 23, “Intersystem session queue management” on page 221.)
- For MRO links only, set the priority of the transaction attached in the AOR.
- Cause a user-defined program to run if the transaction cannot be routed, or if the routed-to transaction abends. For example, if all remote CICS regions are unavailable and the transaction cannot be routed, you might want to run a program in the local terminal-owning region to send an appropriate message to the user.
- Monitor the number of requests routed to particular systems.

A dynamic transaction routing program can issue EXEC CICS commands, but EXEC CICS RECEIVE prevents the routed-to transaction from obtaining the initial terminal data.

For programming information about writing a dynamic transaction routing program, see the *CICS Customization Guide*.

Transaction affinities

CICS transactions use many techniques to pass information between one another, and to synchronize activity between themselves. Some of these techniques require the transactions exchanging data to execute in the same CICS region, and therefore impose restrictions on the dynamic routing of the transactions. If you are using dynamic transaction routing for workload-balancing purposes (where equivalent transactions reside on multiple systems), your routing program must be aware of transactions that contain affinities, so that it can route them consistently.

For further information about transaction affinities, see the *CICS Application Programming Guide*.

Using CICSPlex SM®

Normally, to take advantage of dynamic transaction routing, you have to write a dynamic transaction routing program. However, if you use the CICSPlex System Manager (CICSPlex SM) product to manage your CICSplex, you need not do so. CICSPlex SM provides a dynamic routing program that supports both workload balancing and workload separation. All you have to do is to tell CICSPlex SM, through its user interface, which TORs and AORs in the CICSplex can participate in dynamic transaction routing, and define any affinities that govern the AORs to which particular transactions must be routed.

For introductory information about CICSPlex SM, see the *CICSPlex SM Concepts and Planning* manual.

Automatic transaction initiation (ATI)

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. For example, in Figure 18 on page 58, an application in AOR1 issues an EXEC CICS START request for transaction TRAA to be attached to terminal PRT1.

Although the original ATI request occurs in the AOR, it is sent by CICS to the TOR for execution. So, in the example, AOR1 sends the EXEC CICS START request to TOR1 to be executed. In the TOR, the ATI request causes the relay program to be initiated, in conjunction with the specified terminal (PRT1 in the example).

The user transaction in the application-owning region is then accessed in the manner described for terminal-initiated transaction routing. There is, however, one important difference—the transaction is always routed back to the system in which the ATI request originated. Associated with the request is an automatic initiate descriptor (AID) that specifies the names of the remote transaction (TRAA) and system (AOR1). For static transaction routing, the terminal-owning region (TOR1) must find a transaction definition that specifies REMOTESYSTEM(AOR1) and REMOTENAME(TRAA); if it cannot, the request fails. For dynamic transaction routing, when DYNAMIC(YES) is coded on the transaction definition, the dynamic routing program is invoked but cannot reroute the request, because the remote system name is taken from the AID.

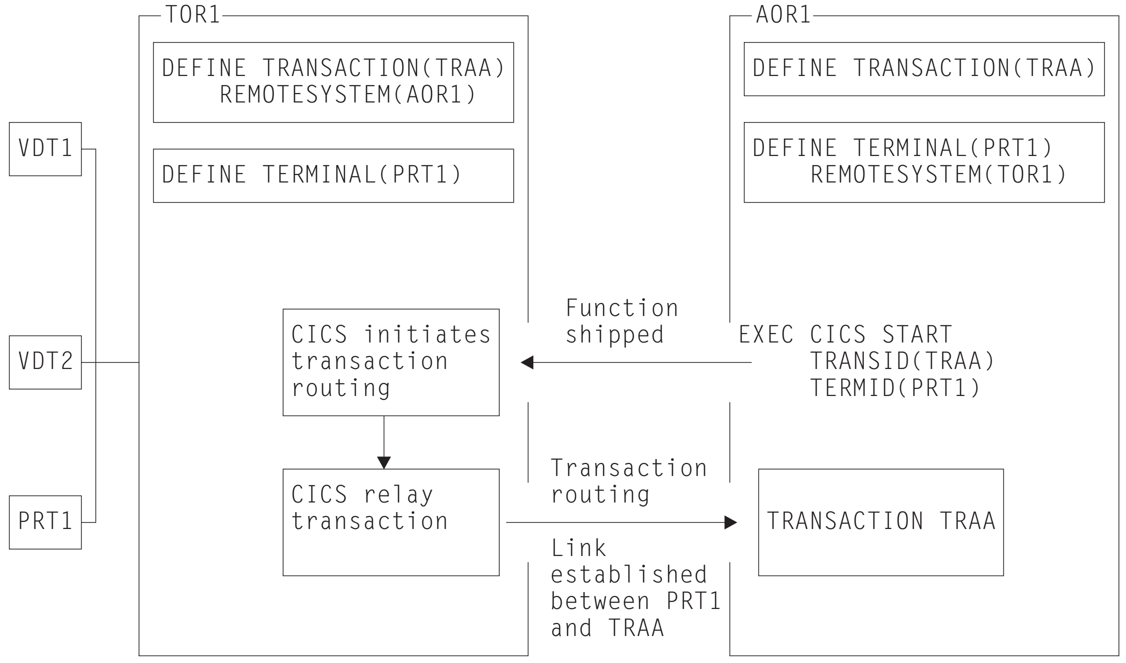

TOR1
DEFINE TRANSACTION(TRAA) REMOTESYSTEM(AOR1)
DEFINE TERMINAL(PRT1)
VDT1
VDT2
PRT1
CICS initiates transaction routing
CICS relay transaction
Function shipped
Transaction routing
Link established between PRT1 and TRAA
AOR1
DEFINE TRANSACTION(TRAA)
DEFINE TERMINAL(PRT1) REMOTESYSTEM(TOR1)
EXEC CICS START TRANSID(TRAA) TERMID(PRT1)
TRANSACTION TRAA

Figure 18. ATI-initiated 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 not available.

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.

Shipping terminals for automatic transaction initiation

A CICS system, CICA, can cause an ATI request to be executed in another CICS system, CICB, in three ways:

1. CICA function-ships an EXEC CICS 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 an EXEC CICS 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 in the REMOTESYSTEM option of the terminal definition. Remember that an ATI request is executed 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*. (For information about using autoinstall, see the *CICS Resource Definition Guide*.) 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 and connection definitions" on page 147.) This works when routing from a terminal to a remote system, but there are cases where a system cannot process an ATI request, because it has not been told the location of the associated terminal.

The example shown in Figure 19 should make this clear:

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

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

   and exits.
4. CICA function-ships the EXEC CICS START command to CICB.
5. 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.
6. Only if a definition of T1, resulting from an earlier routed transaction, is present can CICB determine where to send the ATI request. Assuming no such definition exists, the interval control program rejects the START request with TERMIDERR.

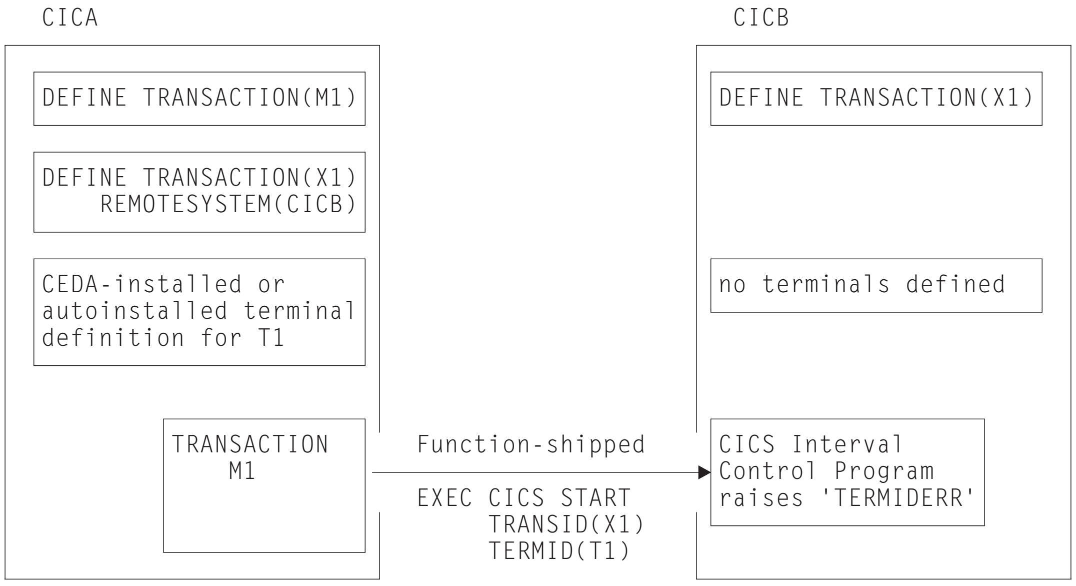

CICA
CICB
DEFINE TRANSACTION(M1)
DEFINE TRANSACTION(X1) REMOTESYSTEM(CICB)
CEDA-installed or autoinstalled terminal definition for T1
TRANSACTION M1
Function-shipped
EXEC CICS START TRANSID(X1) TERMID(T1)
DEFINE TRANSACTION(X1)
no terminals defined
CICS Interval Control Program raises 'TERMIDERR'

Figure 19. Failure of an ATI request in a system where the termid is unknown

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 interval control processes an EXEC CICS START command and discovers the associated termid is not defined to the system. XALTENF is driven from the terminal allocation program also when the termid is not defined.

The terminal allocation program schedules requests resulting both from the eventual execution of an EXEC CICS START command and from the transient data queue trigger mechanism. This means that an EXEC CICS 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 this information:

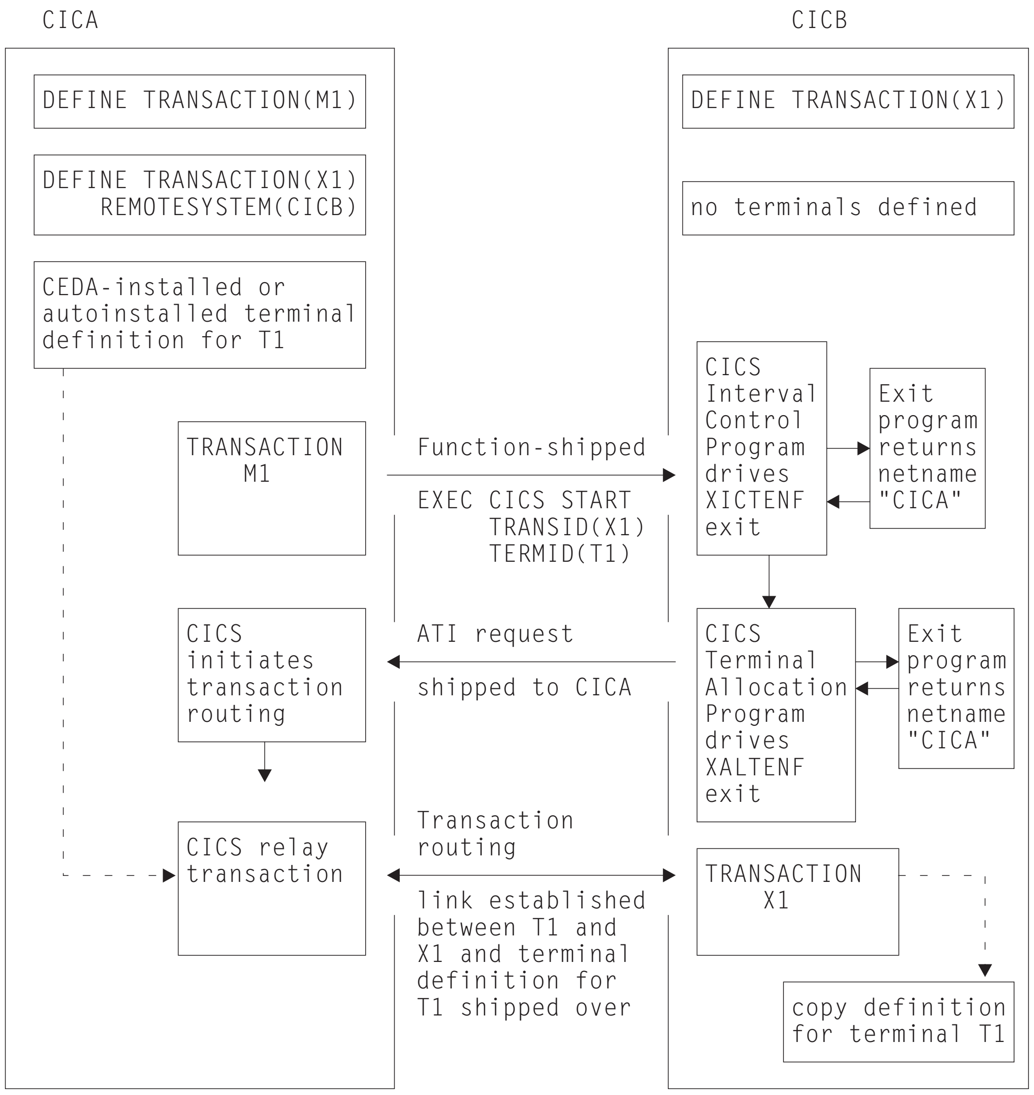
- Whether the ATI request resulted from: an EXEC CICS START command with data, a START command without data, or a transient data queue trigger.
- Whether the START command was issued by a transaction that had been the subject of transaction routing.
- Whether the START command was function-shipped from another region.
- The identifier of the transaction to be run.
- The identifier of the terminal with which the transaction should run.
- 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 returned netname.

On exit from the program, you tell 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 therefore a solution to the problem shown in Figure 19 on page 59. It is necessary only 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 in Figure 20 on page 61:

1. The interval control program accepts the EXEC CICS START command and signals acceptance to the issuing system if this is required.
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.



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

Figure 20. Resolving a 'terminal not known' condition on a START request

The example in Figure 20 shows only one of many possible configurations. From this elementary example, you can see how to approach a solution for the more complex situations that can arise in multiregion networks.

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

Note: The 'terminal not known' condition can arise in CICS terminal-allocation modules during restart, before any global user exit programs have been enabled. If you want to intervene here too, you must enable your XALTENF exit program in a first-phase PLTPI program (for programming information about PLTPI programs, see the *CICS Customization Guide*). This applies to both warm start and emergency start.

Important

The XICTENF and XALTENF exits can be used only if there is a direct link between the AOR and 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 EXEC CICS START request. In a more complex situation, you may decide to give each terminal a name that reflects the system on which it resides.

For programming information about the exit program, see the *CICS Customization Guide*. A sample program, DFHXTENF, is available in the VSE/ESA sublibrary (PRD1.BASE).

Shipping terminals for ATI from multiple TORs

Consider the following network setup:

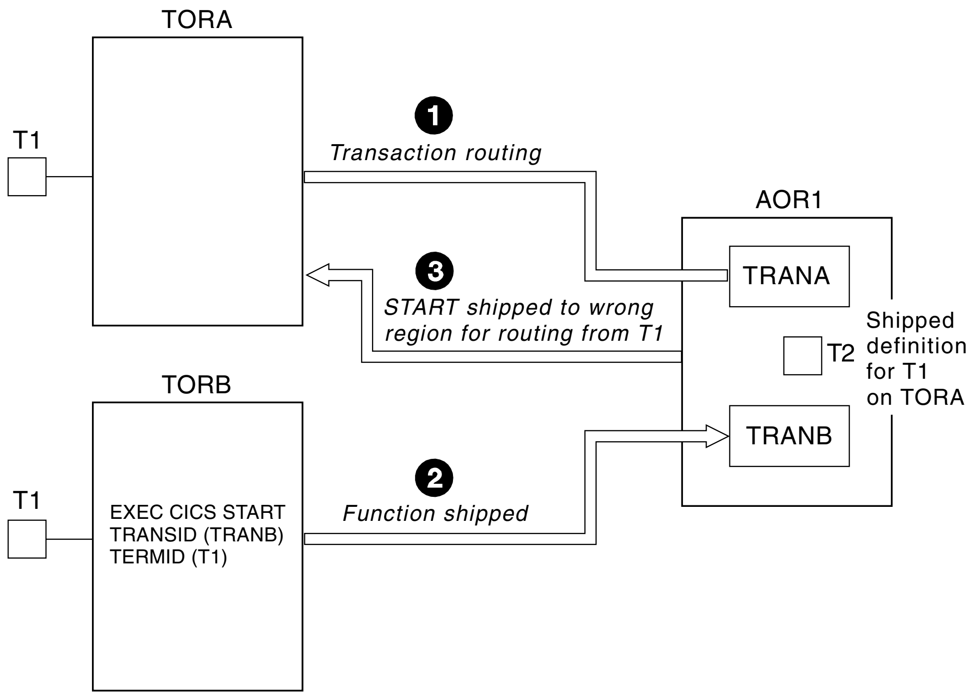
1. You have an application-owning region that is connected to two or more terminal-owning regions (TORs) that use the same, or a similar, set of terminal identifiers.
2. One or more of the TORs issues EXEC CICS START requests for transactions in the AOR.
3. The EXEC CICS START requests are associated with terminals.
4. You are using shippable terminals, rather than statically defining remote terminals in the AOR.

Now consider the following scenario:

Terminal-owning region TORB issues an EXEC CICS START request for transaction TRANB, which is owned by region AOR1. It is to be run against terminal T1. Meanwhile, terminal T1 <u>on region TORA</u> *has been transaction routing to AOR1.; a definition of T1 has been shipped to AOR1 from TORA. When the START request arrives at AOR1, it is shipped to TORA,* <u>rather than TORB</u>*, for transaction routing from terminal T1.*

Figure 21 on page 63 illustrates what happens.

Figure 21. Function-shipped START request started against an incorrect terminal. Because a shipped definition of terminal T1 (owned by TORA) is installed on AOR1, the START request received from TORB is shipped to TORA, for routing, rather than to TORB.


TORA
T1
1
Transaction routing
AOR1
TRANA
3
START shipped to wrong region for routing from T1
T2
Shipped definition for T1 on TORA
TORB
TRANB
2
T1
EXEC CICS START TRANSID (TRANB) TERMID (T1)
Function shipped

To prevent this situation, code `'YES'` on the FSSTAFF system initialization parameter in the AOR. This ensures that, when a START request is received from a terminal-owning region, and a shipped definition for the terminal named on the request is already installed in the AOR, the request is always shipped back to a TOR, for routing, *across the link it was received on*, irrespective of the TOR referenced in the remote terminal definition. (The only exception to this is if the START request supplies a TOR_NETNAME and a remote terminal with the correct TOR_NETNAME is located; in which case, the request is shipped to the appropriate TOR.)

If the TOR to which the START request is returned is **not** the one referenced in the installed remote terminal definition, a definition of the terminal is shipped to the AOR, and the autoinstall user program is called. Your autoinstall user program can then allocate an *alias* termid in the AOR, to avoid a conflict with the previously installed remote definition. Terminal aliases are described on page 155. For information about writing an autoinstall program to control the installation of shipped definitions, see the *CICS Customization Guide*.

For full details of the FSSTAFF system initialization parameter, see the *CICS System Definition Guide*.

Allocation of remote APPC connections

A transaction running in the application-owning region can issue an EXEC CICS ALLOCATE command, to obtain a session to an APPC terminal or connection that is owned by another system.

A relay program is started in the terminal-owning region to convey requests between the transaction and the remote APPC system or terminal.

Transaction routing with APPC devices

An APPC device presents a data interface to CICS that is an implementation of the APPC architecture. The APPC session linking it to a transaction represents the principal facility of the transaction rather than the device itself. The transaction converses across the link with a transaction program within the device, which may be a hard-coded terminal device, a programmable system, or even another CICS system.

There is no essential difference between transaction routing with APPC devices and transaction routing with any other terminals. However, remember these points:

- APPC devices have their own “intelligence”. They can interpret operator input data or the data received from CICS in any way the designer chooses.
- There are no error messages from CICS. The APPC device receives indications from CICS, which it may translate into text for a human operator.
- CICS does not directly support pseudoconversational operation for APPC devices, but the device itself could possibly be programmed to produce the same effect.
- Basic mapping support (BMS) has no meaning for APPC devices.
- APPC devices can be linked by more than one session to the host system.
- TCTUAs will be shipped across the connection for APPC single-session terminals, but not when the principal facility is an APPC parallel session.

You use the APPC application program interface to communicate with APPC devices. For relevant introductory information, see Chapter 9, “Distributed transaction processing” on page 71.

Allocating an alternate facility

One of the design criteria in transaction routing is that, if a transaction running in a single-CICS environment is transferred to an alternative, linked system, there should be no loss of function if the transaction now has to be routed to the original terminal.

Because an APPC device can have more than one session, it is possible, in the single-CICS case, for a transaction to acquire further sessions to the same device (but to different tasks) by using the EXEC CICS ALLOCATE command. Each session thus acquired becomes an **alternate facility** to the transaction. Sessions can also be established to other terminals or systems.

Similarly, transaction routing allows any transaction to acquire an alternate facility to an APPC device by using EXEC CICS ALLOCATE, even though there are intermediate systems between the APPC device and the AOR. For this, the AOR needs a remote version of the APPC link definition that is installed in the TOR. Perhaps you can rely on this having been shipped to the AOR by a transaction routing operation. If not, you will have to install it expressly. You cannot use the user exits XICTENF and XALTENF as an aid to routing the alternate facility.

The system as a terminal

Because the resource definitions for APPC devices can take the CONNECTION and SESSIONS form, it is easy to confuse them with the definitions for the intersystem links. It is important to remember that definitions for the intersystem links are either **direct** or **indirect**, while those for APPC devices are **direct** in the TOR and **remote** in the AOR and any intermediate systems. Note also that remote CONNECTION definitions do not need corresponding SESSIONS definitions.

Figure 22 shows a network of three CICS systems chained together, of which the first is linked to an APPC terminal.

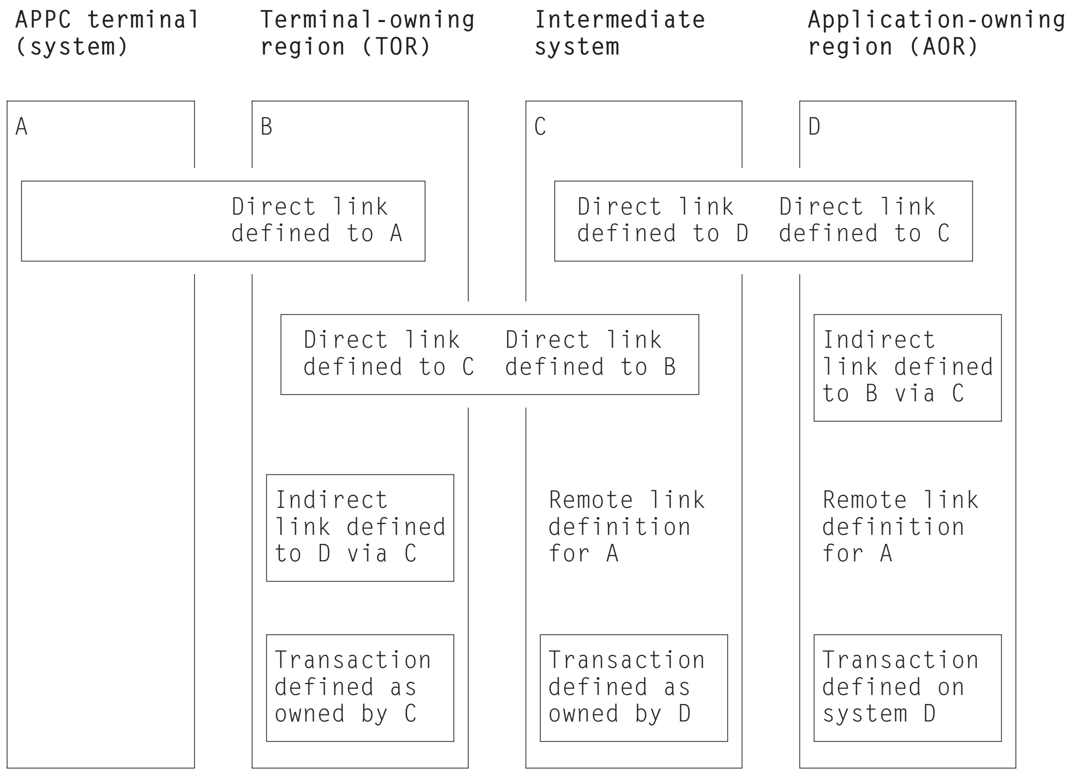

APPC terminal (system)
Terminal-owning region (TOR)
Intermediate system
Application-owning region (AOR)
A
B
C
D
Direct link defined to A
Direct link defined to D
Direct link defined to C
Direct link defined to C
Direct link defined to B
Indirect link defined to B via C
Indirect link defined to D via C
Remote link definition for A
Remote link definition for A
Transaction defined as owned by C
Transaction defined as owned by D
Transaction defined on system D

Figure 22. Transaction routing to an APPC terminal across daisy-chained systems

Notes:

1. The remote link definitions for A could either be defined by the user or be shipped from system B during transaction routing.
2. The indirect links are not necessary to this example, but are included to complete all possible linkage combinations. See “Indirect links for transaction routing” on page 121.
3. The links B-C and C-D may be either MRO or APPC.

System A (or any one of the four systems) can take on the role of a terminal. This is a technique that allows a pair of transactions to converse across intermediate systems. Consider this sequence of events:

1. A transaction running in A allocates a session on the link to B and makes an attach request for a particular transaction.

2. B sees that the transaction is on C, and initiates the relay program in conjunction with the principal facility represented by the link definition to A.
3. The attach request arrives at C together with details of the terminal; that is, B's link to A. C builds a remote definition of the terminal and goes to attach the transaction.
4. C also finds the transaction **remote** and defined as owned by D. C initiates the relay program, which tries to attach the transaction in D.
5. D also builds a remote definition of B's link to A, and attaches the local transaction.
6. The transaction in A that originated the attach request can now communicate with the target transaction through the transaction routing mechanism.

Note these points:

- APPC terminals are always shippable. There is no need to define them as such.
- Attach requests on other sessions of the A-B link could be routed to other systems.
- Neither partner to a conversation made possible by transaction routing knows where the other resides, although the routed-to transaction can find out the TERMINAL/CONNECTION name by using the EXEC CICS ASSIGN PRINSYSID command. This name can be used to allocate one or more additional sessions back to A.
- The transaction in D could start with an EXEC CICS (GDS) EXTRACT PROCESS command, but it is more usual for the transaction to start with an EXEC CICS (GDS) RECEIVE command.

The relay program

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

Although CICS determines the program to be associated with the transaction, the user's definition for the remote transaction determines the attributes. These are usually those of the "real" transaction in the remote system.

Because it executes the relay program, the transaction is 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 through the relay transaction to the user transaction.

Automatic transaction initiation (ATI) 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.

Basic mapping support (BMS)

The mapping operations of BMS are performed in the system on which the user's transaction is running; that is, in the application-owning region. 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 application-owning region. When the logical message is complete, the pages are shipped to the terminal-owning region (or regions, if they were generated by a routing request), and deleted from the application-owning 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 EXEC CICS ROUTE command to route BMS messages to remote terminals. For programming information about the BMS EXEC CICS ROUTE command, see the *CICS Application Programming Reference* manual. 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 5 shows how the possible combinations of route list entries and OPCLASS options govern the delivery of routed messages to remote terminals. In all cases, the remote terminal must be defined in the system that issues the EXEC CICS ROUTE command (or a shipped terminal definition must already be available; see "Shipping terminal and connection definitions" on page 147). Note that the facility described in "Shipping terminals for automatic transaction initiation" on page 58 does not apply to terminals addressed by the EXEC CICS ROUTE command.

Table 5 (Page 1 of 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.

Table 5 (Page 2 of 2). *BMS message routing to remote terminals and operators*

LIST entry	OPCLASS	Result
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.

The routing transaction (CRTE)

The routing transaction (CRTE) is a CICS-supplied 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 the terminal operator needs ESM authority if the remote system is protected. 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 option of the DEFINE CONNECTION command, 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 163.) 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 CESN; 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 the master terminal transaction, CEMT, on a particular remote system. It avoids the necessity of installing a definition of the remote CEMT in the local system. CRTE is also useful for testing remote transactions before final installation.

System programming considerations

You 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 and system definition" on page 81.
2. Define MRO or ISC links between the systems that are to be connected, as described in Chapter 12, "Defining links to remote systems" on page 93.
3. Define the terminals and transactions that will participate in transaction routing, as described in Chapter 14, "Defining remote resources" on page 137.
4. Ensure that the local communication profiles, transactions, and programs required for transaction routing are defined and installed on the local system, as described in Chapter 15, "Defining local resources" on page 163.
5. If you want to use dynamic transaction routing, customize the supplied dynamic transaction routing program, DFHDYP, or write your own version. For programming information about how to do this, see the *CICS Customization Guide*.
6. 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. For programming information about coding these exits, see the *CICS Customization Guide*.

Intersystem queuing

If the link to a remote region is established, but there are no free sessions available, transaction routing requests may be queued in the issuing region. Performance problems can occur if the queue becomes excessively long.

For guidance information about controlling intersystem queues, see Chapter 23, "Intersystem session queue management" on page 221.

Chapter 9. Distributed transaction processing

This chapter contains the following topics:

Overview of DTP

When CICS arranges function shipping, distributed program link (DPL), asynchronous transaction processing, or transaction routing for you, it establishes a logical data link with a remote system. A data exchange between the two systems then follows. This data exchange is controlled by CICS-supplied programs, using APPC, LUTYPE6.1, or MRO protocols. The CICS-supplied programs issue commands to allocate conversations, and send and receive data between the systems. Equivalent commands are available to application programs, to allow applications to converse. The technique of distributing the functions of a transaction over several transaction programs within a network is called **distributed transaction processing (DTP)**.

Of the five intercommunication facilities, DTP is the most flexible and the most powerful, but it is also the most complex. This chapter introduces you to the basic concepts.

For guidance on developing DTP applications, see the *CICS Distributed Transaction Programming Guide*.

Advantages over function shipping and transaction routing

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. 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 supermarket 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.*

An analyst might be tempted to use function shipping to write each reorder record to a remote file as it arises. This method has the virtue of simplicity, but must be rejected for several reasons:

- Data is transmitted to the remote systems irregularly in small packets. This causes inefficient use of the links.
- The transactions associated with the point-of-sale devices are competing for sessions with the remote systems. This could cause unacceptable delays at point-of-sale.
- Failure of a link results in a catastrophic suspension of operations at a branch.
- Intensive intercommunication 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.

Restocking requests are seldom urgent, so 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 to trigger the sender transaction when a predefined data level is reached. Either way, the sender transaction has the same job to do.

Again, it is tempting 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 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 requester.

This is a lot of activity—two flows on the network; and the data flow can be quite significant. 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 incurred in 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. DTP does the following:

- Allows some measure of parallel processing to shorten response times
- Provides a common interface to a transaction that is to be attached by several different transactions
- Enables communication with applications running on other systems, particularly on non-CICS systems
- 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
- Enables batching of less urgent 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 terminal-initiated transaction at the very top. Figure 23 on page 74 shows a possible configuration. Transaction TRAA is attached over the terminal session. Transaction TRAA attaches transaction TRBB, which, in turn, attaches transactions TRCC and TRDD. Both these transactions attach the same transaction, SUBR, in system CICSE. This gives rise to two different tasks of SUBR.

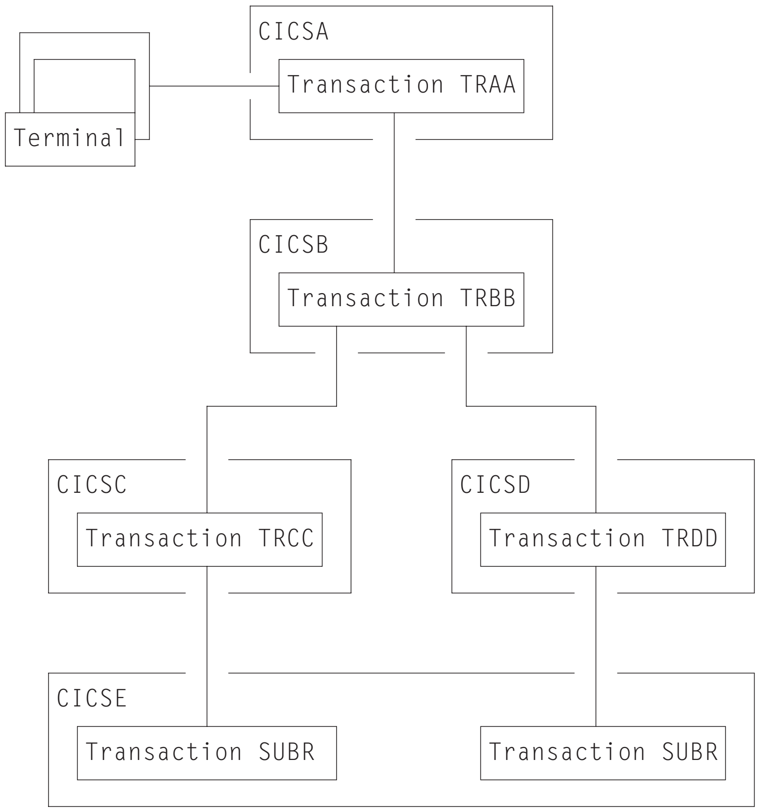


Figure 23. DTP in a multisystem configuration

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 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. (Some books refer to the front end as the initiator and the back end as the recipient.) 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 complication. This is further explained in the discussion on synchronization later in this chapter.

Dialog between two transactions

A conversation transfers data from one transaction to another. For this to function properly, each transaction must know what the other intends. It would be nonsensical for the front end to send data if all the back end wants to do is print out the weekly sales report. 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 71, 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 an EXEC CICS SEND command for each record that it takes off the queue. The back end issues EXEC CICS 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 an EXEC CICS 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 APPC 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.

The APPC 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, non-urgent control indicators are accumulated until it is necessary to send user data, at which time they are added to the header.

For the formats of the headers and control indicators used by APPC, see the *SNA Formats* manual.

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.

MRO is not unlike APPC in its internal organization. It is based on LUTYPE6.1, which is also an 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, the front end cannot receive any more data on the conversation.

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. If anything goes wrong during the running of a transaction, the associated resources should not be left in an inconsistent 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 intercommunication 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 two problems. On one side, you need to restore the queue entries that you have sent; on the other side, you need to delete the corresponding entries in the DASD file.

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 in which 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 restoring 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, you could 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 this overhead, CICS arranges resource recovery for you. LU management works 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**. Syncpoints are implied at the beginning and end of a transaction. A transaction can define other syncpoints by program command. All processing between two consecutive 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 syncpoint are made irreversible.

Although CICS commits and backs out changes to resources for you, the service must be paid for in performance. You might have transactions that do not need such complexity, 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 sync levels

The APPC architecture defines three levels of synchronization (called **sync levels**):

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 EXEC CICS 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. Transaction abend 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 if sync level 2 has been selected for the conversation between them.

Remember that syncpoint and rollback are not peculiar to any one conversation within a transaction. They are propagated on every sync level 2 conversation that is currently *in bracket*.

MRO or APPC for DTP?

You can program DTP applications for both MRO and APPC 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.

Choosing between MRO and APPC can be quite simple. The options depend on the configuration of your CICS complex and on the nature of the conversation partner. A CICS Transaction Server for VSE/ESA transaction can use MRO only to communicate with a partner in a CICS Transaction Server for VSE/ESA or

CICS/VSE 2.3 system in the same VSE/ESA operating system. It cannot use MRO to communicate with a partner in a non-CICS system.

For communication with a partner in another CICS system, where the CICS systems are in the same VSE/ESA operating system, you can use either the MRO or the APPC protocol. There are good performance reasons for using MRO. But if there is any possibility that the distributed transactions will need to communicate with partners in other operating systems, it is better to use APPC so that the transaction remains unchanged.

Table 6 summarizes the main differences between the two protocols.

Table 6. MRO compared with APPC

MRO	**APPC**
Function is realized within CICS	Depends on VTAM or similar
Nonstandard architecture	SNA architecture
CICS-to-CICS links only	Links to non-CICS systems possible
Communicates within single VSE/ESA operating system	Communicates across multiple VSE/ESA and other operating systems
PIP data not supported	PIP data supported
Data transmission not deferred	Deferred data transmission
Partner transaction identified in data	Partner transaction defined by program 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 flows expedited
WAIT command has no function	WAIT command causes transmission of deferred data

APPC 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. In mapped conversations, you 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 in EXEC CICS commands for basic conversations.

Table 7 summarizes the differences between mapped and basic conversations. Note that it only applies to the CICS API. CPI Communications, introduced in the next section, has its own rules.

Table 7. APPC conversations – mapped or basic?

Mapped	**Basic**
The conversation partners exchange data that is relevant only 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 in assembler language or C only.

EXEC CICS or CPI Communications?

CICS Transaction Server for VSE/ESA Release 1 gives you a choice of two application programming interfaces (APIs) for coding your DTP conversations on APPC sessions. The first, the **CICS API**, is the programming interface of the CICS implementation of the APPC architecture. It consists of EXEC CICS commands and can be used with all CICS-supported languages. The second, **Common Programming Interface Communications** (CPI Communications) is the communication interface defined for the SAA environment. It consists of a set of defined verbs, in the form of program calls, which are adapted for the language being used.

Table 8 compares the two methods to help you to decide which API to use for a particular application.

Table 8. CICS API compared with CPI Communications

CICS API	**CPI Communications**
Portability between different members of the CICS family.	Portability between systems that support SAA facilities.
Basic conversations can be programmed only in assembler language or C.	Basic conversations can be programmed in any of the available languages.
Sync levels 0, 1, and 2 supported.	Sync levels 0, 1, and 2 supported, *except for transaction routing, for which only sync levels 0 and 1 are supported*.
PIP data supported.	PIP data not supported.
Only a few conversation characteristics are programmable. The rest are defined by resource definition.	Most conversation characteristics can be changed dynamically by the transaction program.
Can be used on the principal facility to a transaction started by ATI.	Cannot be used on the principal facility to a transaction started by ATI.
Limited compatibility with MRO.	No compatibility with MRO.

You can mix CPI Communications calls and EXEC CICS commands in the same transaction, but not on the same side of the same conversation. You can implement a distributed transaction where one partner to a conversation uses CPI Communications calls and the other uses the CICS API. In such a case, it would be up to you to ensure that the APIs on both sides map consistently to the APPC architecture.

Part 2. Installation and system definition

Table 9. Installation road map

If you want to...	**Refer to...**
Set up CICS to support multiregion operation	Chapter 10, "Installation considerations for multiregion operation" on page 83
Set up CICS to support intersystem communication	Chapter 11, "Installation considerations for intersystem communication" on page 85

Chapter 10. Installation considerations for multiregion operation

This chapter discusses those aspects of installation that apply particularly to CICS multiregion operation. It contains the following topics:

- "Installation steps"
- "Batching of work in an MRO region"
- "Logging on to the IRC access method" on page 84
- "MRO subtask failure" on page 84

Installation steps

To use CICS MRO, you must:

1. Ensure that the required CICS modules are included in your CICS system.
2. Place some modules in the SVA.

Installing modules required for MRO

To use MRO, you must include the intersystem communication management modules in your system. To do this, specify `'YES'` on the ISC system initialization parameter (the default value is `'NO'`). For information about system initialization parameters, see the *CICS System Definition Guide*.

Installing MRO modules in the shared virtual area

For multiregion operation, certain modules must be resident in the shared virtual area.

You must place the following modules in the shared virtual area (SVA) of VSE/ESA:

- DFHIRP, the CICS interregion communication program
- DFHSCTE, the CICS subsystem control table extension module
- DFHCSEOT, the CICS end-of-job cleanup routine

During execution, DFHIRP acquires storage in the SVA GETVIS area for control blocks and data buffers. The maximum amount of storage used, which depends on how heavily MRO is used, is given in a CICS message produced when CEMT SET IRC CLOSED is issued or a warm shutdown is performed.

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. (Posting means notifying that a region should start processing the queue.) 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 system initialization parameter. The number can have a value in the range 1 through 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 system initialization parameter. 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.

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.

MRO subtask failure

CICS interregion communication uses a VSE/ESA subtask. If this subtask abends, or detects an unrecoverable error, VSE/ESA issues the following message:

```
OS121  SUB DFHIRPST CANCELED
```

IRC can be restarted by issuing the commands:

```
CEMT SET IRC CLOSED
CEMT SET IRC OPEN
```

Chapter 11. 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 VTAM and IMS when these products are to be used with CICS in an intersystem communication environment.

The chapter contains the following topics:

- “Installing modules required for ISC”
- “VTAM definition for CICS”
- “Considerations for IMS” on page 86

The information on VTAM and IMS given in this chapter is for guidance only. Always consult the current VTAM or IMS publications for the latest information.

Installing modules required for ISC

To use ISC, you must include the intersystem communication management modules in your system. To do this, specify 'YES' on both the ISC and VTAM system initialization parameters. For information about system initialization parameters, see the *CICS System Definition Guide*.

VTAM definition for CICS

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

MODETAB=logon-mode-table-name
This operand names the VTAM logon mode table that contains your customized logon mode entries (see “VTAM LOGMODE table entries for CICS” on page 86). 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 MODIFY vtamname USERVAR command. (For further information about the significance of USERVARs, see the *CICS XRF Guide*.) 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 operand 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.

Take care when selecting 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 operand 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 operand specifies LUTYPE6 parallel session support.

SONSCIP=YES
This operand specifies session outage notification (SON) support. SON enables CICS, in particular cases, to recover a failed session without requiring operator intervention.

APPC=NO
This is necessary to let CICS use VTAM macros. CICS does not issue the APPCCMD macro.

For further information about the VTAM APPL statement, see the *Advanced Communication Function for VTAM Installation and Resource Definition* manual.

VTAM LOGMODE table entries for CICS

For APPC sessions, you can use the MODENAME operand (see "Syncpoint exchanges" on page 234) 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 APPC links must be matched by a VTAM LOGMODE name. All that is required are entries of the following form:

```
MODEENT LOGMODE=modename,COS=cosname
MODEEND
```

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

```
MODEENT LOGMODE=SNASVCMG,COS=cosname
MODEEND
```

If you plan to use autoinstall for single-session APPC terminals, additional information is required in the MODEENT entry. For guidance information about coding the VTAM LOGON mode table, see the *CICS 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 read "Indirect links for transaction routing" on page 121, especially the section on defining compatible CICS and IMS nodes. For full details of IMS installation, see the installation guide for the IMS product.

VTAM definition for IMS

When the IMS system is defined to VTAM the following operands 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 operand 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 network addressable units must include the total number of parallel sessions for this IMS system.

PARSESS=YES
: This operand specifies LUTYPE6 parallel session support.

For further information about the VTAM APPL statement, see the *Advanced Communication Function for VTAM (ACF/VTAM) Installation and Resource Definition* manual.

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 in (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 VTAM defaults

Figure 24 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 24. A typical IMS logmode table entry

IMS system definition for intersystem communication

This section summarizes the IMS ISC-related macros and parameters that are used in IMS system definition. You should also see "Defining compatible CICS and IMS nodes" on page 114. For full details of IMS installation, see the installation guide for the IMS product.

The COMM macro

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

However, for an IMS Version 2, or later, 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.ADFHPROC, 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.

This size applies to all other 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-to-IMS ISC, an alias name must not be longer than four characters.

The TYPE macro

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

The TERMINAL macro identifies the remote CICS system to IMS. It therefore serves the equivalent purpose to DEFINE CONNECTION in CICS.

NAME=name
Identifies the CICS node to IMS. It must be the same as the applid 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 option of the DEFINE SESSIONS commands 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=[NOLTWA|LTWA]
Specifies whether Log Tape Write Ahead (LTWA) is required. For LTWA, IMS logs session restart information for all active parallel sessions before sending a syncpoint request. LTWA is recommended for integrity reasons, but it can adversely affect performance. NOLTWA is the default.

OPTIONS=[SYNCSESS|FORCSESS]
Specifies the message resynchronization requirement following an abnormal session termination. SYNCSESS is the default. It requires both the incoming and the outgoing 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 is determined on a transaction-by-transaction basis. This is the default.

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 with 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.

TRANSRESP is recommended.

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

{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 parameters of the SUBPOOL macro.

The following types of component can be defined:

SINGLE1
Used by IMS for asynchronous output. One output message is sent for each 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.

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

The VTAMPOOL macro

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

The SUBPOOL macro

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

NAME=subpool-name
Specifies the IMS name for this session. A CICS-to-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 option of the DEFINE SESSIONS command for the session.

The IMS subpool name is specified to CICS in the NETNAMEQ option of the DEFINE SESSIONS command.

The NAME macro

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

COMPT={1|2|3|4}
Specifies the output component associated with this session. The component specified determines the protocol that IMS ISC uses 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 uppercase (UC) or not (ULC) before transmission.

Part 3. Resource definition

Table 10. Resource definition road map

If you want to...	**Refer to...**
Define connections to remote CICS or non-CICS systems	Chapter 12, “Defining links to remote systems” on page 93
Define resources owned by other systems to the local CICS system	Chapter 14, “Defining remote resources” on page 137
Define local resources that are needed for intersystem communication	Chapter 15, “Defining local resources” on page 163

Chapter 12. Defining links to remote systems

This chapter tells you how to define and manage communication links to other systems or to other CICS regions. *It is intended to be read in conjunction with the CICS Resource Definition Guide*. The types of link described are:

- Links for multiregion operation
- Links for use by the external CICS interface (EXCI)
- Links to remote systems using logical unit type 6.2 (APPC) protocols
- Links to remote systems using logical unit type 6.1 protocols
- 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 APPC links.

The chapter contains the following topics:

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

The remote system is defined by the CEDA DEFINE CONNECTION command. Each session, or group of parallel sessions, is defined by the CEDA DEFINE SESSIONS command. The definitions of the remote system and the sessions are always separate, and are not associated with each other until they are installed.

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

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, APPC terminals), the link definition must match the fixed attributes of the remote system concerned.

Naming the local CICS system

A CICS system can be known by more than one name:

- Application identifier (applid)
- System identifier (sysidnt)

All CICS systems have an applid and a sysidnt.

The applid of the local CICS system

The 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 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.

For ISC, the applid is used on a VTAM APPL statement, to identify CICS to VTAM.

You specify the applid on the APPLID system initialization parameter. The default value is DBDCCICS. This value can be overridden during CICS startup.

All applids in your intercommunication network should be unique.

Note: CICS systems that use XRF have *two* applids, to distinguish between the active and alternate systems. This special case is described in "Generic and specific applids for XRF" on page 125.

The sysidnt of the local CICS system

The sysidnt of a CICS system is a name (1–4 characters) 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**

Note: 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 14, "Defining remote resources" on page 137.) The sysidnt of a running CICS system is always the one specified by the system initialization parameters.

Identifying remote systems

In addition to 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 (its netname), is made when you define the intercommunication link. For example:

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

If NETNAME is omitted, sysidnt must be coded explicitly as the applid of the remote system. Each sysidnt name defined to a CICS system must be unique.

Defining links for multiregion operation

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

Note: The external CICS interface (EXCI) uses a specialized form of MRO link, that is described on page 99. This present section describes MRO links between CICS systems. However, most of its contents apply also to EXCI links, except where noted otherwise on page 99.

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 subsequent 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 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.

Defining an MRO link

The definition for an MRO link is shown in Figure 25 on page 96.

Note: For reasons of clarity and conciseness, inapplicable and inessential options have been omitted from Figure 25 on page 96, and from all the example definitions in this chapter, and no attempt has been made to mimic the layout of the CEDA DEFINE panels. For details of all RDO options, refer to the *CICS Resource Definition Guide*.

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:

- The CONNECTION and SESSIONS must be in the same GROUP.
- The SESSIONS must have PROTOCOL(LU61), but the PROTOCOL option of CONNECTION must be left blank.

- The CONNECTION option of SESSIONS must match the sysidnt specified for the CONNECTION.
- Only one SESSIONS definition can be related to an MRO CONNECTION.
- There can be only one MRO link between any two CICS regions; that is, each DEFINE CONNECTION must specify a unique netname.

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 with which the remote system logs on to the interregion SVC; that is, its applid. If you do not specify a netname, then sysidnt must satisfy these requirements.

```
DEFINE
  CONNECTION(sysidnt)
  GROUP(groupname)
  NETNAME(name)
  ACCESSMETHOD(IRC)
  QUEUELIMIT(NO|0-9999)
  MAXQTIME(NO|0-9999)
  INSERVICE(YES)
  ATTACHSEC(LOCAL|IDENTIFY)
  USEDFLTUSER(NO|YES)
DEFINE
  SESSIONS(csdname)
  GROUP(groupname)
  CONNECTION(sysidnt)
  PROTOCOL(LU61)
  RECEIVEPFX(prefix1)
  RECEIVECOUNT(number1)
  SENDPFX(prefix2)
  SENDCOUNT(number2)
  SESSPRIORITY(number)
  IOAREALEN(value)
```

Figure 25. Defining an MRO link

On the CONNECTION definition, the QUEUELIMIT option specifies the maximum number of requests permitted to queue for free sessions to the remote system. The MAXQTIME option specifies the maximum time between a queue becoming full and it being purged because the remote system is unresponsive. Further information is given in Chapter 23, “Intersystem session queue management” on page 221.

On the SESSIONS definition, you must specify the number of SEND and RECEIVE sessions that are required (at least one of each). You can also specify the prefixes which allow the sessions to be named. A prefix is a one-character or two-character string that is used to generate session identifiers (TRMIDNTs). If you do not specify prefixes, they default to '>' (for SEND) and '<' (for RECEIVE). It is recommended that you allow the prefixes to default, because:

- This guarantees that the session names generated by CICS are unique—prefixes must not cause a conflict with an existing connection or terminal name.

- If you specify your own 2-character prefixes, the number of sessions you can define for each connection is limited to 99. If you specify your own 1-character prefixes, the limit increases to 999—the same as for default prefixes—but you may find it harder to guarantee unique session names.

If you want to specify your own MRO session prefixes, the method is the same as that for LUTYPE6.1 sessions, described on page 111.

For an explanation of how CICS generates names for MRO sessions, see the *CICS Resource Definition Guide*.

Figure 26 shows a typical definition for an MRO link.

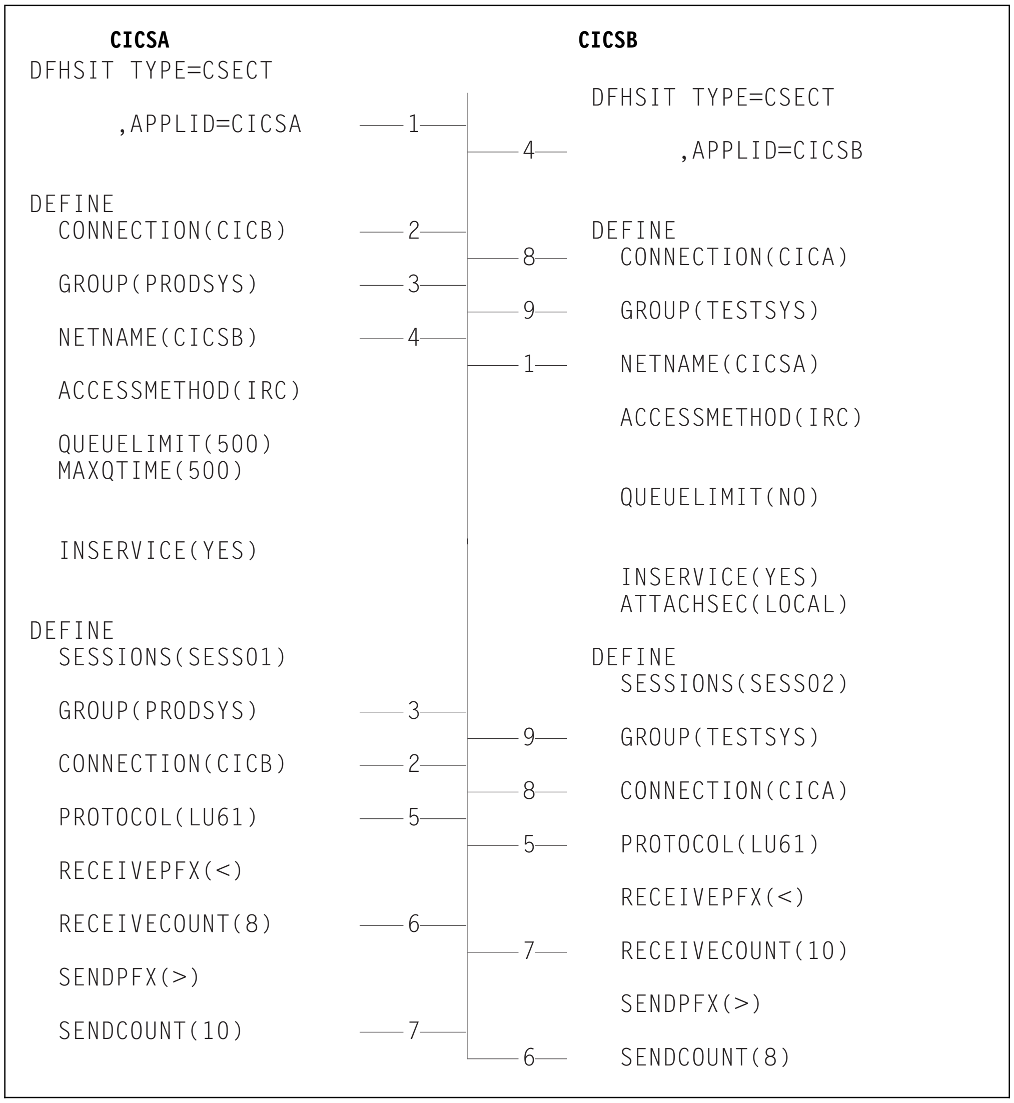
```
DEFINE
  CONNECTION(CICB)     local name for remote system
  GROUP(groupname)     groupname of related definitions
  NETNAME(CICSB)       applid of remote system
  ACCESSMETHOD(IRC)
  QUEUELIMIT(NO)       if no free sessions, queue all requests
  INSERVICE(YES)
  ATTACHSEC(LOCAL)     use security of the link only
  USEDFLTUSER(NO)
DEFINE
  SESSIONS(csdname)    unique csd name
  GROUP(groupname)     same group as the connection
  CONNECTION(CICB)     related connection
  PROTOCOL(LU61)
  RECEIVEPFX(<)
  RECEIVECOUNT(5)      5 receive sessions
  SENDPFX(>)
  SENDCOUNT(3)         3 send sessions
  SESSPRIORITY(100)
  IOAREALEN(300)       minimum TIOA size for sessions
```

Figure 26. Example of MRO link definition

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 six sending sessions, the other definition requires six receiving sessions.

The compatibility requirements are shown in Figure 27.



```
     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)
  NETNAME(CICSB)         ——4——
                              ——1——          NETNAME(CICSA)
  ACCESSMETHOD(IRC)
                                             ACCESSMETHOD(IRC)
  QUEUELIMIT(500)
  MAXQTIME(500)
                                             QUEUELIMIT(NO)

  INSERVICE(YES)
                                             INSERVICE(YES)
                                             ATTACHSEC(LOCAL)
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(<)
                                             RECEIVEPFX(<)
  RECEIVECOUNT(8)        ——6——
                              ——7——          RECEIVECOUNT(10)
  SENDPFX(>)
                                             SENDPFX(>)
  SENDCOUNT(10)          ——7——
                              ——6——          SENDCOUNT(8)
```

Figure 27. Defining compatible MRO nodes

In Figure 27, related options are shown by the numbered paths, all of which pass through the central connecting line.

Defining links for use by the external CICS interface

This section describes how to define connections for use by non-CICS programs using the external CICS interface (EXCI) to link to CICS server programs. The definitions required are similar to those needed for MRO links between CICS systems. Each connection requires a CONNECTION and a SESSIONS definition.

Because EXCI connections are used for processing work from external sources, you must not define any SEND sessions.

EXCI connections can be defined as “specific” or “generic”. A specific EXCI connection is an MRO link on which all the RECEIVE sessions are dedicated to a single user (client program). A generic EXCI connection is an MRO link on which the RECEIVE sessions are shared by multiple users. Only one generic EXCI connection can be defined on each CICS region.

On definitions of both specific and generic connections, you must:

- Specify PROTOCOL(EXCI).
- Specify ACCESSMETHOD(IRC).
- Let SENDCOUNT and SENDPFX default to blanks

Figure 28 shows the definition of a specific EXCI connection.

```
DEFINE
  CONNECTION(EIP1)     local name for connection
  GROUP(groupname)     groupname of related definitions
  NETNAME(CLAP1)       User name on INITIALIZE_USER command
  ACCESSMETHOD(IRC)
  PROTOCOL(EXCI)
  CONNTYPE(Specific)   pipes dedicated to a single user
  INSERVICE(YES)
  ATTACHSEC(LOCAL)
DEFINE
  SESSIONS(csdname)    unique csd name
  GROUP(groupname)     same group as the connection
  CONNECTION(EIP1)     related connection
  PROTOCOL(EXCI)       external CICS interface
  RECEIVEPFX(<)
  RECEIVECOUNT(5)      5 receive sessions
  SENDPFX              leave blank
  SENDCOUNT            leave blank
```

Figure 28. Example definition for a specific EXCI connection. For use by a non-CICS client program using the external CICS interface.

For a specific connection, NETNAME must be coded with the name of the user program that will be passed on the EXCI INITIALIZE_USER command. CONNTYPE must be Specific.

Figure 29 on page 100 shows the definition of a generic EXCI connection.

```
DEFINE
  CONNECTION(EIP2)     local name for connection
  GROUP(groupname)     groupname of related definitions
  ACCESSMETHOD(IRC)
  NETNAME()            must be blank for generic connection
  INSERVICE(YES)
  PROTOCOL(EXCI)
  CONNTYPE(Generic)    pipes shared by multiple users
  ATTACHSEC(LOCAL)
DEFINE
  SESSIONS(csdname)    unique csd name
  GROUP(groupname)     same group as the connection
  CONNECTION(EIP2)     related connection
  PROTOCOL(EXCI)       external CICS interface
  RECEIVEPFX(<)
  RECEIVECOUNT(5)      5 receive sessions
  SENDPFX              leave blank
  SENDCOUNT            leave blank
```

Figure 29. Example definition for a generic EXCI connection. For use by non-CICS client programs using the external CICS interface.

For a generic connection, NETNAME must be blank. CONNTYPE must be Generic.

Installing MRO and EXCI link definitions

You cannot install new MRO or EXCI connections, or modify existing connections, while IRC is open. You must close IRC before installing or modifying MRO or EXCI connections, and then open IRC to cause CICS to make use of the new links.

For further information about installing MRO links, see the *CICS Resource Definition Guide*.

Defining APPC links

An APPC 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** that enables it to be mapped to a VTAM logmode name and from there to a class of service (COS). A set of APPC sessions is therefore referred to as a **modeset**.

Note: An APPC terminal is often an APPC system that supports only a single session and which does not support an LU services manager. There are several ways of defining such terminals; further details are given under "Defining single-session APPC terminals" on page 105. This section describes the definition of one or more modesets containing more than one session.

To define an APPC link to a remote system, you must:

1. Use DEFINE CONNECTION to define the remote system.
2. Use DEFINE SESSIONS to define each set of sessions to the remote system.

However, you must not have more than one APPC connection installed at the same time between an LU-LU pair. Nor should you have an APPC and an LUTYPE6.1 connection installed at the same time between an LU-LU pair.

For all APPC links, except single-session links to APPC 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 cannot be used for any of the sets that you define.

If you are defining a VTAM logon mode table, remember to include an entry for the SNASVCMG sessions. (See "VTAM LOGMODE table entries for CICS" on page 86.)

Defining the remote APPC system

The form of definition for an APPC system is shown in Figure 30 on page 102.

```
DEFINE
  CONNECTION(name)
  GROUP(groupname)
  NETNAME(name)
  ACCESSMETHOD(VTAM)
  PROTOCOL(APPC)
  SINGLESESS(NO)
  QUEUELIMIT(NO|0-9999)
  MAXQTIME(NO|0-9999)
  AUTOCONNECT(NO|YES|ALL)
  SECURITYNAME(value)
  ATTACHSEC(LOCAL|IDENTIFY|VERIFY|PERSISTENT|MIXIDPE)
  BINDPASSWORD(password)
  BINDSECURITY(YES|NO)
  USEDFLTUSER(NO|YES)
  PSRECOVERY(SYSDEFAULT|NONE)
For LUTYPE6.1 on APPC
  DATASTREAM(USER|3270|SCS|STRFIELD|LMS)
  RECORDFORMAT(U|VB)
```

Figure 30. Defining an APPC system

You must specify ACCESSMETHOD(VTAM) and PROTOCOL(APPC) to define an APPC system. The CONNECTION name (that is, the sysidnt) and the netname have the meanings explained in "Identifying remote systems" on page 94.

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 105.)

The AUTOCONNECT option specifies which of the sessions associated with the connection are to be bound when CICS is initialized. Further information is given in "The AUTOCONNECT option" on page 107.

The QUEUELIMIT option specifies the maximum number of requests permitted to queue for free sessions to the remote system. The MAXQTIME option specifies the maximum time between a queue becoming full and it being purged because the remote system is unresponsive. Further information is given in Chapter 23, "Intersystem session queue management" on page 221.

If you are using VTAM persistent session support, the PSRECOVERY option specifies whether sessions to the remote system are recovered, if the local CICS fails and restarts within the persistent session delay interval. Further information is given in "Using VTAM persistent sessions on APPC links" on page 109.

Note: 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 options to specify data stream information for asynchronous processing. The information provided by these options is not used by APPC application programs.

Defining groups of APPC sessions

Each group of sessions for an APPC system is defined by means of a DEFINE SESSIONS command. The definition is shown in Figure 31.

Each individual group of sessions is referred to as a **modeset**.

```
DEFINE
   SESSIONS(csdname)
   GROUP(groupname)
   CONNECTION(name)
   MODENAME(name)
   PROTOCOL(APPC)
   MAXIMUM(m1,m2)
   SENDSIZE(size)
   RECEIVESIZE(size)
   SESSPRIORITY(number)
   AUTOCONNECT(NO|YES|ALL)
   USERAREALEN(value)
   RECOVOPTION(SYSDEFAULT|CLEARCONV|RELEASESESS|UNCONDREL|NONE)
```

Figure 31. Defining a group of APPC sessions

The CONNECTION option specifies the name (1–4 characters) of the APPC system for which the group is being defined; that is, the CONNECTION name in the associated DEFINE CONNECTION command.

The MODENAME option enables you to specify a name (1–8 characters) to identify this group of related sessions. The name must be unique among the modenames for any one APPC intersystem link, and you must not use the reserved name SNASVCMG.

The MAXIMUM(m1,m2) option specifies the maximum number of sessions that are to be supported for the group. The parameters of this option 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 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 RECEIVESIZE option, which specifies the maximum size of request unit (RU) to be received, must be in the range 256 through 30 720.

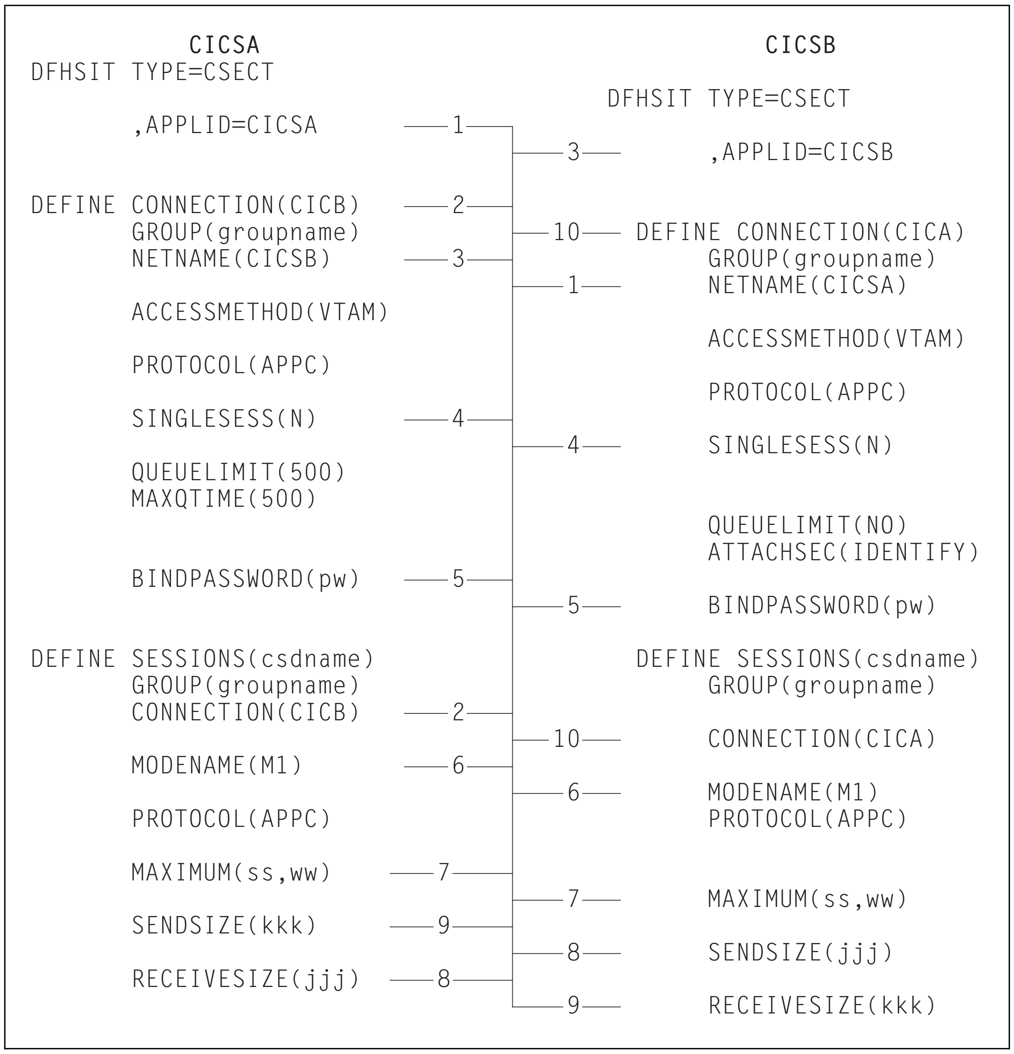
The AUTOCONNECT option specifies whether the sessions are to be bound when CICS is initialized. Further information is given in “The AUTOCONNECT option” on page 107.

If you are using VTAM persistent session support, and CICS fails and restarts within the persistent session delay interval, the RECOVOPTION option specifies how CICS recovers the sessions. (The RECOVNOTIFY option does not apply to APPC sessions.) Further information is given in “Using VTAM persistent sessions on APPC links” on page 109.

Defining compatible CICS APPC nodes

When you are defining an APPC 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 32.



```
            CICSA                                          CICSB
DFHSIT TYPE=CSECT
                                               DFHSIT TYPE=CSECT
       ,APPLID=CICSA      ──1──┐
                               ├──3──       ,APPLID=CICSB

DEFINE CONNECTION(CICB)   ──2──┤
       GROUP(groupname)        ├──10── DEFINE CONNECTION(CICA)
       NETNAME(CICSB)     ──3──┤              GROUP(groupname)
                               ├──1──         NETNAME(CICSA)
       ACCESSMETHOD(VTAM)      │
                               │              ACCESSMETHOD(VTAM)
       PROTOCOL(APPC)          │
                               │              PROTOCOL(APPC)
       SINGLESESS(N)      ──4──┤
                               ├──4──         SINGLESESS(N)
       QUEUELIMIT(500)         │
       MAXQTIME(500)           │
                               │              QUEUELIMIT(NO)
                               │              ATTACHSEC(IDENTIFY)
       BINDPASSWORD(pw)   ──5──┤
                               ├──5──         BINDPASSWORD(pw)

DEFINE SESSIONS(csdname)       │       DEFINE SESSIONS(csdname)
       GROUP(groupname)        │              GROUP(groupname)
       CONNECTION(CICB)   ──2──┤
                               ├──10──        CONNECTION(CICA)
       MODENAME(M1)       ──6──┤
                               ├──6──         MODENAME(M1)
       PROTOCOL(APPC)          │              PROTOCOL(APPC)

       MAXIMUM(ss,ww)    ──7───┤
                               ├──7──         MAXIMUM(ss,ww)
       SENDSIZE(kkk)     ──9───┤
                               ├──8──         SENDSIZE(jjj)
       RECEIVESIZE(jjj)  ──8───┤
                               └──9──         RECEIVESIZE(kkk)
```

Figure 32. Defining compatible CICS APPC ISC nodes

In Figure 32, related options and operands are shown by the numbered paths, all of which pass through the central connecting line.

Notes:

1. The values specified for MAXIMUM on either side of the link need not match, because they are negotiated by the LU services managers. However, a matching specification avoids unusable TCTTE entries, and also avoids unexpected bidding because of the “contention winners” negotiation.
2. If the value specified for SENDSIZE on one side of the link does not match that specified for RECEIVESIZE on the other, CICS negotiates the values at BIND time.

Automatic installation of APPC links

You can use the CICS autoinstall facility to allow APPC links to be defined dynamically on their first usage, thereby saving on storage for installed definitions, and on time spent creating the definitions.

Note: The method described here applies only to APPC parallel-session and single-session links initiated by BIND requests. The method to be used for APPC single-session links initiated by VTAM CINIT requests is described in "Defining single-session APPC terminals." You cannot autoinstall APPC parallel-session links initiated by CINIT requests.

If autoinstall is enabled, and an APPC BIND request is received for an APPC service manager (SNASVCMG) session (or for the only session of a single-session connection), and there is no matching CICS CONNECTION definition, a new connection is created and installed automatically.

Like autoinstall for terminals, autoinstall for APPC links requires model definitions. However, unlike the model definitions used to autoinstall terminals, those used to autoinstall APPC links do not need to be defined explicitly as models. Instead, CICS can use any previously-installed link definition as a "template" for a new definition. In order for autoinstall to work, you must have a template for each kind of link you want to be autoinstalled.

The purpose of a template is to provide CICS with a definition that can be used for all connections with the same properties. You customize the supplied autoinstall user program, DFHZATDY, to select an appropriate template for each new link, based on the information it receives from VTAM.

A template consists of a CONNECTION definition and its associated SESSIONS definitions. You should have a definition installed for each different set of session properties you are going to need.

Any installed link definition can be used as a template but, for performance reasons, your template should be an installed link definition that you do not actually use. The definition is locked while CICS is copying it, and if you have a very large number of sessions being autoinstalled, the delay may be noticeable.

Autoinstall support is likely to be beneficial if you have large numbers of APPC parallel session devices with identical characteristics. For example, if you had 1000 PS/2s, all with the same characteristics, you would set up one template to autoinstall all of them. If 500 of your PS/2s had one set of characteristics, and 500 had another set, you would set up two templates to autoinstall them.

For further information about using autoinstall with APPC links, see the *CICS Resource Definition Guide*. For programming information about the autoinstall user program, see the *CICS Customization Guide*.

Defining single-session APPC terminals

There are two methods available for defining a single-session APPC terminal: you can define a CONNECTION-SESSIONS pair, with SINGLESESS(Y) specified for the connection; or you can define a TERMINAL-TYPETERM pair.

Defining an APPC terminal – method 1

You can define a CONNECTION-SESSIONS pair to represent a single-session APPC terminal.

The forms of DEFINE CONNECTION and DEFINE SESSIONS commands that are required are similar to those shown in Figure 30 on page 102 and Figure 31 on page 103. The differences are shown below:

```
DEFINE CONNECTION(sysidnt)
       .
       SINGLESESS(Y)
       .
DEFINE SESSIONS(csdname)
       .
       MAXIMUM(1,0)
       .
```

You must specify SINGLESESS(Y) for the connection. The MAXIMUM option must specify only one session. The second value has no meaning for a single session definition as CICS always binds as a contention winner. However, CICS accepts a negotiated bind or a negotiated bind response in which it is changed to the contention loser.

Defining an APPC terminal – method 2

You can define a single-session 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 single-session terminals.

   Autoinstall for APPC single sessions initiated by a VTAM CINIT works in the same way as autoinstall for other terminals, in that you must supply a TERMINAL—TYPETERM model pair. For further information about using autoinstall with APPC single-session terminals, see the *CICS Resource Definition Guide*.

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 APPC devices are seen as systems by CICS, the name in the TERMINAL option 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 accepts a negotiated bind resulting in CICS becoming the 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 need the model terminal definition (LU62) that is provided in the CICS-supplied CSD group DFHTERM. You also have to write an autoinstall user program, and provide suitable VTAM LOGMODE entries.

For further information about TERMINAL and TYPETERM definition, the CICS-supplied CSD groups, and automatic installation, see the *CICS Resource Definition Guide*. For guidance about VTAM LOGMODE entries, and for programming information about the autoinstall user program, see the *CICS Customization Guide*.

The AUTOCONNECT option

You can use the AUTOCONNECT option of DEFINE CONNECTION and DEFINE SESSIONS (and of DEFINE TYPETERM for APPC terminals) to control CICS attempts to establish communication with the remote APPC system.

Except for single-session APPC terminals (see "Defining single-session APPC terminals" on page 105), two events are necessary to establish sessions to a remote APPC system.

1. The connection to the remote system must be established. This 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 option of the DEFINE CONNECTION command, and in part by the AUTOCONNECT of the DEFINE SESSIONS command.

The AUTOCONNECT option of DEFINE CONNECTION

On the DEFINE CONNECTION command, the AUTOCONNECT option specifies whether CICS is to try to bind the LU services manager sessions at the earliest opportunity (when the VTAM ACB is opened). It can have the following values:

AUTOCONNECT(NO)
specifies that CICS **is not** to try to bind the LU services manager sessions.

AUTOCONNECT(YES)
specifies that CICS **is** to try 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 is to specify ALL.

The LU services manager sessions cannot, of course, 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, 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 option of DEFINE SESSIONS

On the DEFINE SESSIONS command, the AUTOCONNECT option 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 option can have the following values:

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. By specifying AUTOCONNECT(ALL), you may cause CICS to bind a number of contention winners other than the number originally specified in this system. The number of contention winners that CICS binds depends on the reply that the partner system gives to the request to initiate sessions (CNOS exchange). CICS will try to bind as contention winners *all* sessions that are not designated as contention losers in the CNOS reply. For example, suppose that you define a modegroup with DEFINE SESSIONS MAXIMUM(10,4) on this system and DEFINE SESSIONS MAXIMUM(10,2) on the remote system. If the sessions are acquired from this system, and the contention-losing sessions bind successfully, the result is eight primary contention-winning sessions.

Warning: 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 and made available by means of a CEMT SET MODENAME ACQUIRED AVAILABLE command. (For details of the CEMT SET MODENAME command, see the *CICS-Supplied Transactions* manual.) 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 tries to bind the single session or not.

Using VTAM persistent sessions on APPC links

You can use VTAM persistent sessions to improve the availability of APPC links. After a failed CICS has been restarted, CICS persistent session support enables sessions to be recovered without the need for network flows. CICS determines for how long the sessions should be retained from the PSDINT system initialization parameter. Thus, for persistent session support you must specify a PSDINT value greater than zero (and, on the XRF system initialization parameter, a value of `'NO'`—persistent session support is incompatible with XRF). If a failed CICS is restarted within the PSDINT interval, it can use the retained sessions immediately—there is no need for network flows to rebind them. The interval can be changed using the CEMT SET VTAM command, or the EXEC CICS SET VTAM command.

If CICS is terminated through CEMT PERFORM SHUTDOWN IMMEDIATE, or if it fails, sessions are placed in "recovery pending" state. During emergency restart, CICS restores APPC sessions that are defined as persistent to an "in session" state.

The PSRECOVERY option of DEFINE CONNECTION

In a CICS region running with persistent session support, you use this to specify *whether* the APPC sessions used by this connection are recovered on system restart within the persistent session delay interval. It can have the following values:

SYSDEFAULT
If a failed CICS system is restarted within the persistent session delay interval, the following actions occur:

- User modegroups are recovered to the SESSIONS RECOVOPTION value.
- The SNASVCMG modegroup is recovered.
- The connection is returned in ACQUIRED state and the last negotiated CNOS state is returned.

NONE
All sessions are unbound as out-of-service with no CNOS recovery.

The RECOVOPTION option of DEFINE SESSIONS and DEFINE TYPETERM

In a CICS region running with persistent session support, the RECOVOPTION option of DEFINE SESSIONS specifies *how* APPC sessions are to be recovered, after a system restart within the persistent session delay interval.

If you want the sessions to be persistent, you should allow the value to default to SYSDEFAULT. This specifies that CICS is to select the optimum procedure to recover a session on system restart within the persistent delay interval.

For a single-session APPC terminal, the RECOVOPTION option of DEFINE SESSIONS or DEFINE TYPETERM specifies how the terminal is to be returned to service after a system restart within the persistent session delay interval.

Without persistent session support, if AUTOCONNECT(YES) is specified for a terminal, the end-user must wait until the GMTRAN transaction has run before being able to continue working. If AUTOCONNECT(NO) is specified, the user has no way of knowing (unless told by support staff) when CICS is operational again unless he or she tries to log on. In either case, the user is disconnected from CICS and needs to reestablish his session, to regain his working environment. With persistent session support, the session is put into recovery pending state on a CICS failure. If CICS starts within the specified interval, and RECOVOPTION is set to SYSDEFAULT, the user does not need to reestablish his session to regain his working environment.

For definitive information about the SYSDEFAULT value, and about the other possible values of RECOVOPTION, see the *CICS Resource Definition Guide*.

For further information about CICS support for persistent sessions, see Chapter 27, "Intercommunication and VTAM persistent sessions" on page 253.

Defining logical unit type 6.1 links

LUTYPE6.1 links are necessary for intersystem communication between CICS and any system, such as IMS, that supports LUTYPE6.1 protocols but not APPC protocols.

You must not have an LUTYPE6.1 and an APPC connection installed at the same time between an LU-LU pair.

You can also, of course, define LUTYPE6.1 links between CICS systems. However, you are advised to use MRO or APPC links for CICS-to-CICS communication whenever possible.

A DEFINE CONNECTION is always required to define the remote system on an LUTYPE6.1 link. 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, that 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.

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 an intersystem session to the remote system, it always tries to allocate a contention winner. Only if no contention winners are available does it select a contention loser. It then has 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.

The definition for an LUTYPE6.1 link is shown in Figure 33 on page 112.

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:

- The CONNECTION and SESSIONS must be in the same GROUP.
- Both the CONNECTION and the SESSIONS must have PROTOCOL(LU61).
- On the CONNECTION definition, you must specify ATTACHSEC(LOCAL).
- On the SESSIONS definition, the value of the CONNECTION option must match the sysidnt specified on the CONNECTION definition.

On the SESSIONS definition, you must specify the number of SEND and RECEIVE sessions that are required (at least one of each). You must also specify the prefixes which allow the sessions to be named. A prefix is a 1-character or 2-character string that is used to generate session identifiers (TRMIDNTs). Do not use the characters '>' or '<', because these are the default prefixes for MRO sessions.

The prefixes and the number of sessions are specified separately. The combination of the prefix and the number of sessions must not exceed four characters. For example:

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

generates 10 receive sessions with identifiers RR01 through RR10.

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

generates 150 receive sessions with identifiers R001 through R150.

The AUTOCONNECT and INSERVICE options

The AUTOCONNECT option 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 be specified on the DEFINE CONNECTION command.

INSERVICE(NO) on the DEFINE CONNECTION command initializes the sessions in an 'out of service' state only if AUTOCONNECT(NO) is specified on the corresponding 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.

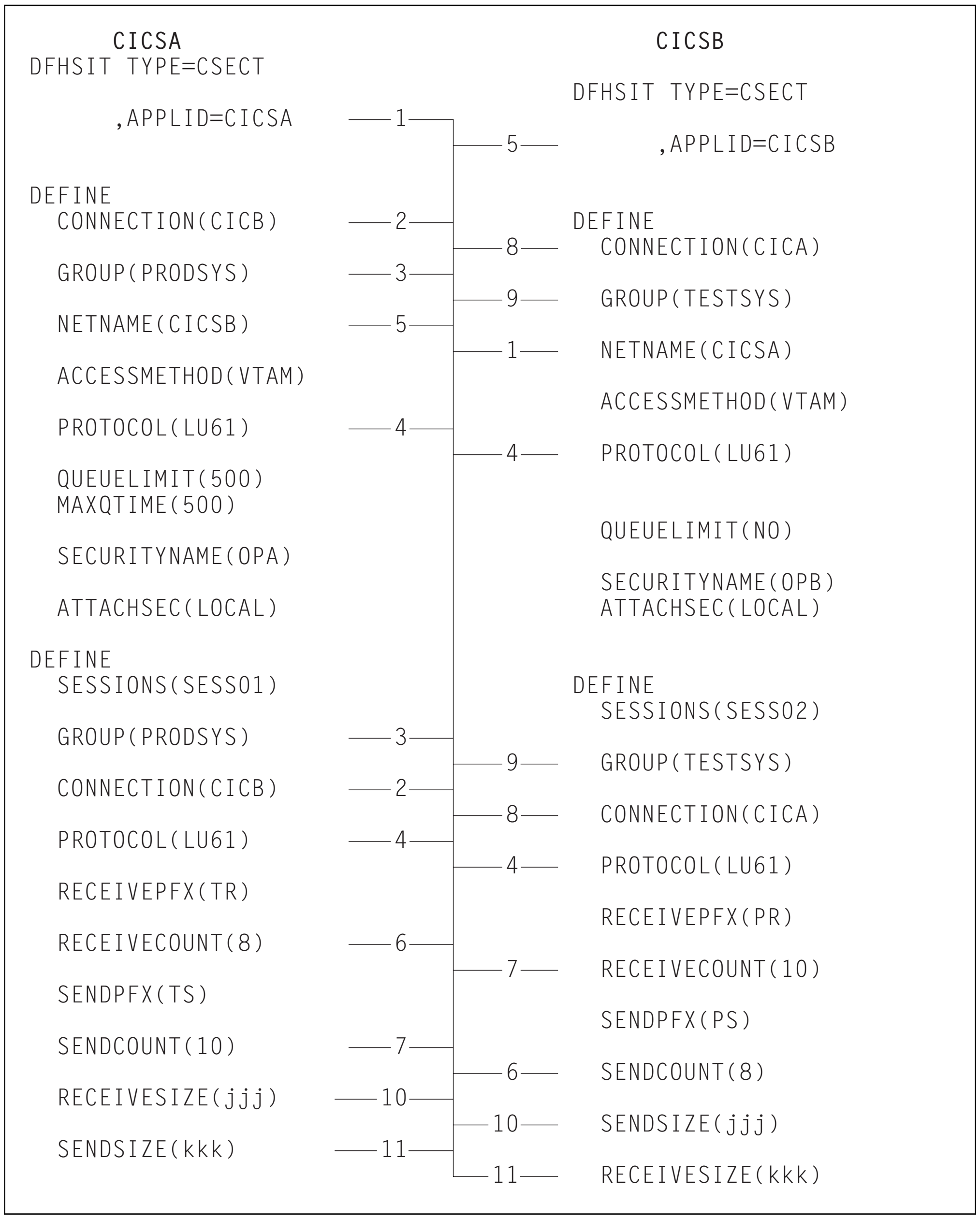
```
DEFINE
  CONNECTION(sysidnt)
  GROUP(groupname)
  NETNAME(name)
  ACCESSMETHOD(VTAM)
  PROTOCOL(LU61)
  DATASTREAM(USER|3270|SCS|STRFIELD|LMS)
  RECORDFORMAT(U|VB)
  QUEUELIMIT(NO|0-9999)
  MAXQTIME(NO|0-9999)
  INSERVICE(YES)
  SECURITYNAME(name)
  ATTACHSEC(LOCAL)
DEFINE
  SESSIONS(csdname)
  GROUP(groupname)
  CONNECTION(sysidnt)
  PROTOCOL(LU61)
  RECEIVEPFX(prefix1)
  RECEIVECOUNT(number1)
  SENDPFX(prefix2)
  SENDCOUNT(number2)
  SENDSIZE(size)
  RECEIVESIZE(size)
  SESSPRIORITY(number)
  AUTOCONNECT(NO|YES|ALL)
  INSERVICE(YES)
  BUILDCHAIN(YES)
  IOAREALEN(value)
```

Figure 33. 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 compatibility requirements are shown in Figure 34.



```
      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)
  NETNAME(CICSB)        ──5──┤
                             ├──1──             NETNAME(CICSA)
  ACCESSMETHOD(VTAM)         │
                             │                  ACCESSMETHOD(VTAM)
  PROTOCOL(LU61)        ──4──┤
                             ├──4──             PROTOCOL(LU61)
  QUEUELIMIT(500)            │
  MAXQTIME(500)              │
                             │                  QUEUELIMIT(NO)
  SECURITYNAME(OPA)          │
                             │                  SECURITYNAME(OPB)
  ATTACHSEC(LOCAL)           │                  ATTACHSEC(LOCAL)

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)     ──10──┤
                             ├──10──            SENDSIZE(jjj)
  SENDSIZE(kkk)        ──11──┤
                             └──11──            RECEIVESIZE(kkk)
```

Figure 34. Defining compatible CICS LUTYPE6.1 ISC nodes

In Figure 34, related attributes are shown by the numbered paths, all of which pass through the central connecting line.

Note: If the value specified for SENDSIZE on one side of the link does not match that specified for RECEIVESIZE on the other, CICS negotiates the values at BIND time.

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.

Note: This method can also be used for defining CICS-to-CICS links if you require sessions of differing characteristics. However, you are advised to use APPC links for CICS-to-CICS communication whenever possible.

The form of definition for individual LUTYPE6.1 sessions is shown in Figure 35.

```
DEFINE
  CONNECTION(sysidnt)
  GROUP(groupname)
  NETNAME(name)
  ACCESSMETHOD(VTAM)
  PROTOCOL(LU61)
  DATASTREAM(USER|3270|SCS|STRFIELD|LMS)
  RECORDFORMAT(U|VB)
  QUEUELIMIT(NO|0-9999)
  MAXQTIME(NO|0-9999)
  INSERVICE(YES)
  SECURITYNAME(name)
  ATTACHSEC(LOCAL)
Each individual session is then defined as follows:
DEFINE
  SESSIONS(csdname)
  GROUP(groupname)
  CONNECTION(sysidnt)
  SESSNAME(name)
  NETNAMEQ(name)
  PROTOCOL(LU61)
  RECEIVECOUNT(1|0)
  SENDCOUNT(0|1)
  SENDSIZE(size)
  RECEIVESIZE(size)
  SESSPRIORITY(number)
  AUTOCONNECT(NO|YES|ALL)
  BUILDCHAIN(YES)
  IOAREALEN(value)
```

Figure 35. Defining an LUTYPE6.1 link with individual sessions

Defining compatible CICS and IMS nodes

This section describes the writing of suitable CICS definitions that are compatible with the corresponding IMS definitions.

An overview of IMS system definition is given in Chapter 11, “Installation considerations for intersystem communication” on page 85. The relationships between CICS and IMS definitions are summarized in Figure 36 on page 118.

System names

The network name of the CICS system (its applid) is specified on the APPLID CICS system initialization parameter. This name must be specified on the NAME operand of the IMS TERMINAL macro that defines the CICS system. For CICS systems that use XRF, the name will be the CICS generic applid. For non-XRF CICS systems, the name will be the single applid specified on the APPLID SIT parameter (see "Generic and specific applids for XRF" on page 125).

The network name of the IMS system may be specified in various ways:

- For systems with XRF support, as the USERVAR that is defined in the DFSHSBxx member of IMS.PROCLIB.
- For systems without XRF:
  - On the APPLID operand of the IMS COMM macro
  - As a label on the EXEC statement of the IMS startup job (if APPLID is coded as NONE)
  - As a started task name (if APPLID is coded as NONE)

You must specify the network name of the IMS system on the NETNAME option of the CICS DEFINE CONNECTION command that defines the IMS system.

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-to-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 option 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 option 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 options of the DEFINE CONNECTION and DEFINE SESSIONS commands that are of significance for CICS-to-IMS sessions.

ATTACHSEC
: Must be specified as LOCAL.

BUILDCHAIN(YES)
Specifies that multiple RU chains are to be assembled before being passed to the application program. A complete chain is passed to the application program in response to each RECEIVE command, and the application performs any required deblocking.

BUILDCHAIN(YES) must be specified (or allowed to default) for LUTYPE6.1 sessions.

DATASTREAM(USER)
Must be specified with the value USER or allowed to default.

This option is used only when CICS is communicating with IMS by using the EXEC CICS 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.

QUEUELIMIT(NO|0-9999)
Specifies the maximum number of requests permitted to queue for free sessions to the remote system. Further information is given in Chapter 23, "Intersystem session queue management" on page 221.

MAXQTIME(NO|0-9999)
Specifies the maximum time, in seconds, between the queue for sessions to the remote system becoming full (that is, reaching the limit specified on QUEUELIMIT) and the queue being purged because the remote system is unresponsive. Further information is given in Chapter 23, "Intersystem session queue management" on page 221.

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:

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

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 22, "CICS-to-IMS applications" on page 197.

SENDCOUNT and RECEIVECOUNT

Used to specify whether the session is a SEND session or a RECEIVE session.

A SEND session is one in which the local CICS is the secondary and is the contention winner. Specify:

```
SENDCOUNT(1)
```

Allow `RECEIVECOUNT` to default. Do *not* specify `RECEIVECOUNT(0)`.

A RECEIVE session is one in which the local CICS is the primary and is the contention loser. Specify:

```
RECEIVECOUNT(1)
```

Allow `SENDCOUNT` to default. Do *not* specify `SENDCOUNT(0)`.

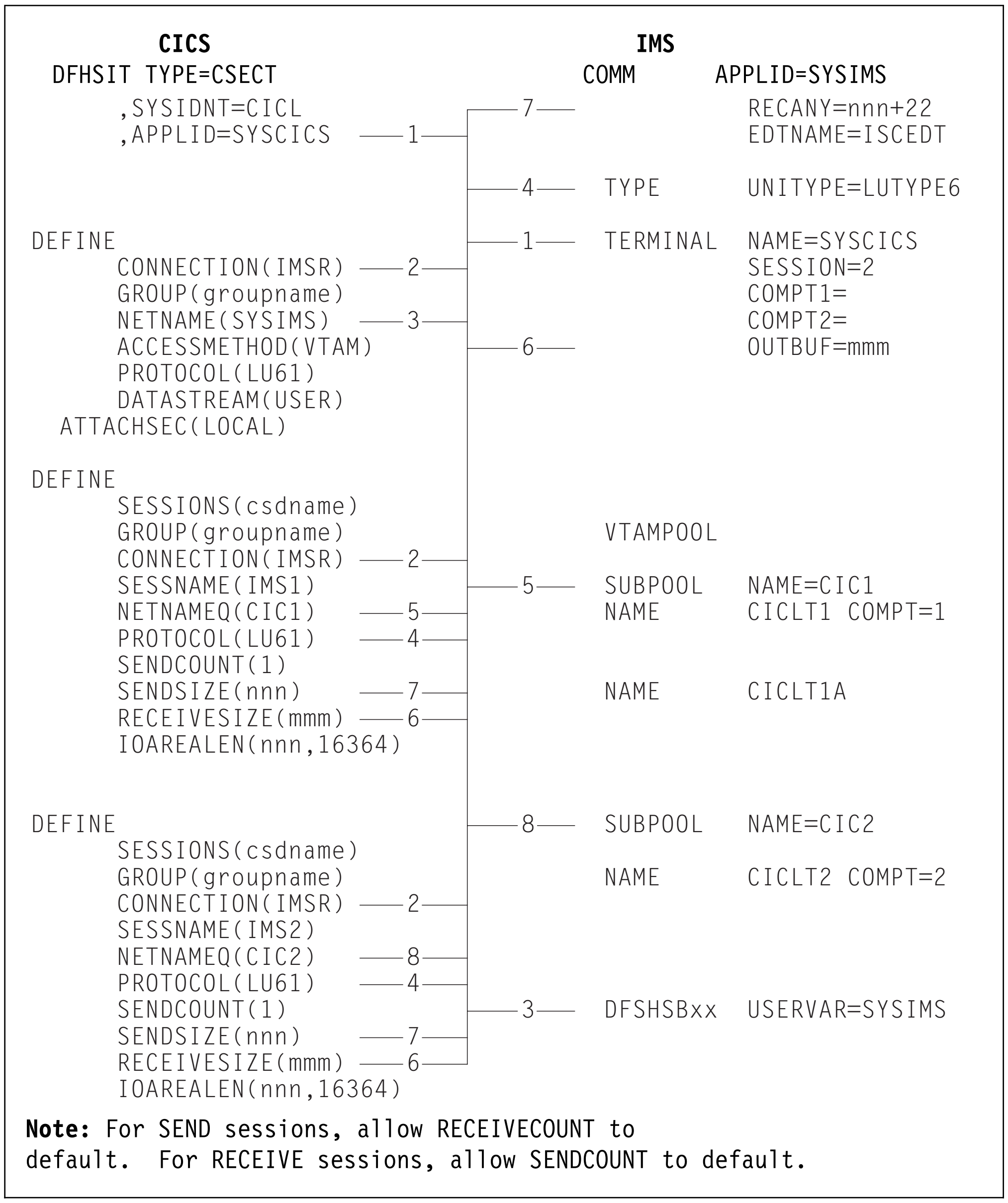
SEND sessions are recommended for all CICS-to-IMS sessions.

You need not specify a SENDPFX or a RECEIVEPFX; the name of the session is taken from the SESSNAME option.

SENDSIZE

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:

- Not less than 256 bytes
- At least the value in the RECANY parameter minus 22 bytes



```
        CICS                               IMS
  DFHSIT TYPE=CSECT                     COMM     APPLID=SYSIMS
      ,SYSIDNT=CICL             ┌──7──             RECANY=nnn+22
      ,APPLID=SYSCICS   ──1─────┤                  EDTNAME=ISCEDT
                                │
                                ├──4──  TYPE      UNITYPE=LUTYPE6
                                │
DEFINE                          ├──1──  TERMINAL  NAME=SYSCICS
      CONNECTION(IMSR)  ──2─────┤                 SESSION=2
      GROUP(groupname)          │                 COMPT1=
      NETNAME(SYSIMS)   ──3─────┤                 COMPT2=
      ACCESSMETHOD(VTAM)        ├──6──            OUTBUF=mmm
      PROTOCOL(LU61)            │
      DATASTREAM(USER)          │
  ATTACHSEC(LOCAL)              │
                                │
DEFINE                          │
      SESSIONS(csdname)         │
      GROUP(groupname)          │       VTAMPOOL
      CONNECTION(IMSR)  ──2─────┤
      SESSNAME(IMS1)            ├──5──  SUBPOOL   NAME=CIC1
      NETNAMEQ(CIC1)    ──5─────┤       NAME      CICLT1 COMPT=1
      PROTOCOL(LU61)    ──4─────┤
      SENDCOUNT(1)              │
      SENDSIZE(nnn)     ──7─────┤       NAME      CICLT1A
      RECEIVESIZE(mmm)  ──6─────┤
      IOAREALEN(nnn,16364)      │
                                │
DEFINE                          ├──8──  SUBPOOL   NAME=CIC2
      SESSIONS(csdname)         │
      GROUP(groupname)          │       NAME      CICLT2 COMPT=2
      CONNECTION(IMSR)  ──2─────┤
      SESSNAME(IMS2)            │
      NETNAMEQ(CIC2)    ──8─────┤
      PROTOCOL(LU61)    ──4─────┤
      SENDCOUNT(1)              ├──3──  DFSHSBxx  USERVAR=SYSIMS
      SENDSIZE(nnn)     ──7─────┤
      RECEIVESIZE(mmm)  ──6─────┘
      IOAREALEN(nnn,16364)
```

Note: For SEND sessions, allow RECEIVECOUNT to default. For RECEIVE sessions, allow SENDCOUNT to default.

Figure 36. Defining compatible CICS and IMS nodes

Figure 36 shows the relationship between the CICS and IMS definitions of an intersystem link. Related options and operands are shown by the numbered paths, all of which pass through the central connecting line. Note that DFSHSBxx is in IMSVS.PROCLIB, and that, for non-XRF IMS systems, NETNAME should match the APPLID from the IMS COMM macro.

Note: For an example of a VTAM logmode table entry for IMS, see Figure 24 on page 87.

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 creating two or more CONNECTION definitions (with their associated SESSION definitions), with the same netname but with different sysidnts (Figure 37 on page 120). Although all the system definitions resolve to the same netname, and therefore to the same IMS system, the use of a sysidnt name in CICS causes 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. 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 is held ('busy') until the conversation is terminated.

2. For CICS-initiated asynchronous processing.

   CICS applications that use the START command can name the sysidnt of this group. CICS uses 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 shows the heaviest usage, and the frequency of usage decreases towards the last session in the group.

3. 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 uses 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 be queuing 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 this sort of queuing problem.

```
  DFHSIT TYPE=CSECT,
         SYSIDNT=CICL,
         APPLID=SYSCICS
CICS-initiated distributed transaction processing
  DEFINE CONNECTION(IMSA)
         NETNAME(SYSIMS)
         ACCESSMETHOD(VTAM)
  DEFINE SESSIONS(csdname)
         CONNECTION(IMSA)
         SESSNAME(IMS1)
         NETNAMEQ(DTP1)
         PROTOCOL(LU61)
  DEFINE SESSIONS(csdname)
         .
         .
CICS-initiated asynchronous processing
  DEFINE CONNECTION(IMSB)
         NETNAME(SYSIMS)
         ACCESSMETHOD(VTAM)
  DEFINE SESSIONS(csdname)
         CONNECTION(IMSB)
         SESSNAME(IMS1)
         NETNAMEQ(ASP1)
         PROTOCOL(LU61)
  DEFINE SESSIONS(csdname)
         .
         .
IMS-initiated asynchronous processing
  DEFINE CONNECTION(IMSC)
         NETNAME(SYSIMS)
         ACCESSMETHOD(VTAM)
  DEFINE SESSIONS(csdname)
         CONNECTION(IMSC)
         SESSNAME(IMS1)
         NETNAMEQ(IST1)
         PROTOCOL(LU61)
  DEFINE SESSIONS(csdname)
         .                               .
         .                               .
```

Figure 37. Defining multiple links to an IMS node

Indirect links for transaction routing

In releases prior to CICS Transaction Server for VSE/ESA Release 1, indirect links between CICS systems were required for transaction routing across intermediate systems. In a CICS Transaction Server for VSE/ESA Release 1 network (that is, a network in which all CICS systems are at release levels later than CICS/VSE 2.3), indirect links are only required if you are using non-VTAM terminals. Optionally, you can define them for use with VTAM terminals. In a mixed-release network, you must still define them, as before, on the pre-CICS Transaction Server for VSE/ESA Release 1 systems. Indirect links are never used for function-shipping, distributed program link, asynchronous processing, or distributed transaction processing.

The following figure shows the concept of an indirect link.

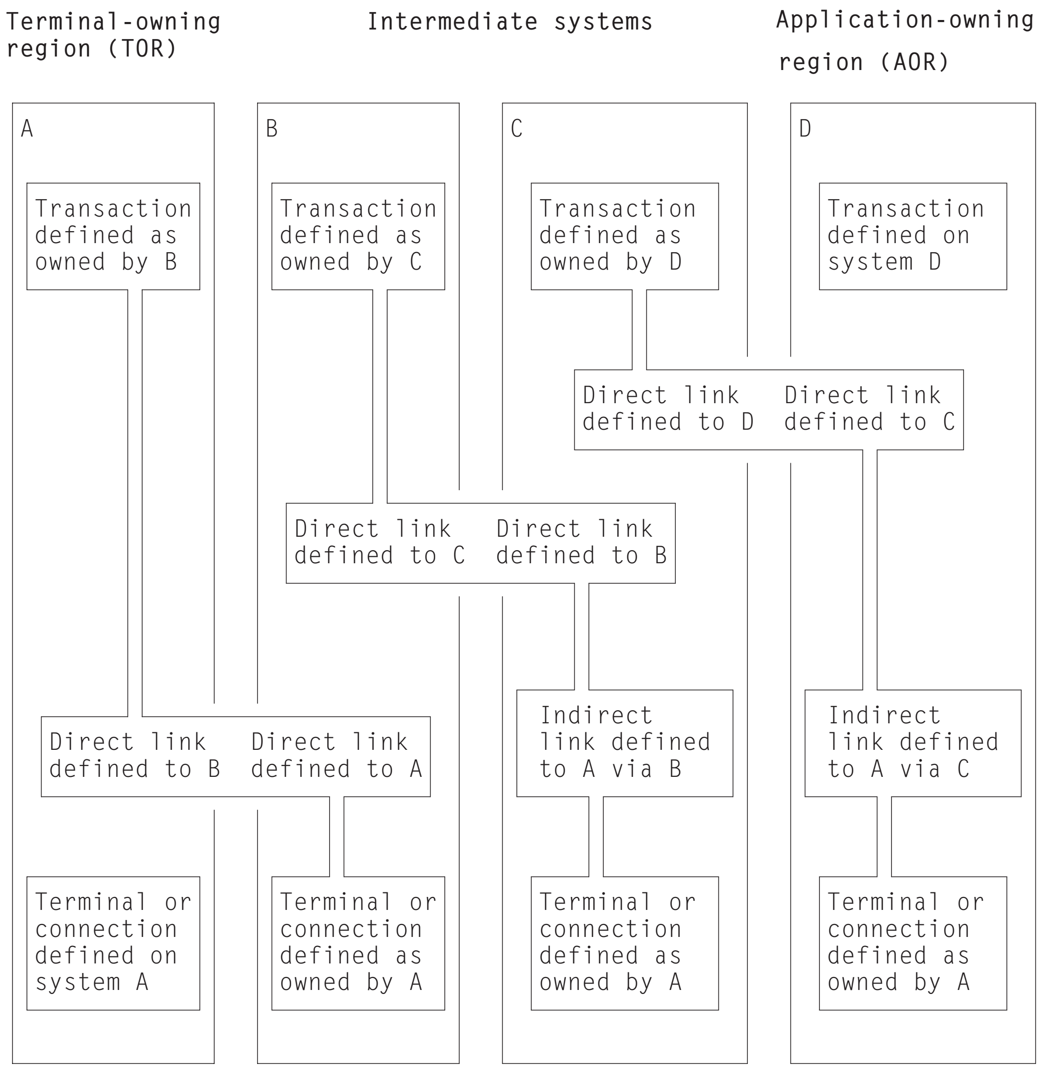

Terminal-owning region (TOR)
Intermediate systems
Application-owning region (AOR)
A
B
C
D
Transaction defined as owned by B
Transaction defined as owned by C
Transaction defined as owned by D
Transaction defined on system D
Direct link defined to D
Direct link defined to C
Direct link defined to C
Direct link defined to B
Indirect link defined to A via B
Indirect link defined to A via C
Direct link defined to B
Direct link defined to A
Terminal or connection defined on system A
Terminal or connection defined as owned by A
Terminal or connection defined as owned by A
Terminal or connection defined as owned by A

Figure 38. Indirect links for transaction routing

This figure illustrates a chain of systems (A, B, C, D) linked by MRO or APPC 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 system A and system D, but a path is available via the **intermediate** systems B and C.

To enable transaction-routing requests to pass along the path, resource definitions for both the terminal (which may be an APPC connection) and the transaction must be available in all four systems. The terminal is a local resource in the terminal-owning system A, and a remote resource in systems B, C, and D. Similarly, the transaction is a local resource in the transaction-owning system D, and a remote resource in the systems A, B, and C.

Why you may want to define indirect links in CICS Transaction Server for VSE/ESA Release 1

As explained in Chapter 14, "Defining remote resources" on page 137, CICS systems reference remote terminals by means of a unique identifier that is formed from:

- The applid (netname) of the terminal-owning region
- The identifier by which the terminal is known on the terminal-owning region

For CICS to form the fully-qualified terminal identifier, it must have access to the netname of the TOR. In earlier releases of CICS, an indirect link definition had two purposes. Where there was no direct link to the TOR, it:

1. Supplied the netname of the terminal-owning region.
2. Identified the **direct** link that was the start of the path to the terminal-owning region

Thus, in Figure 38 on page 121, the indirect link definition in system D provides the netname of system A and identifies system C as the next system in the path. Similarly, the indirect link definition in system C provides the netname of system A and identifies system B as the next system in the path. System B has a direct link to system A, and therefore does not require an indirect link.

In CICS Transaction Server for VSE/ESA Release 1, unless you are using non-VTAM terminals, indirect links are optional. Different considerations apply, depending on whether you are using shippable or hard-coded terminal definitions.

Shippable terminals

Indirect links are not necessary to allow terminal definitions to be shipped to an AOR across intermediate systems. Each shipped definition contains a pointer to the previous system in the transaction routing path (or to an indirect connection to the TOR, if one exists). This allows routed transactions to be attached, by identifying the netname of the TOR and the path from the AOR to the TOR.

If several paths are available, you can use indirect links to specify the preferred path to the TOR.

Note: Non-VTAM terminals are not shippable.

Hard-coded terminals

If you are using VTAM terminals exclusively, indirect links are not required. You use the REMOTESYSNET option of the TERMINAL definition (or the CONNECTION definition, if the "terminal" is an APPC device) to specify the netname of the TOR; and the REMOTESYSTEM option to specify the next system in the path to the TOR. If several paths are available, use REMOTESYSTEM to specify the next system in the preferred path.

If you are using non-VTAM terminals, indirect links are required. This is because you cannot use RDO to define non-VTAM terminals; the DFHTCT TYPE=REMOTE or TYPE=REGION macros used to create the remote definitions do not include an equivalent of the REMOTESYSNET option of CEDA DEFINE TERMINAL.

Thus, in CICS Transaction Server for VSE/ESA Release 1, you may decide to define indirect links:

- If you are using non-VTAM terminals for transaction routing across intermediate systems.
- To enable you to use existing remote terminal definitions that do not specify the REMOTESYSNET option. For example, you may have hundreds of remote VTAM terminals defined to a CICS/VSE 2.3 system. If you introduce a new CICS Transaction Server for VSE/ESA Release 1 back-end system into your network, you may want to copy the existing definitions to the CSD of the new system. If the structure of your network means that there is no direct link to the TOR, it may be quicker to define a single indirect link, rather than change all the copied definitions to include the REMOTESYSNET option.
- To specify the preferred path to the TOR, if more than one exists, and you are using shippable terminals.

Mixed-release networks

In a mixed-release network, you must continue to define indirect links on the pre-CICS Transaction Server for VSE/ESA Release 1 systems, as in CICS/VSE 2.3.

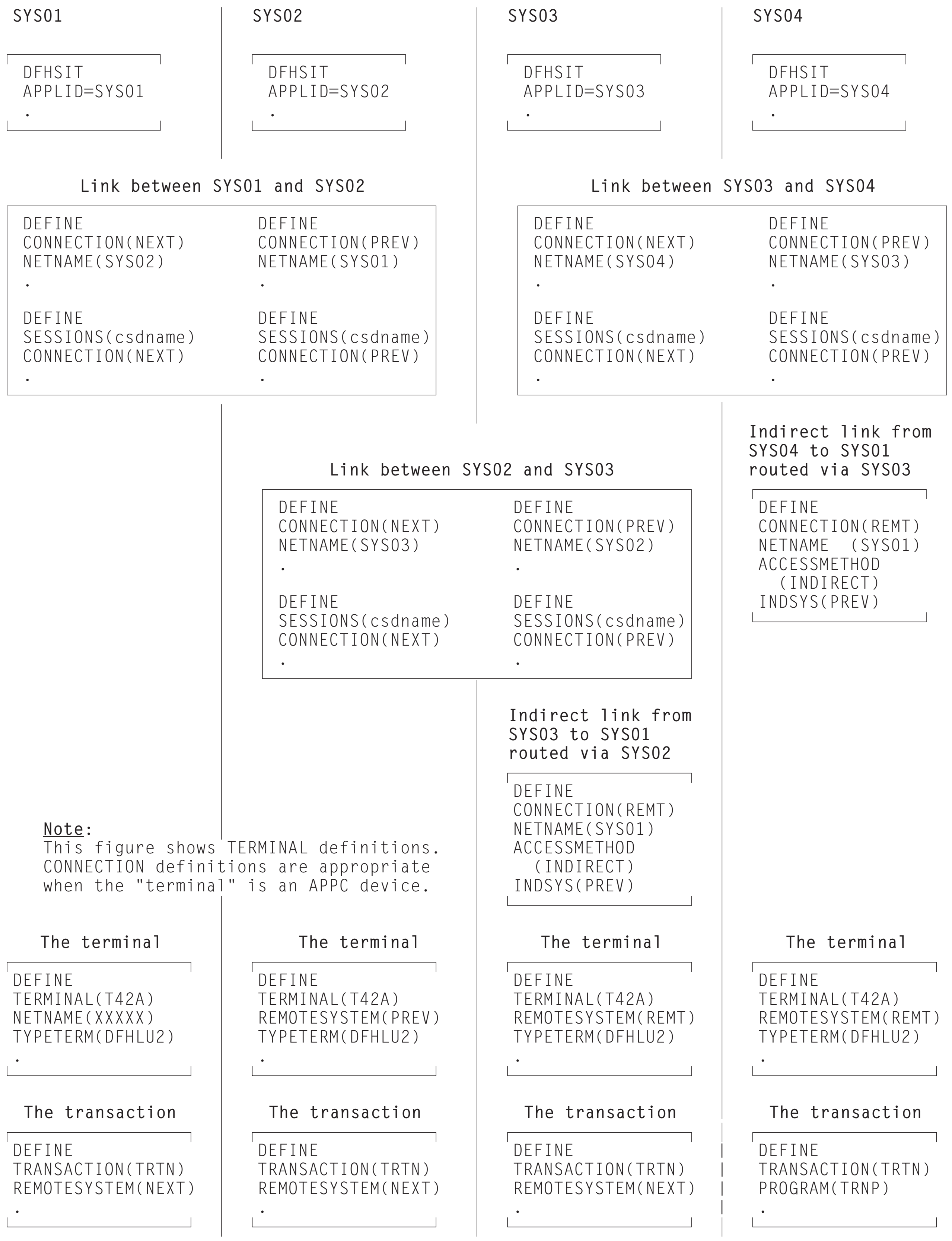
The INDSYS option of the indirect CONNECTION definition must name the direct link to the TOR.

Resource definition for transaction routing using indirect links

This section outlines the resource definitions required to establish a transaction-routing path between a terminal-owning region SYS01 and an application-owning region SYS04 via two intermediate systems SYS02 and SYS03, using indirect links.

The resource definitions required are shown in Figure 39 on page 124.

Note: For clarity, the figure shows hard-coded remote terminal definitions that do not use the REMOTESYSNET option (if REMOTESYSNET had been used, indirect links would not be required). Shippable terminals could equally well have been used.



```
SYS01                 SYS02                 SYS03                 SYS04

DFHSIT                DFHSIT                DFHSIT                DFHSIT
APPLID=SYS01          APPLID=SYS02          APPLID=SYS03          APPLID=SYS04
.                     .                     .                     .
```

```
Link between SYS01 and SYS02

DEFINE                  DEFINE
CONNECTION(NEXT)        CONNECTION(PREV)
NETNAME(SYS02)          NETNAME(SYS01)
.                       .

DEFINE                  DEFINE
SESSIONS(csdname)       SESSIONS(csdname)
CONNECTION(NEXT)        CONNECTION(PREV)
.                       .
```

```
Link between SYS03 and SYS04

DEFINE                  DEFINE
CONNECTION(NEXT)        CONNECTION(PREV)
NETNAME(SYS04)          NETNAME(SYS03)
.                       .

DEFINE                  DEFINE
SESSIONS(csdname)       SESSIONS(csdname)
CONNECTION(NEXT)        CONNECTION(PREV)
.                       .
```

```
Link between SYS02 and SYS03

DEFINE                  DEFINE
CONNECTION(NEXT)        CONNECTION(PREV)
NETNAME(SYS03)          NETNAME(SYS02)
.                       .

DEFINE                  DEFINE
SESSIONS(csdname)       SESSIONS(csdname)
CONNECTION(NEXT)        CONNECTION(PREV)
.                       .
```

```
Indirect link from
SYS04 to SYS01
routed via SYS03

DEFINE
CONNECTION(REMT)
NETNAME  (SYS01)
ACCESSMETHOD
  (INDIRECT)
INDSYS(PREV)
```

```
Indirect link from
SYS03 to SYS01
routed via SYS02

DEFINE
CONNECTION(REMT)
NETNAME(SYS01)
ACCESSMETHOD
  (INDIRECT)
INDSYS(PREV)
```

<u>Note</u>:
This figure shows TERMINAL definitions.
CONNECTION definitions are appropriate
when the "terminal" is an APPC device.

```
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. Because the remote terminal definitions in SYS04 and SYS03 do not specify the REMOTESYSNET option, indirect links are required.

Defining the direct links

The direct links between SYS01 and SYS02, SYS02 and SYS03, and SYS03 and SYS04 are MRO or APPC links defined as described earlier in this chapter.

Defining the indirect links

Indirect links to the TOR can be defined to some systems in a transaction-routing path and not to others, depending on the structure of your network and how you have coded your remote terminal definitions. For example, if one of the intermediate systems is a CICS/VSE 2.3 system that does not have a direct link to the TOR, an indirect link will be required. Indirect links are never required in the system to which the terminal-owning region has a direct link.

In the current example, indirect links are defined in SYS04 and SYS03. The following rules apply to the definition of an indirect link:

- ACCESSMETHOD must be INDIRECT.
- NETNAME must be the applid of the terminal-owning region.
- INDSYS (meaning indirect system) must name the CONNECTION name of an MRO or APPC link that is the start of the path to the terminal-owning region.
- No SESSIONS definition is required for the indirect connection; the sessions that are used are those of the direct link named in the INDSYS option.

Defining the terminal

The recommended methods for defining remote terminals and connections to a CICS Transaction Server for VSE/ESA Release 1 system are described in Chapter 14, “Defining remote resources” on page 137.

If shippable terminals are used, no remote terminal definitions are required.

Figure 39 on page 124 shows hard-coded remote terminal definitions that (as in CICS/VSE 2.3) do not specify the REMOTESYSNET option. If you use these:

- The REMOTESYSTEM (or SYSIDNT) option in the remote terminal or connection definition must always name a link to the TOR (that is, a CONNECTION definition on which NETNAME specifies 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 definition of remote transactions is described in Chapter 14, “Defining remote resources” on page 137.

Generic and specific applids for XRF

CICS systems that use XRF have *two* applid names: a **generic** name and a **specific** name. The names are specified on the APPLID=(generic-applid,specific-applid) system initialization parameter.

If you are using XRF, you must specify both names on the APPLID parameter. This is because the active and alternate CICS systems must have the same generic applid and different specific applids.

Remember that “generic” and “specific” applids apply only to systems that use XRF; CICS systems that don’t use XRF have only one applid.

Note: The active and alternate systems that have the same generic applid must also have the same sysidnt. For further information about generic and specific applids, see the *CICS XRF Guide*.

Chapter 13. Managing APPC links

This chapter shows you how to use the master terminal transaction, CEMT, to manage APPC connections. (The information is mainly about parallel-sessions connections between CICS systems.) The chapter shows how the action of the CEMT commands is affected by the way the connections have been defined to CICS. The commands used are:

- CEMT SET CONNECTION ACQUIRED|RELEASED
- CEMT SET MODENAME AVAILABLE|ACQUIRED|CLOSED

Detailed formats and options of CEMT commands are given in the *CICS-Supplied Transactions* manual.

Note: The operator commands controlling APPC 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 on the following pages 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 APPC 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.

The principles of operation described in these sections also apply to the EXEC CICS INQUIRE CONNECTION, INQUIRE MODENAME, SET CONNECTION, and SET MODENAME commands. For programming information about these commands, see the *CICS System Programming Reference* manual.

The rest of the chapter contains the following topics:

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:

- Establishing of the two LU services manager sessions in the modegroup SNASVCMG.
- Initiating 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 DFHZC4900 and DFHZC4901.

- Establishing of the sessions that carry CICS application data.

The following processes, also part of connection establishment, are described in Part 6, "Recovery and restart" on page 231:

- Exchanging lognames
- Resolving and reporting 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 option on the SESSIONS definition, or to satisfy allocate requests from applications.

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 option of the SESSIONS definition as described in the following section.

Effects of the AUTOCONNECT option

The meanings of the AUTOCONNECT option for APPC connections are described in "The AUTOCONNECT option" on page 107. The effect of the AUTOCONNECT option of the SESSIONS definition is to control the acquisition of sessions in modegroups associated with the connection. Each modegroup has its own AUTOCONNECT option and the setting of this option affects the sessions in the modegroup as described in Table 11.

Table 11. 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.

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.

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 been bound.

The ALLOCATE command, either as an explicit command in your application or as implied in automatic transaction initiation (ATI), cannot bind contention-loser sessions, although it can assign them to conversations if they are already bound.

Effects of the MAXIMUM option

The MAXIMUM option 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 APPC 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 effect 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 DFHZC4900 and DFHZC4901.

These results can also be deduced, as shown in Table 12 on page 130, by issuing a CEMT INQUIRE MODENAME command.

Table 12. Data displayed by INQ MODENAME

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 value of the MAXIMUM option.
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 APPC connection, without releasing or reacquiring the connection. The processes used 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 13 on page 131.

Table 13. 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. Queued session requests are honored before sessions are unbound.
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.

Command scope and restrictions

User modegroups, which are built from CEDA DEFINE SESSIONS (or equivalent macro) definitions, can be modified by using the SET MODENAME command or by overtyping the INQUIRE MODENAME display data.

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 display data, are suppressed. It is controlled by the SET CONNECTION command, or by overtyping the INQUIRE CONNECTION display data, which also affects 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, is suppressed.

You must use the SET|INQ CONNECTION|MODENAME to manage the status of user sessions and to control negotiation with remote systems.

A change to an APPC connection or modegroup can be requested by an operator issuing CEMT SET commands or by an application program issuing EXEC CICS SET commands. It is possible to issue one of these SET commands while a previous, perhaps contradictory, SET command is still in progress. This is particularly likely to occur in systems configured with large numbers of parallel sessions, in which the status of many sessions may be affected by an individual change to a connection or modegroup. Such overlapping SET commands can produce unpredictable results. You should therefore ensure that previously issued SET commands have fully completed before issuing the next SET command.

A similar situation can occur at startup if a SET CONNECTION or SET MODEGROUP command is issued while sessions are autoconnecting. You should therefore also ensure that all sessions have finished autoconnecting before issuing such a SET command.

Releasing the connection

The SET CONNECTION RELEASED command causes CICS to quiesce a connection and release all sessions associated with it. The major processes involved in this operation are:

- Executing 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 zero.
- Quiescing 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.”

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 (by, for example, a CEMT SET MODENAME AVAILABLE(0) command), ALLOCATE requests are rejected. Transaction routing and function shipping requests are also rejected. 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.

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).

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.

You can prevent your partner from binding the SNASVCMG sessions by invoking CEMT SET CONNECTION OUTSERVICE, which is ignored unless the connection is already in the RELEASED state.

To summarize, you can make a connection unavailable and retain it under your control by issuing these commands in the order shown:

```
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
```

Figure 40. Making the connection unavailable

Allocating from APPC mode groups with no available sessions

An application program can issue ALLOCATE commands for APPC sessions that can be satisfied in either of two ways:

1. Only by a session in a particular mode group
2. By a session in any mode group on the connection

An operator can issue CEMT SET MODENAME AVAILABLE(0) or CEMT SET MODENAME CLOSE to reduce the number of available sessions on an individual mode group to zero.

If an ALLOCATE for a particular mode group is issued when that mode group has no available sessions, the command is immediately rejected with the SYSIDERR condition.

If an ALLOCATE command is issued without specifying a particular mode group, and no mode groups on the connection have any sessions available, this command is immediately rejected with the SYSIDERR condition.

If a relevant mode group is still draining when an allocate request is received, the allocate is satisfied and added to the drain queue. An operator command to reduce the number of available sessions to zero does not complete until draining completes. In a very busy system allocating many sessions, this may mean that such modegroup operator commands take a long time to complete.

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 reestablish 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 41 on page 135 summarizes the effect of CEMT commands on the status of an APPC link.

Command scope and restrictions

User modesets, which are built from CEDA DEFINE SESSIONS definitions, may be modified by using the SET MODENAME command or by overtyping the INQUIRE MODENAME display data. The SNASVCMG modeset, on the other hand, is built from the CONNECTION definition and any attempts to modify its status with a SET or INQUIRE MODENAME command is suppressed. It is, however, controlled by the SET|INQ CONNECTION, which also affects the user modesets.

CEMT INQUIRE NETNAME, where the netname is the applid of the partner system, displays the status of all sessions associated with that link. Any attempt to alter the status of these sessions is suppressed. You must use SET|INQ CONNECTION|MODENAME to manage the status of user sessions and to control negotiation with remote systems. INQ NETNAME may also be useful in error diagnosis.

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	N	N	N	N	N	N	N	N	ALLOCATE requests suspended
	Y	Y	N	N	N	N	Y	N	Partner can renegotiate
	Y	Y	Y	Y	Y	Y	Y	Y	ALLOCATE rejected with SYSIDERR
	N	Y	Y	N	Y	Y	Y	Y	SNASVCMG sessions released
	-	Y	N	-	Y	N	Y	N	Partner can rebind SNASVCMG

Figure 41. Effect of CEMT commands on an operational APPC link

Chapter 14. Defining remote resources

This chapter contains guidance information about identifying and defining remote resources.

Note: For detailed information about the RDO commands and macros used to define CICS resources, see the *CICS Resource Definition Guide*.

The chapter contains the following topics:

Which remote resources need to be defined?

Remote resources are resources that reside on a remote system but which need to be accessed by the local CICS system. In general, you 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 online (RDO) or resource definition macros, depending on the resource type.

You may need to define remote resources for CICS function shipping, DPL, asynchronous processing (START command shipping), and transaction routing. No remote resource definition is required for distributed transaction processing, nor for link requests to CICS programs made via the external CICS interface[9].

The remote resources that can be defined are:

- Remote files (function shipping)
- Remote DL/I PSBs (function shipping)
- Remote transient data destinations (function shipping)
- Remote temporary storage queues (function shipping)
- Remote programs for distributed program link (DPL)
- Remote terminals (transaction routing)
- Remote APPC connections (transaction routing)
- Remote transactions (transaction routing and asynchronous processing)

All remote resources must, of course, also be defined on the systems that own them.

[9] But see “A note on “daisy-chaining””.

A note on “daisy-chaining”

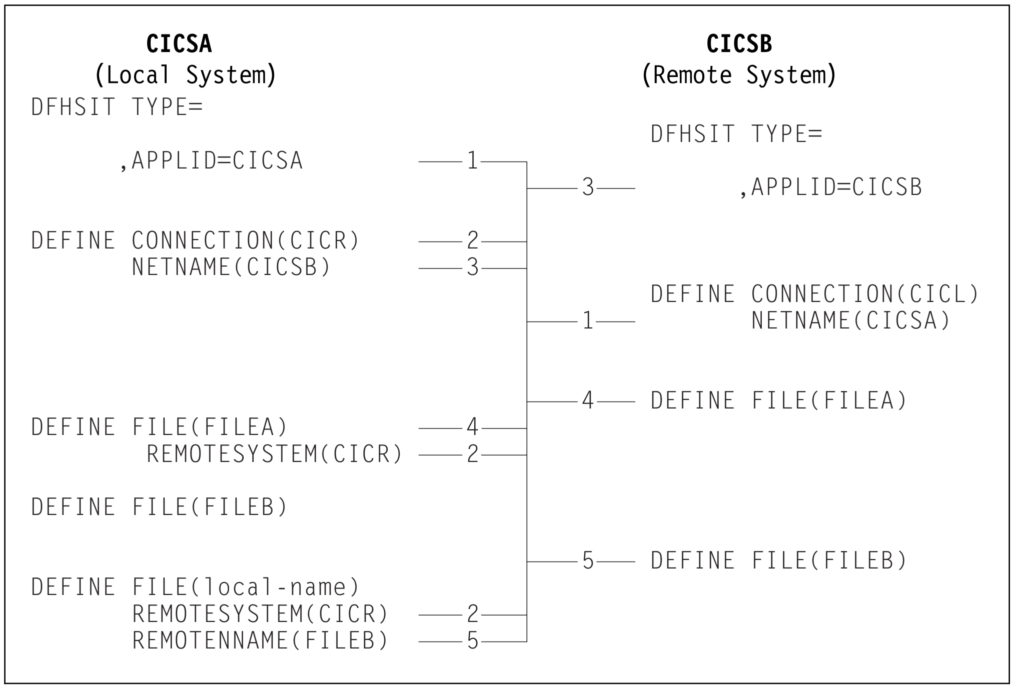
The descriptions of how to define remote resources in this chapter usually assume that there is a direct link between the local CICS and that on which the remote resource resides. In fact, in all types of CICS intercommunication, the local and remote systems need not be directly connected. A request for a remote resource can be “daisy-chained” across CICS systems by defining the resource as remote in each intermediate system, as well as (where necessary) in the local system.

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.

CICS definitions for remote resources all have a REMOTENAME option (RMTNAME on macro-level definitions) to enable you to specify the name by which the resource is known on the remote system. If you omit this option, CICS assumes that the local and remote names of the resource are identical.

Local and remote resource naming is illustrated in Figure 42.



```
         CICSA                                     CICSB
     (Local System)                           (Remote System)
DFHSIT TYPE=
                                               DFHSIT TYPE=
      ,APPLID=CICSA         ──1──┐
                                 ├───3──            ,APPLID=CICSB

DEFINE CONNECTION(CICR)     ──2──┤
       NETNAME(CICSB)       ──3──┤
                                 │             DEFINE CONNECTION(CICL)
                                 ├───1──              NETNAME(CICSA)


                                 ├───4──       DEFINE FILE(FILEA)
DEFINE FILE(FILEA)          ──4──┤
        REMOTESYSTEM(CICR)  ──2──┤

DEFINE FILE(FILEB)
                                 │
                                 ├───5──       DEFINE FILE(FILEB)
DEFINE FILE(local-name)          │
       REMOTESYSTEM(CICR)   ──2──┤
       REMOTENNAME(FILEB)   ──5──┘
```

Figure 42. Local and remote resource names

Figure 42 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 a local name on the local system, so that the file is referred to by its local name in the local system and by FILEB on the remote system. The “real” name of the remote file is specified in the REMOTENAME option. Note that CICSA can also own a local file called FILEB.

In naming remote resources, **be careful** not to create problems for yourself. You could, for instance, in Figure 42 on page 138, define FILEA in CICSB with REMOTESYSTEM(CICL). If you did that, CICS would recursively reship any request for FILEA until all available sessions had been allocated.

CICS function shipping

The remote resources that you may have to define if you are using CICS function shipping are:

- Remote files
- Remote DL/I PSBs
- Remote transient data destinations
- 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.

Applications can be designed to access files without being aware of their location. To support this facility, the remote file must be defined (with the REMOTESYSTEM option) in the local system.

Alternatively, 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.

A remote file can be defined using a DFHFCT TYPE=REMOTE macro or, for VSAM files, using RDO. The definitions shown below provide CICS with sufficient information to enable it to ship file control requests to a specified remote system.

```
Resource definition online    Macro-level definition
DEFINE                        DFHFCT TYPE=REMOTE
   FILE(name)                        ,FILE=name
   GROUP(.....)
   DESCRIPTION(......)
 Remote Attributes
   REMOTESYSTEM(name)                ,SYSIDNT=name
   REMOTENAME(name)                  [,RMTNAME=name]
   RECORDSIZE(record-size)           [,LRECL=record-size]
   KEYLENGTH(key-length)             [,KEYLEN=key-length]
```

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 “File control” on page 22.

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 REMOTESYSTEM option. If the name specified is that of the local system, the request is not shipped.

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 REMOTENAME 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 systems, the REMOTENAME option need not be specified.

Record lengths

The record length of a remote file can be specified in the RECORDSIZE option.

If your installation uses C for VSE, you should specify the record length for any file that has fixed-length records.

In all other cases, the record length either is a mandatory option on file control commands or can be deduced by the command-language translator.

Sharing file definitions

In some circumstances, two or more CICS systems can share a common CICS system definition (CSD) file. (For information about sharing a CSD, see the *CICS System Definition Guide*.) If the local and remote systems share a CSD, you need define each VSE/VSAM file used in function shipping only once.

A file must be fully defined by means of DEFINE FILE, just like a local file definition. In addition, the REMOTESYSTEM option must specify the sysidnt of the file-owning region. When such a file is installed on the file-owning region, a full, local, file definition is built. On any other system, a remote file definition is built.

Defining remote DL/I PSBs with CICS Transaction Server for VSE/ESA

To enable the local CICS system to access remote DL/I databases, you must define them using the DL/I DLZACT TYPE=RPSB macro. For details of the use of this macro, see the *DL/I DOS/VS Resource Definition and Utilities* manual.

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 CICS function shipping.

CICS application programs can name a remote system explicitly on transient data requests, by using 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 a DFHDCT TYPE=REMOTE resource definition macro. The format of this macro is shown in Figure 44.

```
DFHDCT   TYPE=REMOTE
         ,DESTID=name
         ,SYSIDNT=name
         [,LENGTH=length]
         [,RMTNAME=name]
```

Figure 44. Macro for defining remote transient data destinations

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 CICS function shipping.

CICS application programs can name a remote system explicitly on temporary storage requests, by using 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. Whether or not the SYSID option has been coded on the temporary storage request, you could use an XTSEREQ global user exit program to direct the request to a system on which the appropriate queue is defined. If you use this method, there is again no need for the remote temporary storage queue to be defined on the local system. For programming information about the XTSEREQ and XTSEREQC global user exits, see the *CICS Customization Guide*.

If the temporary storage request does not explicitly name the remote system, and you are not using an XTSEREQ exit, then 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 a DFHTST TYPE=REMOTE resource definition macro. The format of this macro is shown in Figure 45.

```
DFHTST   TYPE=REMOTE
         ,SYSIDNT=name
         ,DATAID=character-string
         [,RMTNAME=character-string]
```

Figure 45. Macro for defining remote temporary storage queues

CICS distributed program link (DPL)

You may have to define remote server programs if you are using CICS DPL. A remote server program is a program that resides on another CICS system. CICS program-control LINK requests that are made against a remote program are shipped to the remote system by means of CICS DPL.

Applications can be designed to link to programs without being aware of their location. To support this facility, the remote programs must be defined (with the REMOTESYSTEM option) in the local system.

Alternatively, the client programs can name a remote system explicitly on program-control LINK requests, by means of the SYSID option. If this is done, there is no need for the remote server program to be defined on the local CICS system.

Defining remote server programs

A remote server program can be defined using the CEDA transaction. The definitions shown below provide CICS with sufficient information to enable it to ship DPL requests to a specified remote system.

```
DEFINE
   PROGRAM(name)
   GROUP(.....)
   DESCRIPTION(......)
 Remote Attributes
   REMOTESYSTEM(name)
   REMOTENAME(name)
   TRANSID(name)
```

Figure 46. Defining a remote program (DPL)

The name of the remote system

The name of the server system to which LINK requests for this program are to be shipped is specified in the REMOTESYSTEM option.

Program names

The name by which the server program is known on the local CICS system is specified in the PROGRAM option. This is the name that is used in LINK requests by client programs in the local system.

The name by which the server program is known on the remote CICS system is specified in the REMOTENAME option. This is the name that is used in LINK requests that are shipped by CICS to the remote system.

If the name of the server program is to be the same on both the local and the remote systems, the REMOTENAME option need not be specified.

Transaction names

It is possible to use the program resource definition to specify the name of the mirror transaction under which the program, when used as a DPL server, is to run. The TRANSID option is used for this purpose.

Using autoinstall

As an alternative to being statically defined in the client system, the remote server program can be autoinstalled when a DPL request for it is first issued. If you use this method, you need to write an autoinstall user program to supply the name of the remote system. (For details of the CICS autoinstall facility for programs, see the *CICS Resource Definition Guide*. For programming information about writing program-autoinstall user programs, see the *CICS Customization Guide*.)

Asynchronous processing

The only remote resource definitions needed for asynchronous processing are for transactions that are named in the TRANSID option of EXEC CICS START commands.

Note, however, that an application can use the EXEC 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 is invoked from the local CICS system only by EXEC CICS 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.

More generally, however, applications are designed to start transactions without being aware of their location, and in this case an installed transaction definition for the transaction must be available.

Note: If the transaction is owned by another CICS system and may be invoked by CICS transaction routing as well as by START commands, you must define the transaction for transaction routing.

Remote transactions that are invoked only by START commands without the SYSID option require only basic information in the installed transaction definition. The form of resource definition used for this purpose is shown in Figure 47.

```
DEFINE
  TRANSACTION(name)
  GROUP(groupname)
 Remote attributes
  REMOTESYSTEM(sysidnt)
  REMOTENAME(name)
  LOCALQ(NO|YES)
```

Figure 47. 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 7, “Asynchronous processing” on page 41.

Restriction on the REMOTENAME option

Some asynchronous-processing requests are for processes that involve transaction routing. One example is an EXEC CICS START command to attach a remote transaction on a local terminal. To support such requests, the value of the REMOTENAME option and the transaction name must be the same on the local resource definition of the transaction to be started. If they are different, the requested transaction does not start, and the message DFHCR4310 is sent to the CSMT transient data queue in the requesting system.

CICS transaction routing

CICS transaction routing enables a “terminal” that is owned by one CICS system (the terminal-owning region) to be connected to a transaction that is owned by another CICS system (the application-owning region). The terminal- and application-owning regions must be connected either by MRO or by an APPC link.

Most of the terminal and session types supported by CICS are eligible for transaction routing. However, the following terminals are **not** eligible, and cannot be defined as remote resources:

- LUTYPE6.1 connections and sessions
- MRO connections and sessions
- IBM 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- VSE system consoles

Both the terminal and the transaction must be defined in both CICS systems, as follows:

1. In the terminal-owning region:
   a. The terminal must be defined as a local resource (or must be autoinstallable).
   b. The transaction must be defined as a remote resource if it is to be initiated from a terminal or by ATI.

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 and connection definitions” on page 147).
   b. The transaction must be defined as a local resource.

If transaction routing requests are to be “daisy-chained” across intermediate systems, 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 systems. If you are using non-VTAM terminals, you also need to define indirect links to the TOR on the AOR and the intermediate systems (see “Indirect links for transaction routing” on page 121).

Transactions are defined by resource definition online (RDO).

VTAM terminals are also defined by RDO, but for non-VTAM terminals you must use macro-level definition.

Defining remote VTAM terminals

This section tells you how to define remote VTAM terminals using RDO. However, you do not 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 and connection definitions” on page 147.

Remote VTAM terminals are defined by means of a DEFINE TERMINAL command on which:

- The REMOTESYSNET option specifies the netname (applid) of the TOR. This enables CICS to form the fully-qualified identifier of the remote terminal, even where there is no direct link to the TOR. (See “Local and remote names for terminals” on page 153.)
- The REMOTESYSTEM option specifies the name of the next link in the path to the TOR. If there is more than one possible path to the TOR, use REMOTESYSTEM to specify the next link in the preferred path.

  If REMOTESYSTEM names a direct link to the TOR, you do not need to specify REMOTESYSNET.

Only a few of the various terminal properties need be specified for a remote terminal definition. The required properties are shown in Figure 48.

```
DEFINE
 TERMINAL(trmidnt)
 GROUP(groupname)
Terminal identifiers
 TYPETERM(terminal-type)
 REMOTESYSTEM(sysidnt_of_next_system)
 REMOTESYSNET(netname_of_TOR)
 REMOTENAME(trmidnt_on_TOR)
```

Figure 48. 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 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**.

Defining remote APPC connections

Remote single-session APPC terminals can be defined by means of TERMINAL and TYPETERM definitions, as described for VTAM terminals in the previous section.

For remote parallel-session APPC systems and devices, you must define a remote connection, as shown in Figure 49. A SESSIONS definition is not required for a remote connection.

```
DEFINE
 CONNECTION(sysidnt_of_device)
 GROUP(groupname)
Connection identifiers
 NETNAME(netname_of_device)
Remote attributes
 REMOTESYSTEM(sysidnt_of_next_system)
 REMOTESYSNET(netname_of_TOR)
 REMOTENAME(sysidnt_of_device_on_TOR)
Connection properties
 ACCESSMETHOD(VTAM)
 PROTOCOL(APPC)
```

Figure 49. Defining a remote APPC connection (transaction routing)

Sharing terminal and connection definitions

In some circumstances, two or more CICS systems can share a common CICS system definition (CSD) file. (For information about sharing a CSD, see the *CICS System Definition Guide*.) If the local and remote systems share a CSD, you need define each terminal and APPC connection only once.

A 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 REMOTESYSNET option should specify the netname of the terminal-owning region.
- The REMOTESYSTEM option should specify the sysidnt by which the terminal-owning region knows itself.

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.

Similarly, an APPC connection must be fully defined by means of DEFINE CONNECTION, and must have one or more associated SESSIONS definitions. In addition, the REMOTESYSNET option should specify the netname of the TOR, and the REMOTESYSTEM option the sysidnt by which the TOR knows itself. When such a connection is installed on the terminal-owning region, a full, local, connection definition is built. On any other system, a remote connection definition is built, and the SESSIONS definition is ignored.

Note: The links you define between systems on the transaction routing path that share common terminal (or connection) definitions must be given the same name. That is, the CONNECTION definitions must be given the name that you specify on the REMOTESYSTEM option of the common TERMINAL definitions.

Shipping terminal and connection definitions

If you are using VTAM terminals on your terminal-owning region, 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.

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 valid definition of the terminal (which may have been shipped previously), it ignores the flag. Otherwise, it asks for the definition to be shipped.

Shipped terminal definitions are propagated to the connected CICS system using the ISC or MRO sessions providing the connection. When a terminal definition is shipped to another region, the TCTUA is also shipped, except when the principal facility is an APPC parallel session. When a routed transaction terminates, information from the TCTTE and the TCTUA is communicated back to the region that owns the terminal.

Note: APPC connection definitions and APPC terminal definitions are always shippable; no special resource definition is required.

Terminal definitions can be shipped across intermediate systems. If you use shippable terminals and there is more than one possible path from the AOR to the TOR, you may want to specify the preferred path by defining indirect links to the TOR on the AOR and the intermediate systems (see "Indirect links for transaction routing" on page 121).

Shipping terminals for ATI requests

If you require a transaction that is started by ATI to acquire a remote terminal, you normally statically define the terminal to the AOR and any intermediate systems. For example, specifying a remote terminal in DFHDCT DESTFAC=(TERMINAL,trmidnt) for an intrapartition transient data queue (see "Defining intrapartition transient data queues" on page 169) does *not* cause a terminal definition to be shipped from the remote system. However, if a shipped terminal definition has already been received, following a previous transaction routing request, the terminal is eligible for ATI requests.

Similarly, you normally statically define a remote APPC terminal or connection that is named in an ALLOCATE command.

However, if the TOR and AOR are directly connected, CICS does allow you to cause terminal definitions to be shipped to the AOR to satisfy 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.

Remember that **XALTENF and XICTENF can be used to ship terminal definitions only if there is a direct link between the TOR and the AOR**. See "Shipping terminals for automatic transaction initiation" on page 58 for more information.

If you function ship ATI requests from a terminal-owning region to the application-owning region, you may need to consider using the FSSTAFF (function-shipped START affinity) system initialization parameter. See "Shipping terminals for ATI from multiple TORs" on page 62 for more details.

Defining terminals as shippable

To make a terminal definition eligible for shipping, you must associate it with a TYPETERM that specifies SHIPPABLE(YES):

```
DEFINE
  TERMINAL(trmidnt)
  GROUP(groupname)
  AUTINSTMODEL(YES|NO|ONLY)
  AUTINSTNAME(name)
  TYPETERM(TRTERM1)
  .
  .
DEFINE
  TYPETERM(TRTERM1)
  .
  .
  SHIPPABLE(YES)
```

Figure 50. Defining a shippable terminal (transaction routing)

This method can be used for any VTAM terminal. It is particularly efficient if you use autoinstall in the TOR. In effect, it gives automatic installation of remote terminal definitions. For further information about autoinstall, see the *CICS Resource Definition Guide*. For programming information about the autoinstall user program for terminals, see the *CICS Customization Guide*.

Terminal definitions that have been shipped to an application-owning region eventually become redundant, and must be deleted from the AOR (and from any intermediate systems between the TOR and AOR). For information about this, see Chapter 24, "Efficient deletion of shipped terminal definitions" on page 225.

Defining remote non-VTAM terminals

A remote non-VTAM terminal requires a full terminal control table entry in the remote system (TOR), and a terminal control table entry in the local system (AOR) 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 definition of a remote terminal.

Non-VTAM terminal definitions are not shippable.

With resource definition macros, you can define remote terminals in either of two ways:

1. By means of DFHTCT TYPE=REMOTE macros
2. By means of normal DFHTCT TYPE=TERMINAL macros preceded by a DFHTCT TYPE=REGION macro

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.

Note: CICS Transaction Server for VSE/ESA Release 1 does not support the BTAM telecommunication access method. However, BTAM terminals can use transaction routing from a TOR that runs an earlier CICS release to gain access to a CICS Transaction Server for VSE/ESA Release 1 AOR. It follows from this that BTAM terminals can only be defined as *remote* in a CICS Transaction Server for VSE/ESA Release 1 system. For information about how to define remote BTAM terminals, refer to the manuals for the earlier CICS release.

Definition using DFHTCT TYPE=REMOTE

The format of the DFHTCT TYPE=REMOTE macro is reproduced here for ease of reference.

```
DFHTCT   TYPE=REMOTE
     ,ACCMETH=access-method
     ,SYSIDNT=name-of-CONNECTION-to-TOR
     ,TRMIDNT=name
     ,TRMTYPE=terminal-type
     [,ALTPGE=(lines,columns)]
     [,ALTSCRN=(lines,columns)]
     [,ALTSFX=number]
     [,DEFSCRN=(lines,columns)]
     [,ERRATT={NO|([LASTLINE][,INTENSIFY]
     [,{BLUE|RED|PINK|GREEN|TURQUOISE|YELLOW
        |NEUTRAL}]
     [,{BLINK|REVERSE|UNDERLINE}])}]
     [,FEATURE=(feature[,feature],...)]
     [,LPLEN={132|value}]
     [,PGESIZE=(lines,columns)]
     [,RMTNAME={name-specified-in-TRMIDNT|name}]
     [,STN2980=number]
     [,TAB2980={1|value}]
     [,TCTUAL=number]
     [,TIOAL={value|(value1,value2)}]
     [,TRMMODL=numbercharacter]
```

Figure 51. Defining a remote non-VTAM terminal (transaction routing)

SYSIDNT specifies the name of the connection to the terminal-owning region. If there is no direct link to the TOR, SYSIDNT must specify the name of an **indirect link** (see "Indirect links for transaction routing" on page 121).

Sharing terminal definitions: With the exception of SYSIDNT, the operands of DFHTCT TYPE=REMOTE form a subset of those that can be 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 becomes equivalent to the DFHTCT TYPE=TERMINAL form.

A single DFHTCT TYPE=REMOTE macro 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 52 on page 151.

```
  Local System CICL              Remote System CICR
        AOR                             TOR

 DFHSIT TYPE=                    DFHSIT TYPE=
       SYSIDNT=CICL                    SYSIDNT=CICR

 DFHTCT TYPE=INITIAL,            DFHTCT TYPE=INITIAL,
        ACCMETH=NONVTAM,                ACCMETH=NONVTAM,
        SYSIDNT=CICL,                   SYSIDNT=CICR,
        .                               .
        .                               .

                                 DFHTCT TYPE=SDSCI
                                        .
                                        .
                                        .
                                 DFHTCT TYPE=SDSCI
                                        .
                                        .
                                        .

                                 DFHTCT TYPE=LINE
                                        .
                                        .

 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 52. Typical use of DFHTCT TYPE=REMOTE macro

In Figure 52, the same terminal definition is used in both the local and the remote systems.

In the local system, the fact that the terminal sysidnt differs from that of the local system (specified on the DFHTCT TYPE=INITIAL macro) causes a remote terminal entry to be built. In the remote system, the fact that the terminal sysidnt is that of the remote system itself causes the TYPE=REMOTE macro to be treated exactly as if it were a TYPE=TERMINAL macro.

Note: For this method to work, the CONNECTION from the local system to the remote system must be given the name of the sysidnt by which the remote system knows itself (CICR in the example).

The terminal identification is "aaaa" in both systems.

Definition using DFHTCT TYPE=REGION

If you use the DFHTCT TYPE=REGION macro, you can define terminals in the same way as local terminals, using DFHTCT TYPE=SDSCI, TYPE=LINE, and TYPE=TERMINAL macros. The definitions must, however, be preceded by a DFHTCT TYPE=REGION macro, which has the following form:

```
DFHTCT   TYPE=REGION
         ,SYSIDNT={name-of-CONNECTION-to-TOR|LOCAL}
```

SYSIDNT specifies the name of the connection to the terminal-owning region. If there is no direct link to the TOR, SYSIDNT must specify the name of an **indirect link** (see “Indirect links for transaction routing” on page 121).

Sharing terminal definitions: If SYSIDNT 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. Parameters 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 using DFHTCT TYPE=REGION,SYSIDNT=LOCAL.

A typical use of this method of definition is shown in Figure 53.

```
Terminal-Owning Region              Application-Owning Region

  DFHTCT TYPE=INITIAL,                 DFHTCT TYPE=INITIAL,
         SYSIDNT=TERM,                        SYSIDNT=TRAN,
         ACCMETH=NONVTAM                      ACCMETH=NONVTAM
         .                                    .

                                       DFHTCT TYPE=REGION,
                                              SYSIDNT=TERM

  COPY TERMDEFS                        COPY TERMDEFS

                                       DFHTCT TYPE=REGION,
                                              SYSIDNT=LOCAL

  DFHTCT TYPE=FINAL                    DFHTCT TYPE=FINAL
```

```
 * TERMDEFS COPYBOOK

    DFHTCT TYPE=SDSCI,DEVICE=R3270,DSCNAME=R70IN,
           DDNAME=R3270IN,OPTCD=WU,MACRF=R,RECFM=U,BLKSIZE=2024
    DFHTCT TYPE=SDSCI,DEVICE=R3270P,DSCNAME=R70OUT,
           DDNAME=R3270OUT,OPTCD=WU,MACRF=W,RECFM=U,
           BLKSIZE=2024
 *** INPUT LINE ***
    DFHTCT TYPE=LINE,ACCMETH=BTAM,NPDELAY=16000,INAREAL=2024,
           DSCNAME=R70IN,TRMTYPE=3277
    DFHTCT TYPE=TERMINAL,TRMIDNT=L7IN,TRMPRTY=32,LASTTRM=LINE,
           TIOAL=80,TRMMODL=2
 *** OUTPUT LINE ***
 OUTQ70   DFHTCT TYPE=LINE,ACCMETH=BTAM,NPDELAY=16000,
           INAREAL=2024, DSCNAME=R70OUT,TRMTYPE=3286
 *
 TRM1     DFHTCT TYPE=TERMINAL,TRMIDNT=TRM1,TRMTYPE=LUTYPE2,
           TRMMODL=2,CLASS=(CONV,VIDEO),FEATURE=(SELCTPEN,
           AUDALARM,UCTRAN),TRMPRTY=100,
           TRMSTAT=(TRANSCEIVE),LASTTRM=POOL
```

Figure 53. Typical use of DFHTCT TYPE=REGION macro

In Figure 53, the same copy book of terminal definitions is used in both the terminal-owning region and the application-owning region.

Note: In this example, the application-owning region (AOR) can be either a CICS Transaction Server for VSE/ESA or a CICS/VSE 2.3 system, but the terminal-owning region (TOR) can only be a CICS/VSE 2.3 system.

In the application-owning region, the fact that the sysidnt specified in the TYPE=REGION macro differs from the sysidnt specified in 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 application-owning region, any defaults that they imply 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 (netname) 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 hard-coded remote terminal definitions:

- The definition must enable CICS to access the netname of the terminal-owning region. For example, if you are using VTAM terminals and there is no direct link to the TOR, you should use the REMOTESYSNET option to provide the netname of the TOR.

  If you are using non-VTAM terminals and there is no direct link to the TOR, the SYSIDNT operand of the DFHTCT TYPE=REMOTE or TYPE=REGION macro must specify the name of an **indirect link** (on which the NETNAME option names the applid of the TOR).
- The "real" terminal identifier must always be specified, either directly or by means of an alias.

Providing the netname of the TOR

You must always ensure that the remote terminal definition allows CICS to access the netname of the TOR. In the following examples, it is assumed that the applid of the terminal-owning region is PRODSYS.

```
VTAM terminal definition
  DEFINE TERMINAL            DEFINE CONNECTION(PD1)     Direct link
  REMOTESYSTEM(PD1)          NETNAME(PRODSYS)           to TOR
  .                          .
  .                          .

VTAM terminal definition
  DEFINE TERMINAL            DEFINE CONNECTION(NEXT)    No direct
  REMOTESYSTEM(NEXT)         NETNAME(INTER1)            link to TOR
  REMOTESYSNET(PRODSYS)
  .                          .
  .                          .

Non-VTAM terminal definition
(method 1)
  DFHTCT TYPE=REMOTE,       DEFINE CONNECTION(PD1)      Direct link
  SYSIDNT=PD1,              NETNAME(PRODSYS)            to TOR
  .                         .
  .                         .

Non-VTAM terminal definition
(method 2)
  DFHTCT TYPE=REGION,       DEFINE CONNECTION(PD1)      Direct link
  SYSIDNT=PD1               NETNAME(PRODSYS)            to TOR
  .                         .
  .                         .
Non-VTAM terminal definition
(method 1)
  DFHTCT TYPE=REMOTE,       DEFINE CONNECTION(REMT)     No direct
  SYSIDNT=REMT,             NETNAME(PRODSYS)            link to TOR
                            ACCESSMETHOD(INDIRECT)
                            INDSYS(NEXT)
  DFHTCT TYPE=TERMINAL,
  .
```

Figure 54. Identifying a terminal-owning region

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 have to provide an alias if the terminal-owning region and the application-owning region each 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:

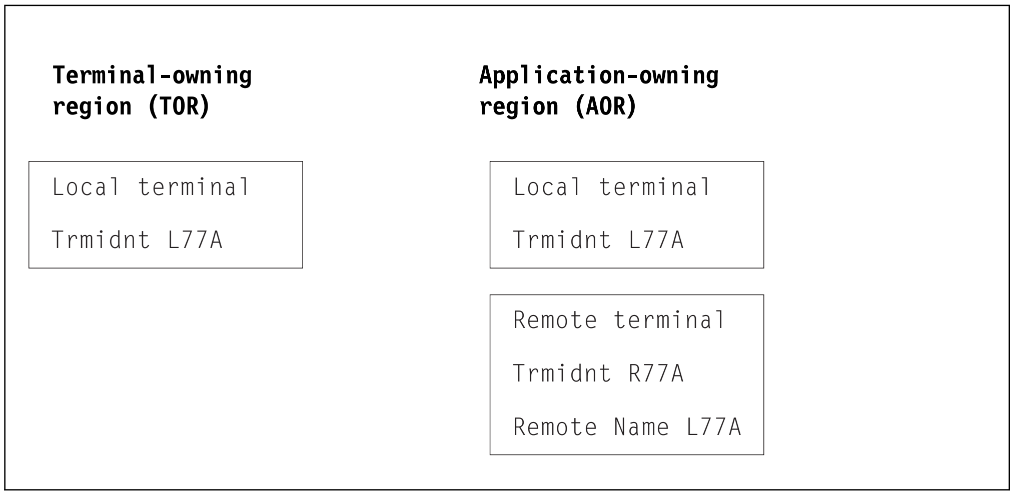


Figure 55. Local and remote names for remote terminals

You specify the remote name in the REMOTENAME option of DEFINE TERMINAL or the RMTNAME operand of DFHTCT TYPE=REMOTE.

Defining transactions for transaction routing

This section discusses the definition of transactions that may be invoked by transaction routing.

The general form of the CEDA DEFINE command for a transaction is shown in Figure 56 on page 156.

```
  DEFINE
    TRANSACTION(name)
    GROUP(groupname)
    PROGRAM(name)
    TWASIZE(0|value)
    PROFILE(DFHCICST|name)
    PARTITIONSET(name)
    STATUS(ENABLED|DISABLED)
    PRIMEDSIZE(00000|value)
    TASKDATALOC(BELOW|ANY)
    TASKDATAKEY(USER|CICS)
    STORAGECLEAR(NO|YES)
    RUNAWAY(SYSTEM|value)
    SHUTDOWN(DISABLED|ENABLED)
 REMOTE ATTRIBUTES
    DYNAMIC(NO|YES)
    REMOTESYSTEM(name)
    REMOTENAME(local-name|remote-name)
    TRPROF(DFHCICSS|name)
    LOCALQ(NO|YES)
 SCHEDULING
    PRIORITY(1|value)
    TCLASS(NO|value)
    TRANCLASS(DFHTLC00|name)
 ALIASES
    ALIAS(name)
    TASKREQ(value)
    XTRANID(value)
    TPNAME(name)
    XTPNAME(name)
 RECOVERY
    DTIMOUT(NO|value)
    INDOUBT(BACKOUT|COMMIT|WAIT)
    RESTART(NO|YES)
    SPURGE(NO|YES)
    TPURGE(NO|YES)
    DUMP(YES|NO)
    TRACE(YES|NO)
 SECURITY
    RESSEC(NO|YES)
    CMDSEC(NO|YES)
    EXTSEC(NO|YES)
    TRANSEC(01|value)
    RSL(00|value|Public)
```

Figure 56. The CEDA DEFINE TRANSACTION options

The way in which a transaction is selected for local or remote execution is determined by the *remote attributes* that are specified in the transaction definition. There are three possible cases:

1. The remote attributes specify DYNAMIC(NO), and the REMOTESYSTEM name is either blank or the sysid of the local system.

   In this case, the transaction is always executed locally, and transaction routing is not involved.

2. The remote attributes specify DYNAMIC(NO), and the REMOTESYSTEM name differs from the sysid of the local system.

   In this case, the transaction is always routed to the system named in the REMOTESYSTEM option. This is known as **static** transaction routing.

3. The remote attributes specify DYNAMIC(YES).

   In this case, the decision about where to execute the transaction is taken by your dynamic transaction routing program. See “Dynamic transaction routing” on page 55.

The name in the TRANSACTION option 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 slot reader, or operator ID card reader, are used.

The attributes that you define always apply to the execution of the transaction in the terminal-owning region, and never to the execution of the routed transaction in the application-owning region.

If there is a possibility that the transaction will be executed locally, the definition must follow the normal rules for the definition of a local transaction. In particular, the PROGRAM option must name a user program that will be installed in the local system. When the transaction is routed to another system, the program associated with it is always the relay program DFHAPRT, irrespective of the name specified in the PROGRAM option.

The PROFILE option names the profile that is to be used for communication between the terminal and the relay transaction (or the user transaction if the transaction is executed locally). For remote execution, the TRPROF option names the profile that is to be used for communication on the session between the relay transaction and the remote transaction-owning system. Information about profiles is given under “Defining communication profiles” on page 163.

When a transaction will always be routed to a remote system, so that the transaction executed in the local system is always the relay transaction, you might want to specify some options for control of the relay transaction:

- You can set or default TWASIZE to zero, because the relay transaction does not require a TWA.
- You should specify transaction security for routed transactions that are operator initiated. You do not need to specify resource security checking, because the relay transaction does not access resources.
- For transaction routing on mapped APPC connections, you should code the RTIMOUT option on the communication profile named on the TRPROF option of the transaction definition. This causes the relay transaction to be timed out if the system to which a transaction is routed does not respond within a reasonable time.

  Deadlock time-out (specified on the DTIMOUT option of the transaction definition) is not triggered for terminal I/O waits. Because the relay transaction does not access resources after obtaining a session, it has little need for DTIMOUT except to trap suspended ALLOCATE requests. (Methods for specifying whether, if there are no free sessions to a remote system, ALLOCATE requests should be queued or rejected, are described in Chapter 23, “Intersystem session queue management” on page 221.)

The method you use to define transactions for routing may differ, depending on whether the transactions are to be statically or dynamically routed.

Static transaction routing

There are two methods of defining transactions that are to be statically routed.

Using separate local and remote definitions: You create a remote definition for the transaction, and install it on the TOR: the REMOTESYSTEM option must specify the name of the target AOR (or the name of an intermediate system, if the request is to be "daisy-chained"). You install separate remote definitions for the transaction on any intermediate systems: the REMOTESYSTEM option must specify the name of the next system in the routing chain. You create a local definition for the transaction, and install it on the target AOR: the REMOTESYSTEM option must be blank, or specify the name of the AOR.

If two or more systems along the transaction-routing path share the same CSD, the transaction definitions should be in different groups.

Using dual-purpose definitions: You create a single transaction definition, which is shared between the TOR and the AOR (and possibly between intermediate systems too, if "daisy chaining" is involved). The REMOTESYSTEM option specifies the name of the target AOR.

When the definition is installed on each system, the local CICS compares its SYSIDNT with the REMOTESYSTEM name. If they are different (as in the TOR), a remote transaction definition is created. If they are the same (as in the target AOR), a local transaction definition is installed.

It is recommended that, for static transaction routing, you use this method wherever possible. Because you have only one set of CSD records to maintain, it provides savings in disk storage and time. However, you can use it only if your systems share a CSD. For information about sharing a CSD, see the *CICS System Definition Guide*.

Dynamic transaction routing

There are likewise two methods of defining transactions that are to be dynamically routed.

Note: Using dual-purpose definitions is a third possible method, but is not recommended for transactions that are to be dynamically routed. This is because the DYNAMIC(YES) attribute on the shared definition causes the dynamic transaction routing program to be invoked unnecessarily in the target AOR, after the transaction has been routed.

Using separate local and remote definitions: This method is as described under "Static transaction routing."

Using a single transaction definition in the TOR: This is the recommended method. Using it, in the TOR (and in any intermediate systems) you install only *one* transaction definition that specifies DYNAMIC(YES). This single definition provides a set of default attributes for *all* transactions that are dynamically routed. The name of the common definition is that specified on the DTRTRAN system initialization parameter. The default name is CRTX, which is the name of a CICS-supplied transaction definition that is included in the CSD group DFHISC.

If, at transaction attach, CICS cannot find an installed resource definition for a user transaction identifier (transid), it attaches a transaction built from the user transaction identifier and the set of attributes taken from the common transaction definition. (If the transaction definition specified on the DTRTRAN parameter is not installed, CICS attaches the CICS-supplied transaction CSAC. This sends message DFHAC2001—“Transaction *‘tranid’* is unrecognized”—to the user’s terminal.) Because the common transaction definition specifies DYNAMIC(YES), CICS invokes the dynamic transaction routing program to select a target application-owning region and, if necessary, name the remote transaction.

In the target AOR, you install a local definition for each dynamically-routed transaction.

If you use this method:

- Dynamically-routed transactions should be installed in the terminal-owning region (if local to the TOR), or the application-owning region (if local to the AOR), but not both.
- The only transaction you should define as dynamic is the dynamic transaction routing definition specified on the DTRTRAN parameter.
- The only transactions you should define as remote are those that are to be started on remote systems by EXEC CICS START commands, and any that are to be statically routed.

This greatly simplifies the task of managing resource definitions.

It is recommended that you create your own common transaction definition for dynamic routing, using CRTX as a model. The attributes specified on the CRTX definition are shown in Figure 57.

```
  DEFINE
    TRANSACTION(CRTX)
    GROUP(DFHISC)
    PROGRAM(########)
    TWASIZE(00000)
    PROFILE(DFHCICST)
    STATUS(ENABLED)
    TASKDATALOC(ANY)
    TASKDATAKEY(CICS)
REMOTE ATTRIBUTES
    DYNAMIC(YES)
    REMOTESYSTEM()
    REMOTENAME()
    TRPROF(DFHCICSS)
RECOVERY
    DTIMOUT(NO)
    INDOUBT(BACKOUT)
    RESTART(NO)
    SPURGE(YES)
    TPURGE(YES)
```

Figure 57. Main attributes of the CICS-supplied CRTX transaction

The key parameters of this transaction definition are described below:

DYNAMIC(YES)
This is required for a dynamic transaction routing definition that is specified on the DTRTRAN system initialization parameter. You can change the other parameters when creating your own definition, but must specify DYNAMIC(YES).

PROGRAM(########)
The CICS-supplied default transaction specifies a dummy program name, ########. If your dynamic transaction routing program allows a transaction to run in the local region, and its definition specifies the dummy program name, CICS is unlikely to find such a program, causing a "program-not-found" condition.

You are recommended to specify the name of a program that you want CICS to invoke whenever the transaction:

- Is not routed to a remote system, and
- Is not rejected by the dynamic transaction routing program by means of the DYRDTRRJ parameter, and
- Is run in the local region.

You can use the local program to issue a suitable response to a user's terminal in the event that the dynamic routing program decides it cannot route the transaction to a remote system.

TRANSACTION(CRTX)
The name of the CICS-supplied dynamic transaction routing definition. Change this to specify your own transaction identifier.

RESTART(NO)
This attribute is forced for a routed transaction.

REMOTESYSTEM
You can code this to specify a default AOR for transactions that are to be dynamically routed.

Distributed transaction processing

For MRO and LUTYPE6.1 links, there is no need to define any remote resources for DTP, provided that the front-end and back-end systems are directly connected. Both the remote system and the remote transaction are identified on the EXEC CICS commands issued by the front-end transaction. CICS therefore has all the necessary information to connect a session and attach the back-end transaction. (However, if the back-end transaction is to be routed to, it must be defined as a remote resource on the intermediate systems—see "A note on "daisy-chaining"" on page 138.)

If you use the EXEC CICS API over APPC links, you can either identify the remote system and transaction explicitly, as for MRO and LUTYPE6.1 links, or by reference to a PARTNER definition. If you choose to do the latter, you need to create the appropriate PARTNER definitions. If you use the CPI Communications API over APPC links, the syntax of the commands *requires* you to create a PARTNER definition for every remote partner referenced.

Figure 58 on page 161 shows the general form of the CEDA DEFINE PARTNER command.

```
DEFINE
  PARTNER(sym_dest_name)
  [GROUP(groupname)]
  [NETWORK(name)]
  NETNAME(name)
  [PROFILE(name)]
  {TPNAME(name)|XTPNAME(value)}
```

Figure 58. Defining a remote partner

The PARTNER resource has been designed specifically to support **Systems Application Architecture (SAA) conventions**. For more guidance about this, see the *CICS Resource Definition Guide* and the *SAA Common Programming Interface Communications Reference* manual.

For guidance about designing and developing distributed transaction processing applications, see the *CICS Distributed Transaction Programming Guide*.

Chapter 15. Defining local resources

This chapter discusses how to define resources, required for intersystem communication, that reside in the local CICS system. The chapter contains the following topics:

Defining communication profiles

When a transaction acquires an APPC, MRO, or LUTYPE6.1 session to another system, either explicitly by means of an EXEC CICS 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:

- Whether function management headers (FMHs) received from the session are to be passed on to the transaction.
- Whether input and output messages are to be journaled, and if so the location of the journal.
- The node error program (NEP) class for errors on the session.
- For APPC 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 EXEC CICS ALLOCATE command.

The options of the CEDA DEFINE PROFILE command that are relevant to intersystem sessions are shown in Figure 59 on page 164. For further information about the CEDA DEFINE PROFILE command, see the *CICS Resource Definition Guide*.

A profile is always required for a session acquired by an EXEC CICS 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.

The only option shown in Figure 59 on page 164 that applies to MRO sessions is INBFMH. And, for MRO sessions that are acquired by an EXEC CICS ALLOCATE command, CICS always uses INBFMH(ALL), no matter what is specified in the profile.

For APPC conversations, INBFMH specifications are ignored; APPC FMHs are never passed to CICS application programs.

```
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)]
```

Figure 59. Defining a communication profile

It is usually important to ensure that an intercommunicating transaction never waits indefinitely for data from its partner transaction. The RTIMOUT option should be given a value suitable for intersystem working: rather less than the time-out periods typically specified for terminals used as operator interfaces. The RTIMOUT value should also be greater than the DTIMOUT value specified on the partner transaction definition.

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 options than the form required for alternate facilities.

The RTIMOUT value defined for a back-end transaction needs to be at least as great as that specified for its front-end partner's principal facility. This is to cover the possibility of the back-end transaction waiting almost that period of time (plus some execution and network time) to receive data from its front-end.

Default profiles

CICS provides a set of communication profiles, which it uses when the user does not or cannot specify a profile explicitly:

DFHCICST
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
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.

DFHCICSP
The profile for principal facilities of the CICS-supplied page-retrieval transaction, CSPG. CICS uses this profile for CSPG even if you alter the CSPG transaction definition to specify a different one. For further information about communication profiles used by CICS-supplied transactions, see the *CICS-Supplied Transactions* manual.

DFHCICSE

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)

The default profile for alternate facilities that are acquired by means of an application program EXEC CICS 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)

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)

The profile that CICS uses in transaction routing for communication between the relay transaction (running in the terminal-owning region) and the interregion link or APPC link.

DFHCICSR INBFMH(ALL)

The profile that CICS uses in transaction routing for communication between the user transaction (running in the transaction-owning region) and the interregion link or APPC link.

Note that the user-transaction's principal facility is the surrogate TCTTE in the transaction-owning region, for which the default profile is DFHCICST.

Modifying the default profiles

You can modify a default profile by means of the CEDA transaction.

A typical reason for modification is to include a modename to provide class of service selection for, say, function shipping requests on APPC links. If you do this, you must ensure that every APPC link in your installation has a group of sessions with the specified modename.

You must not modify DFHCICSV, which is used exclusively by some CICS-supplied transactions.

You can modify DFHCICSP, used by the CSPG page-retrieval transaction. The supplied version of DFHCICSP specifies UCTRAN(YES). Be aware that, if you specify UCTRAN(NO), terminals defined with UCTRAN(NO) will be unable to make full use of page-retrieval facilities.

If you modify DFHCICSA, you must retain INBFMH(ALL), because it is required by some CICS-supplied transactions. Modifying this profile does not affect the profile options assumed for MRO sessions.

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 be 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 APPC service managers.

Note: With the exception of the APPC LU services model, the architected processes are defined in the LUTYPE6.1 architecture. CICS, however, also uses them for function shipping on APPC links by using APPC migration mode.

The appropriate models are also used for CICS-to-CICS communication. The exceptions are CICS file control requests, which are handled by a CICS-defined file control model, and CICS transaction routing, which uses protocols that are private to CICS.

During resource definition, your only involvement with architected processes is to ensure that the relevant transactions and programs are included in your CICS system, and possibly to change their priorities.

Process names

Architected process names are one through 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 60 on page 167.

```
  XTRANID     TRANSID     PROGRAM     DESCRIPTION

For CICS file control
     -        CSMI        DFHMIRS     File control model

For LUTYPE6.1 architected processes
  01000000    CSM1        DFHMIRS     System message model
  02000000    CSM2        DFHMIRS     Scheduler model
  03000000    CSM3        DFHMIRS     Queue model
  05000000    CSM5        DFHMIRS     DL/I model

For APPC architected processes
  06F10000    CLS1        DFHLUP      LU services model
  06F20000    CLS2        DFHLUP      LU services model
     -        CLS3        DFHLUP      LU services model
```

Figure 60. CICS architected process names

Modifying the architected process definitions

The previous list shows that the CICS file control model and the architected processes for function shipping all map to program DFHMIRS, 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, TRANSID, or PROGRAM values.

You can modify any of the definitions by means of the CEDA transaction. In particular, you may want to change the DTIMOUT value on the mirror transactions.

The definitions for the mirror transactions are supplied with DTIMOUT(NO) specified. If you are uncomfortable with this situation, you should change the definitions to specify a value other than NO on the DTIMOUT option. However, before changing these definitions, you first have to copy them to a new group.

Interregion function shipping

Function shipping over MRO links can employ long-running mirror tasks and the short-path transformer program. (See “MRO function shipping” on page 27.)

If you modify one or more of the mirror transaction definitions, you must evaluate the effect that this may have on interregion function shipping.

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 required resource definitions for installation

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. For information about how to include these pregenerated CEDA groups in your CICS system, see the *CICS Resource Definition Guide*.

Some of the contents of groups DFHISC and DFHSTAND are summarized in Figure 61.

TRANSACTIONS

XTRANID	TRANSID	PROGRAM	GROUP	
-	CSMI	DFHMIRS	DFHISC	CICS file control model
01000000	CSM1	DFHMIRS	DFHISC	System message model
02000000	CSM2	DFHMIRS	DFHISC	Scheduler model
03000000	CSM3	DFHMIRS	DFHISC	Queue model
05000000	CSM5	DFHMIRS	DFHISC	DL/I model
06F10000	CLS1	DFHLUP	DFHISC	LU services model
06F20000	CLS2	DFHLUP	DFHISC	LU services model
-	CLS3	DFHLUP	DFHISC	LU services model
-	CEHP	DFHCHS	DFHISC	CICS/VM™ request handler
-	CEHS	DFHCHS	DFHISC	CICS/VM request handler
-	CMPX	DFHMXP	DFHISC	Local queue shipper
-	CPMI	DFHMIRS	DFHISC	Synclevel 1 mirror
-	CRSQ	DFHCRQ	DFHISC	Remote schedule purge program
-	CRSR	DFHCRS	DFHISC	Remote scheduler program
-	CRTE	DFHRTE	DFHISC	Routing transaction
-	CSNC	DFHCRNP	DFHISC	Interregion connection manager
-	CSSF	DFHRTC	DFHISC	CRTE cancel command processor
-	CVMI	DFHMIRS	DFHISC	APPC sync level-1 mirror
-	CXRT	DFHCRT	DFHISC	Relay transaction for LU6.2

PROGRAMS

NAME	GROUP	
DFHCCNV	DFHISC	CICS data conversion program
DFHCRNP	DFHISC	Interregion new connection manager
DFHCRQ	DFHISC	ATI purge program
DFHCRR	DFHISC	IRC session recovery program
DFHCRS	DFHISC	Remote scheduler program
DFHCRSP	DFHISC	Interregion control initialization program
DFHCRT	DFHISC	Transaction routing relay program for APPC alternate facilities
DFHDYP	DFHISC	Standard dynamic transaction routing program
DFHLUP	DFHISC	LU services program
DFHMIRS	DFHISC	Mirror program
DFHMXP	DFHISC	Local queuing shipper program
DFHRTC	DFHISC	CRTE cancel command processor
DFHRTE	DFHISC	Transaction routing 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 61. Some definitions required for ISC and MRO

Defining intrapartition transient data queues

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)}
       ...
```

Figure 62. Defining an intrapartition transient data queue

For further information about the DFHDCT macro, see the *CICS Resource Definition Guide*. This section is concerned with the CICS intercommunication aspects of queues that:

- Cause automatic transaction initiation (TRANSID specified)
- 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 option must not be defined as a remote transaction.

Principal facilities

The principal facility that is to be associated with a transaction started by ATI is specified in the DESTFAC operand. It can be:

- A local terminal
- A remote terminal
- A local session or APPC device
- A remote APPC session or device

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 APPC terminal, you need to specify DESTFAC=(TERMINAL[,termid]). If you omit **termid**, the name of the terminal defaults to the name of the queue (specified in DESTID).

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 8, “CICS transaction routing” on page 53), and the normal transaction routing rules apply.

For any remote terminal other than an APPC terminal, specify:

```
DESTFAC=(TERMINAL[,termid]).
```

The terminal itself must be defined as a remote terminal (or a shipped terminal definition must be made available), and the terminal-owning region must be connected to the local system either by an IRC link or by an APPC link.

Local sessions and APPC devices

You can name a local connection definition in the DESTFAC=(SYSTEM,sysid) operand. The remote system can be connected by IRC, LUTYPE6.1, or APPC link. In the APPC case, "system" can be a hard-coded terminal-like device.

CICS allocates a session on the specified system, which becomes the principal facility to **transid**. The transaction program converses across the session using the appropriate DTP protocol. Read Chapter 9, "Distributed transaction processing" on page 71 for an introduction to DTP.

The transaction starts in 'allocated' state on its principal facility. Then it identifies its partner transaction; that is, the process to be connected to the other end of the session. In the APPC protocol, it does this by issuing the EXEC CICS CONNECT PROCESS command, a command normally only used to start a conversation on an alternate facility.

The partner transaction, having been started in the back end with the conversation in RECEIVE state, also sees the session as its principal facility. This is unusual in that CICS treats either end of the session as a principal facility. On both sides, the conversation identifier is taken from EIBTRMID if needed, but it is also implied on later commands, as is the case for principal facilities.

Remote APPC sessions and devices

A remote connection is a connection that is defined as remote on the system that owns the transient data queue and the associated transaction. Automatic transaction initiation with a remote APPC connection is a form of CICS transaction routing (see Chapter 8, "CICS transaction routing" on page 53), and the normal transaction routing rules apply.

You can name a remote connection definition in the DESTFAC=(SYSTEM,sysid) operand.

The connection itself must be defined as a remote connection (or a shipped connection definition must be made available), and the terminal-owning region must be connected to the local system either by an IRC link or by an APPC link. The remarks in "Local sessions and APPC devices" about handling the link after transaction initiation apply also to routed transactions.

Defining local resources for DPL

To support DPL, special resource definitions are sometimes necessary for server programs and mirror transactions.

Mirror transactions

You can specify whatever names you like for the mirror transactions to be initiated by DPL requests. Each of these transaction names must be defined in the server region on a transaction that invokes the mirror program DFHMIRS. Defining user transactions to invoke the mirror program gives you the freedom to specify appropriate values for all the other options on the transaction resource definition.

Server programs

If a local program is to be requested by some other region as a DPL server, there must be a resource definition for that program. The definition can be statically defined, or installed automatically (autoinstalled) when the program is first called. (For details of the CICS autoinstall facility for programs, see the *CICS Resource Definition Guide*.)

Part 4. Application programming

Table 14. Application programming road map

If you want to...	Refer to...
Read an introduction to application programming in an intercommunication environment	Chapter 16, “Application programming overview” on page 175
Write application programs that use function shipping	Chapter 17, “Application programming for CICS function shipping” on page 177
Write application programs that use distributed programming link	Chapter 18, “Application programming for CICS DPL” on page 181
Write application programs that use EXCI	Chapter 19, “Application programming for the external CICS interface” on page 185
Write application programs that use asynchronous processing	Chapter 20, “Application programming for asynchronous processing” on page 191
Write application programs that may be invoked by transaction routing	Chapter 21, “Application programming for CICS transaction routing” on page 193
Write CICS application programs that communicate with an IMS system	Chapter 22, “CICS-to-IMS applications” on page 197

This part of the manual documents General-use Programming Interface and Associated Guidance Information.

Chapter 16. 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
- Distributed program link
- The external CICS interface
- 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, you can use the relevant chapter 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.

For guidance about application design and programming for distributed transaction processing, see the *CICS Distributed Transaction Programming Guide*.

Terminology

The following terms are sometimes used without further explanation in the remaining chapters of this part:

Principal facility
This term means 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.

Chapter 17. Application programming for CICS function shipping

This chapter contains the following topics:

Introduction to programming for 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, C, COBOL, or assembler language. Function shipping is available by using EXEC CICS commands, DL/I calls or EXEC DLI commands.

The commands that you can use to access remote resources are:

- File control commands
- DL/I calls or EXEC DLI commands
- Temporary storage commands
- Transient data commands

For information about interval control commands, see Chapter 20, “Application programming for asynchronous processing” on page 191.

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 specified in the resource definition. Optionally, you can use the SYSID option on EXEC commands to select the system on which the command is to be executed. In this case, the resource definitions 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 like a remote extension of your application program. For more information about this mechanism, read Chapter 4, “CICS function shipping” on page 21.

Although the same commands are used to access both local and remote resources, there are 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 files located on a remote system.

If you use the SYSID option to access a remote system directly, you must observe the following two 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 definition of the file to this CICS does not specify the appropriate values.

DL/I

Function shipping allows you to access IMS/ESA DM or IMS/VS DB databases associated with a remote CICS Transaction Server for OS/390, CICS/ESA, CICS/MVS or CICS/OS/VS system; or DL/I for VSE/ESA databases associated with a remote CICS/VSE or CICS Transaction Server for VSE/ESA system. (See Chapter 1, "Introduction to CICS intercommunication" on page 3 for a list of systems with which CICS Transaction Server for VSE/ESA Release 1 can communicate.)

Definitions of remote DL/I databases are provided by the system programmer. There is no facility for selecting specific systems in CICS application programs.

The following DL/I requests can be function shipped to a remote CICS system:

```
DLET        GN          PCB
GHN         GNP         REPL
GHNP        GU          TERM
GHU         ISRT
```

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 EXEC CICS 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 EXEC CICS WRITEQ TS command can be used to select the required type of storage.

For APPC 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 EXEC CICS 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 records, 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

The SYSIDERR condition is raised in the application program if:

- The link to the remote system is out of service.
- The named system is not defined. 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 link to the remote system is busy, and the maximum number of queued requests specified on the QUEUELIMIT option of the CONNECTION definition has been reached.
- The link to the remote system is busy, the maximum number of queued requests has *not* been reached, but your XZIQUE or XISCONA global user exit program specifies that the request should not be queued. (For programming information about the XZIQUE and XISCONA exits, see the *CICS Customization Guide*.)

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 in the remote CICS (for example, a deadlock timeout causes the mirror to be abended with a code of ATSC).

In these situations, the application program is also abended, but with an abend code of ATNI (for ISC connections) or AZI6 (for MRO connections). The actual error condition is logged by CICS in an error message sent to the CSMT destination. Any EXEC CICS HANDLE ABEND command issued by the application cannot identify the original cause of the condition and take explicit corrective action (which might have been possible if the resource had been 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 abends with the normal deadlock abend code (ADLD).

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 18. Application programming for CICS DPL

This chapter contains the following topics:

Introduction to DPL programming

CICS distributed program link (DPL) allows you to link to server programs located on a remote system. A client program running in a CICS Transaction Server for VSE/ESA Release 1 region can link to one or more server programs running in remote CICS regions. The remote regions may or may not be CICS Transaction Server for VSE/ESA systems; (they could be, for example, CICS for OS/2 or CICS/400 systems). See Chapter 1, "Introduction to CICS intercommunication" on page 3 for a list of systems with which CICS Transaction Server for VSE/ESA Release 1 can communicate.

DPL programs can be written in PL/I, C, COBOL, or assembler language.

As Chapter 5, "CICS distributed program link" on page 33 indicates, there are two sides (programs) involved in DPL: the client program and the server program. To implement DPL, there are actions that each program must take. These actions are described below.

The client program

If you are writing a client program to link to a server program in a remote system, you code it in much the same way as if the server program were on the local system.

Your client program can run in the CICS intercommunication environment and make use of intercommunication facilities without being aware of the location of the server program being linked to. The location of the server program is specified in the program resource definition. Optionally, you can use the SYSID option on the EXEC CICS LINK command to select the system on which the command is to be executed. In this case, the program resource definition in the local system is not referenced, unless the SYSID option names the local system.

When your client program issues an EXEC CICS LINK command against a server program, CICS ships the request to the remote system, where a mirror transaction is initiated. The mirror transaction executes the LINK request on your behalf, thereby causing the server program to run. When the server program issues an EXEC CICS RETURN command, the mirror transaction returns any communication area data to your client program. The mirror transaction is like a remote extension of your application program. For more information about this mechanism, read Chapter 5, "CICS distributed program link" on page 33.

Although the same command is used to access both local and remote server programs, there are restrictions that apply when the server program is remote.

Also, some errors that do not occur in single systems can arise when DPL is being used. For these reasons, you should always find out whether the server program to which your client program links is remote. If there is any possibility of the server program being remote, the client program should include the additional checks for the exception conditions that can be returned by a remote server program.

The server program

If the server program fails, the ABEND condition and an abend code are returned to the client program. The client transaction therefore also terminates abnormally, unless it has issued the EXEC CICS HANDLE ABEND command before issuing the EXEC CICS LINK command.

If the server program was started by an EXEC CICS LINK command that specified the SYNCONRETURN option, it is able to issue a syncpoint. If it does, this does **not** commit changes made by the client program. For changes to be committed across the distributed unit of work, the client program must issue the syncpoint. The client program can also backout changes across the distributed unit of work, provided that the server program has not already committed its changes.

The server program can find out how it was started, and therefore whether it is allowed to issue independent syncpoint requests, by issuing the EXEC CICS ASSIGN STARTCODE command. This command returns the following values relevant to a DPL server program:

- ‘D’ if the program was started by a LINK request **without** the SYNCONRETURN option, and cannot therefore issue EXEC CICS SYNCPOINT requests.
- ‘DS’ if the program was started by a LINK request **with** the SYNCONRETURN option, and can therefore issue EXEC CICS SYNCPOINT requests. However, the server program need not issue a syncpoint request explicitly, because CICS takes a syncpoint as soon as the server program issues the EXEC CICS RETURN command.
- Values other than ‘D’ and ‘DS’ if the program was not started by a remote LINK request.

DPL exceptional conditions

LINK requests that are shipped to a remote system can raise any of the exceptional conditions for the command that can occur if the server program is local. In addition, there are some conditions that apply only when the server program is remote.

Remote system not available

When the remote system is unavailable, the SYSIDERR condition can be raised in the client program for exactly the same reasons as described for function shipping on page 179 (except that the XISCONA global user exit is not invoked for DPL requests).

The default action for the SYSIDERR condition is to terminate the task abnormally.

Server's work backed out

If the client program issues the EXEC CICS LINK command with the SYNCONRETURN option, the mirror program issues a syncpoint as soon as the server program terminates successfully. It is possible for this syncpoint to fail. If this happens, the ROLLEDBACK condition is returned to the client program. The work done by the server program will also be backed out, *unless the server program has already committed the work by issuing its own syncpoint request*.

Multiple links to the same server region

When a client program issues an EXEC CICS LINK command *with* the SYNCONRETURN option, the mirror transaction terminates as soon as control is returned to the client program. It is therefore possible for the client program to issue a subsequent LINK command to the same server region.

However, when a client program issues an EXEC CICS LINK command *without* the SYNCONRETURN option, the mirror transaction is suspended pending a syncpoint request from the client region. The client program can issue subsequent LINK commands to the same server region as long as the SYNCONRETURN option is omitted and the TRANSID value is not changed. A subsequent LINK command with the SYNCONRETURN option or with a different TRANSID value will be unsuccessful unless it is preceded by an EXEC CICS SYNCPOINT command.

Note: Similar considerations apply if the client program sends function shipping requests to the server region, and the mirror for the function shipping request is suspended. For example:

```
EXEC CICS LINK PROGRAM('PGA') SYSID(SERV)
EXEC CICS SYNCPOINT
EXEC CICS READQ TS QUEUE('RQUEUE') SYSID(SERV)
EXEC CICS LINK PROGRAM('PGB') SYSID(SERV) TRANSID(TRN1)
```

The last EXEC CICS LINK command fails if, for example, MROLRM=YES is specified in the CICS server region (SERV). This is because the mirror used for the EXEC CICS READQ TS command is still around. For the above sequence of commands to work, the client program must issue a syncpoint after the EXEC CICS READQ TS command; alternatively, you could set the MROLRM system initialization parameter to 'NO' in the server region. For detailed information about using DPL and function shipping requests in the same program, see the *CICS Application Programming Guide*.

These errors are indicated by the INVREQ condition. An accompanying RESP2 value of 14 indicates that a syncpoint is necessary before the failed LINK command can be successfully attempted. A RESP2 value of 15 indicates that the TRANSID value is different from that of the linked mirror transaction. A RESP2 value of 16 indicates that a TRANSID value of spaces (blanks) was specified on the LINK command.

Mirror transaction abend

If the mirror program (as opposed to the server program) abends or the session with the server region fails, the TERMERR condition is returned to the client program.

Chapter 19. Application programming for the external CICS interface

The external CICS interface (EXCI) is a special form of DPL. It enables a non-CICS client program running in a VSE/ESA address space to link to a server program running in a CICS Transaction Server for VSE/ESA Release 1 system. For an overview of EXCI, see Chapter 6, “The external CICS interface” on page 39.

The chapter contains the following topics:

- “The VSE/ESA client program”
- “The CICS server program” on page 188
- “Customization” on page 188
- “Sample applications” on page 189

The VSE/ESA client program

The external CICS interface provides two forms of API, both for use by VSE/ESA client programs: the external CICS interface CALL API and an EXEC CICS API.

The EXCI CALL API

The external CICS interface CALL API consists of six commands that allow non-CICS programs running under VSE/ESA to allocate and open sessions to a CICS system, and to issue DPL requests on these sessions.

You must use the CALL API if you want your client program to use an EXCI “specific connection”. A specific connection is an MRO link on which all the sessions are dedicated to a single user (the distinction between a client program and a “user” is explained shortly). The alternative is to use a generic connection, on which the sessions are shared by multiple users.

The commands invoke the external CICS interface via an application stub module, DFHXCSTB, which you must linkedit with your non-CICS program.

All possible return codes are contained in a copybook which you must include in the program source of your external, non-CICS program. The copybooks supplied are as follows:

- DFHXCRCD (assembler)
- DFHXCRCH (C)
- DFHXCRCL (PL/I)
- DFHXCRCO (COBOL)

The six commands are:

INITIALIZE_USER
Initializes the user environment, including obtaining authority to use IRC facilities. The environment is created for the lifetime of the VSE/ESA task, so needs to be issued only once per user per VSE/ESA task. Further commands from this user must be issued under the same VSE/ESA task.

Note: A *user* is a program that has issued an Initialize_user request (or for which an Initialize_user request has been issued), with a unique name per VSE/ESA task. For example:

- A simple client program running under VSE/ESA could be a single user of the external CICS interface.
- A client program running under VSE/ESA could open several pipes and issue external CICS interface calls over them sequentially, on behalf of different vendor packages. From the viewpoint of the client program, each of the packages is a user, identified by a unique user name. Thus a single client program can operate on behalf of multiple users.
- A program running under VSE/ESA could attach several VSE/ESA tasks, under each of which a vendor package issues external CICS interface calls on its own behalf. Each package is a client program in its own right, and runs under its own VSE/ESA task. Each is also a user, with a unique user name.

ALLOCATE_PIPE
Allocates a single session, or pipe, to a CICS system.

OPEN_PIPE
Causes IRC to connect an allocated pipe to a receive session of the appropriate connection defined on the CICS system named in the Allocate_Pipe command. The appropriate connection is either:

- The EXCI connection defined with a NETNAME value equal to the user name on the Initialize_User command (that is, you are using a specific connection, dedicated to this user)

or

- The EXCI connection defined as generic

DPL call
Issues a DPL request across an open pipe connected to the CICS system on which the server (or target) application resides. The command is synchronous and the VSE/ESA task waits for a response from CICS. Once a pipe has been opened, any number of DPL requests may be issued before the pipe is closed. The server program sees the link request as a standard EXEC CICS LINK request from a remote system.

CLOSE_PIPE
Disconnects an open pipe from CICS. The pipe remains in an allocated state, and its tokens remain valid for use by the same user. To reuse a closed pipe, the client program must first reissue an Open_Pipe command against the pipe.

DEALLOCATE_PIPE
Deallocates a pipe from CICS.

The EXCI EXEC API

The external CICS interface EXEC CICS API provides a single command which performs all six functions of the external CICS interface API in one invocation.

Format

```
►►──LINK──PROGRAM(name)──RETCODE(data-area)──SYNCONRETURN──────────────────────►

►──┬─────────────────────────────────────────────────────────────────┬──────────►
   └─COMMAREA(data-area)──LENGTH(data-value)──┬──────────────────────┬─┘
                                              └─DATALENGTH(data-value)─┘

►──┬───────────────┬──┬────────────────┬──►◄
   └─APPLID(name)─┘  └─TRANSID(name)─┘
```

Conditions:
INVREQ, LENGERR, LINKERR, NOTAUTH, PGMIDERR, ROLLEDBACK, SYSIDERR, WARNING

All the parameters are as on a CICS-to-CICS DPL request, except for APPLID which specifies the applid of the target CICS system, as opposed to the system name in a CICS-to-CICS DPL call. Also, RETCODE specifies a 20-byte area into which the external CICS interface places return code information. As for the CALL API, SYNCONRETURN is mandatory. All commands use a generic connection.

Programs that use the EXEC CICS API must be translated using the CICS translator, with the 'EXCI' translator option specified.

Choosing between the CALL API and the EXEC API

As illustrated in the various versions of the CICS-supplied sample client program (described in "Sample applications" on page 189), you can use the CALL API and the EXEC CICS LINK command in the same program, to perform separate requests. However, it is unlikely that you would want to do this in a production program.

Each form of the external CICS interface has its particular benefits and drawbacks:

- For low-frequency or one-shot usage, you are recommended to use the EXEC CICS LINK command.

  It is easier to code, and therefore less prone to programming errors.

  However, each invocation of an EXEC CICS LINK command causes the external CICS interface to perform all the functions of the CALL interface, which may result in an unnecessary overhead. Also, your program is limited to using a generic connection.

- For multiple or frequent DPL requests from the same batch client program, you are recommended to use the EXCI CALL API.

  This is more efficient, because you need only perform the Initialize_User and Allocate_Pipe commands once, at or near the beginning of your program, and the Deallocate_Pipe once on completion of all DPL activity. In between these functions, you can open and close the pipe as necessary, and while the pipe is opened, you can issue as many DPL calls as you want. Your program can use either a generic or specific connection.

  However, to use the CALL API, you need an understanding of pipe management, so that you can use CICS resources efficiently. It would be undesirable, for example, for one client program to open many pipes on a generic connection and then not use them, thus locking out other potential users. (In addition, this would prevent CICS from closing down its IRC facility and perhaps even prevent CICS itself from closing down normally.)

For programming information about the external CICS interface APIs, see the *CICS External Interfaces Guide* manual.

The CICS server program

CICS server programs invoked by VSE/ESA clients are permitted to use the same subset of EXEC CICS commands as those invoked by CICS clients via DPL requests. (The restricted commands are listed on page 36, and amongst the programming information in the *CICS Application Programming Reference* manual.) You may therefore be able to use server programs written for CICS-to-CICS DPL.

Customization

This section describes how you can vary the ways in which your EXCI programs operate.

Setting EXCI parameters

You can use the DFHXCOPT macro to specify a number of EXCI parameters. For example, you can specify:

- Whether EXCI messages are issued in mixed or upper case.
- The time interval for which EXCI waits for a DPL command to complete. (A typical value might be about 10 minutes.)
- Whether EXCI internal tracing is required, and at what level.

For full details of the DFHXCOPT macro, see the *CICS External Interfaces Guide* manual.

Routing external interface requests

The CICS user-replaceable program, DFHXCURM, is invoked in the non-CICS client environment during ALLOCATE_PIPE processing, and after retryable error conditions.

You could use DFHXCURM to change the specified CICS APPLID during ALLOCATE_PIPE processing, in order to route the request to another CICS system.

If DFHXCURM is invoked after a retryable error, it is able to store information regarding CICS availability. This information can be used on its next invocation for ALLOCATE_PIPE processing, to decide to which CICS system to route the request.

For details of the default version of DFHXCURM, and of how to replace it with your own version, see the *CICS External Interfaces Guide* manual.

Sample applications

To help you write programs that use the external CICS interface, a sample VSE/ESA client application program (one version in each of the assembler, COBOL, C, and PL/I programming languages) and a sample CICS server application program (in assembler only) are provided.

The sample programs are included on the VSE/ESA production library, in source and processable form for assembler language, and in source form only for COBOL, PL/I, and C. Each version of the client application has basically the same function, but programming methods vary somewhat according to the language used. Table 15 lists the available programs.

Table 15. Sample programs for the external CICS interface

Type	Language	Identifier	How supplied
VSE/ESA client	Assembler	DFH$AXCC	Source and processable
VSE/ESA client	COBOL	DFH0CXCC	Source
VSE/ESA client	PL/I	DFH$PXCC	Source
VSE/ESA client	C/370™	DFH$DXCC	Source
CICS server	Assembler	DFH$AXCS	Source and processable

The client samples show you how to code a simple VSE/ESA client program using both the CALL and EXEC CICS forms of the API. Each is divided into three separate sections. The first section performs a single EXEC CICS LINK request to the target CICS system to inquire on the state of the target sample file, FILEA. If the file exists, and is in a suitable state, processing continues to sections two and three, which together form a complete example of the use of the CALL API.

The second section initiates a specific MRO connection to the target CICS system and, once the pipe is open, performs a series of calls that each retrieve a single sequential record from the sample file, until no more records are available.

The third section is a simple routine to close the target sample file once processing of the data is complete, and to terminate the MRO connection now that the link is no longer required.

Parameters in the samples that refer to systems and userids must be changed before running the programs. For further details of the EXCI sample applications, see the *CICS External Interfaces Guide* manual.

Chapter 20. 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 EXEC CICS START or RETRIEVE commands is also applicable to CICS-to-IMS communication.

A description of the concepts of asynchronous processing is given in Chapter 7, "Asynchronous processing" on page 41. It is assumed that you are familiar with the concepts of CICS interval control. For programming information about the use of EXEC CICS commands for interval control, see the *CICS Application Programming 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.

Generally, the transaction has been defined as 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 EXEC CICS 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 179).

If NOCHECK is specified, no conditions are raised as a result of the remote execution of the START command. SYSIDERR, however, still occurs if no link to the remote system is available, unless the system programmer has arranged for local queuing of start requests (see "Local queuing of EXEC CICS START commands" on page 47).

Retrieving data associated with a remotely-issued start request

The EXEC CICS 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 EXEC CICS RETRIEVE command apply.

Chapter 21. 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.

Things to watch out for

The program can use either command level or macro level, and can be written in PL/I, COBOL, C, or assembler language. This choice may, of course, be restricted by the terminal or session type: basic APPC conversations, for example, must use CICS command level and be written in C or assembler language.

Note: Information on macro-level programs is intended primarily for:

- Migrating existing programs to a transaction routing environment
- Transaction routing to a CICS Version 1 or CICS Version 2 system.

It is strongly recommended that command level be used for new applications. You cannot run macro-level programs on a CICS Transaction Server for VSE/ESA Release 1 system, nor can you use the C language to write macro-level programs.

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 directs output only to the operators signed on at terminals that are owned by the system in which the transaction is executing.

The mapset name specified in the most recent EXEC CICS SEND MAP command is saved in the TCTTE. For a routed transaction, this means that the mapset name is saved in the surrogate TCTTE and, when the routed transaction terminates, the most recently used mapset name is passed in a DETACH sequence from the AOR to the TOR.

Similarly, when a routed transaction is initiated, the most recently used mapset name is passed in an ATTACH sequence from the TOR to the AOR.

From CICS Transaction Server for VSE/ESA Release 1 onwards, the *map* name is supported in the same way as the *mapset* name. However, earlier CICS releases have no knowledge of map names being passed in ATTACH and DETACH sequences. When sending an ATTACH sequence, CICS Transaction Server for VSE/ESA Release 1 systems set the map name to null values in the "real" TCTTE, in case the AOR is unable to return a map name in the DETACH sequence. In other words, the TCTTE in the TOR contains a null value for the saved map name, rather than a potentially incorrect name.

The names of mapsets and maps saved in the TCTTE can be both queried and updated by the MAPNAME and MAPSETNAME options of the INQUIRE TERMINAL and SET TERMINAL commands. For details of these options, see the *CICS System Programming Reference* manual.

Here are some points to remember (they apply to non-routed as well as to routed transactions):

- Map and mapset names are only remembered when used in EXEC CICS SEND MAP commands and the principal facility is a 3270 device.
- The last map sent may have been partially or completely removed from the device buffer as the result of an EXEC CICS SEND CONTROL, SEND TEXT or terminal control SEND command, or by the operator hitting the CLEAR key. Thus the map and mapset names returned by an INQUIRE command do not necessarily match the current content of the device buffer.
- If the last EXEC CICS SEND MAP command specified the ACCUM option, the map name saved will either only represent a portion of a composite display or may not have been sent to the device.
- If the last EXEC CICS SEND MAP command specified the SET or PAGING option the map name saved may not have been sent to the device.
- The mapset name returned by an INQUIRE command may contain a terminal, or alternate, suffix which must be removed if the name is used in a subsequent EXEC CICS SEND MAP command.

Pseudoconversational transactions

A routed transaction requires the use of an interregion or intersystem (APPC) 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.

Take care 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 runs is represented by a terminal control table table entry (TCTTE). This TCTTE, called a **surrogate TCTTE**, is in many respects a copy of the “real” terminal’s TCTTE in the terminal-owning region. CICS releases the surrogate TCTTE when the transaction terminates. Subsequent tasks run using new copies of the real terminal’s TCTTE.

If your program needs to discover terminal-related information, you should bear in mind the following:

- Your program should not test fields in the TCTTE directly: it should test instead the equivalent fields in the EXEC interface block (EIB).
- If the new task is started by ATI, the contents of certain terminal-related fields in the EIB are unpredictable. Prior to CICS Transaction Server for VSE/ESA

Release 1, these included EIBAID and EIBSCON. However, in CICS Transaction Server for VSE/ESA Release 1, EIBAID, which contains the attention identifier, is always set to zeros at the start of a session. In releases before CICS Transaction Server for VSE/ESA Release 1, it may contain either zeros or residual data from a previous session. The effect of this is that, if you are transaction routing from a CICS Transaction Server for VSE/ESA Release 1 TOR to a pre-CICS Transaction Server for VSE/ESA Release 1 AOR, the content of EIBAID at commencement of the task is unpredictable. This problem does not apply to routing in the reverse direction.

Using the EXEC CICS ASSIGN command in the AOR

You may find that two of the options of the EXEC CICS ASSIGN command cause an unexpected reaction or return unexpected values.

PRINSYSID

This option returns the sysid of the principal facility to the transaction. It requires that this facility be an MRO, an LUTYPE6.1, or an APPC session. For transaction routing, this is further restricted to an APPC session, because neither of the other session types can be made principal facility to a routed transaction. Here, the value returned is the name of the remote connection or terminal defined in this system. If the connection or terminal has been shipped, the name is the original name defined in the TOR. If the principal facility is not an APPC session, the INVREQ condition is raised.

With this option, an application can obtain the connection name of an APPC device. By making this the sysid of an ALLOCATE command, further sessions to the same device can be established.

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 several sources, depending on how you specified your security requirements.

As Table 16 on page 196 shows, CICS returns the following values:

- If the connection is defined with the ATTACHSEC(LOCAL) option, and SEC=YES is specified in the AOR's system initialization parameters, CICS returns:
  - For ISC connections, either:
    1. The USERID from the session definition, if this is specified
    2. The SECURITYNAME value from the connection definition
  - For MRO connections, the ESM userid of the TOR.
- If the connection is defined with the ATTACHSEC(LOCAL) option, and SEC=NO is specified in the AOR's system initialization parameters, CICS returns the DFLTUSER value from the AOR.
- If the connection is defined with the ATTACHSEC(IDENTIFY) option (or, for APPC connections, the VERIFY, PERSISTENT, or MIXIDPE option), and SEC=YES is specified in the TOR's system initialization parameters, CICS returns the userid sent at attach.

- If the connection is defined with the ATTACHSEC(IDENTIFY) option (or, for APPC connections, the VERIFY, PERSISTENT, or MIXIDPE option), and SEC=NO is specified in the TOR's system initialization parameters, CICS returns the DFLTUSER value from the TOR.

Table 16. Values returned by the USERID option of EXEC CICS ASSIGN, for routed transactions

TOR's DFHSIT SEC=	**ATTACHSEC value in CONNECTION definition**		
	IDENTIFY VERIFY PERSISTENT MIXIDPE	**LOCAL**	
		AOR's DFHSIT SEC=YES	**AOR's DFHSIT SEC=NO**
YES	Userid sent at attach	ISC 1. USERID of session 2. SECURITYNAME of connection MRO ESM userid of TOR	DFLTUSER of AOR
NO	Userid sent at attach (DFLTUSER of TOR)		

Chapter 22. 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, refer to 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 chapter contains the following topics:

Designing 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. This is illustrated in Figure 63 on page 198.

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

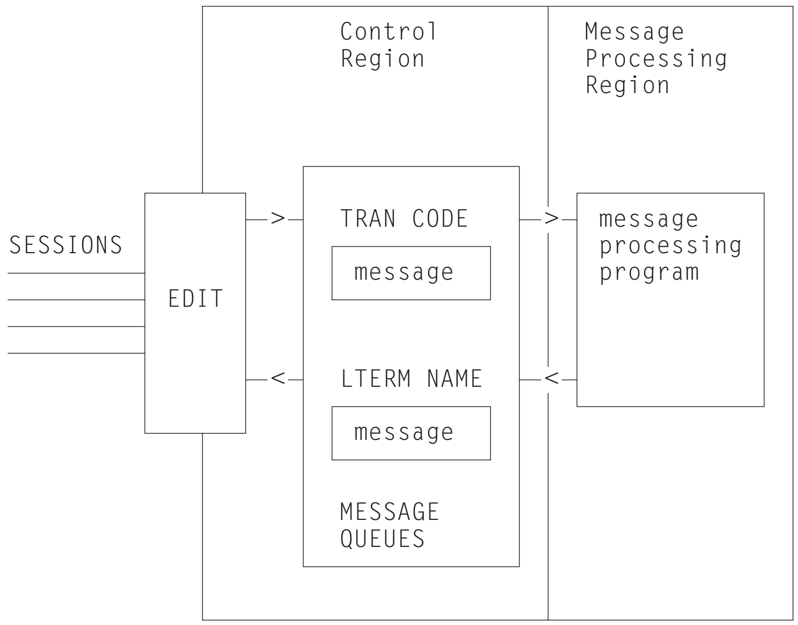


Figure 63. Basic IMS message queuing

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 two-byte length field, as shown here.

```
┌──┬──────────────┬──┬──────────────┐
│LL│  data        │LL│  data        │
└──┴──────────────┴──┴──────────────┘
<── record 1 ──────><─── record 2 ───>
```

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.

```
┌──────────────────────────────────┐
│           data                   │
└──────────────────────────────────┘
<──── multiple SNA RUs ─────────────>
```

In CICS, the I/O area contains a complete message.

Forms of intersystem communication with IMS

There are three forms of CICS-to-IMS communication that must be considered:

1. Asynchronous processing using EXEC CICS START and RETRIEVE commands
2. Asynchronous processing using EXEC CICS SEND LAST and RECEIVE commands
3. Distributed transaction processing (that is, synchronous processing) using EXEC CICS SEND and RECEIVE commands.

The basic differences between these forms of communication are described in Chapter 7, “Asynchronous processing” on page 41 and Chapter 9, “Distributed transaction processing” on page 71.

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. For information about the forms of communication that IMS supports when it is the back-end system, see the *IMS Programming Guide for Remote SNA Systems*.

When IMS is the front-end system, it always uses asynchronous processing (corresponding to the CICS START and 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:

1. The START and RETRIEVE interface
2. The SEND and RECEIVE interface

The START and RETRIEVE interface

For programming information about the EXEC CICS START and RETRIEVE "interval control" commands, see the *CICS Application Programming Reference* manual. The applicable forms of these commands, together with the specific meanings of the command options in a CICS-to-IMS intersystem communication environment, are given later in this section.

CICS front end

When CICS is the front-end system, you can use EXEC CICS START and RETRIEVE commands to process IMS nonresponse mode and nonconversational transactions, message switches, and the IMS /DIS, /RDIS, and /FOR operator commands.

Note: When you issue the operator commands mentioned above, unless you send change direction (CD), IMS expects you to request definite response. You must do this by coding the PROTECT option on the START command.

The general command sequence for your application program is shown in Figure 64.

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 must 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 does not carry the syncpoint-indicator unless PROTECT was specified on the START command.

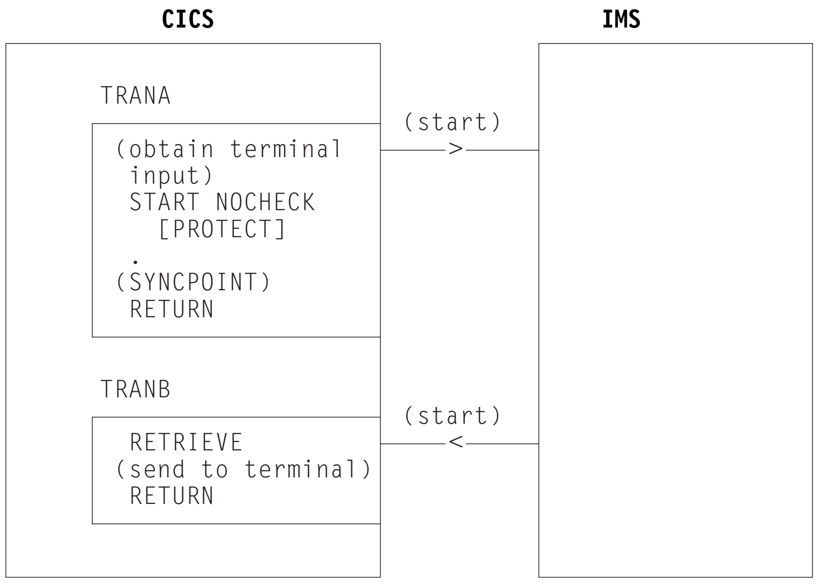


Figure 64. START and RETRIEVE asynchronous processing–CICS front end

Although CICS allows an application program to issue multiple START NOCHECK commands without intervening syncpoints (see “Deferred sending of START requests with NOCHECK option” on page 46), this technique is not recommended for CICS-to-IMS communication.

IMS sends the reply by issuing a start request that 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 clear the start request, and then waits for the reply by issuing a RETRIEVE command with the WAIT option. The terminal is held by the transaction during this time, and CICS returns 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 retrieves any outstanding data intended 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 65 on page 202.

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.

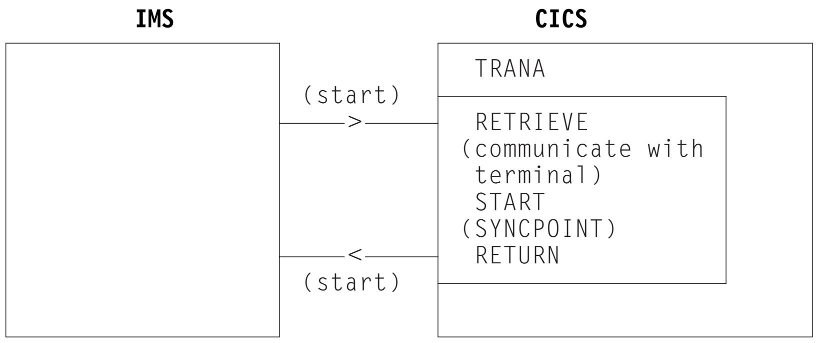


Figure 65. RETRIEVE and START asynchronous processing – IMS front end

The EXEC CICS START command

This section shows the format of the EXEC CICS 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 and PROTECT options must be specified.

```
EXEC CICS START TRANSID(name)
     [SYSID(name)]
     [FROM(data-area) LENGTH(value)]
     [TERMID(name)]
     [RTRANSID(name)]
     [RTERMID(name)]
      NOCHECK
      PROTECT
     [FMH]
```

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 the installed definition of a “remote” transaction. In this case, the SYSID option is not used. The definition of the remote transaction must name the required IMS editor in the RMTNAME option, which can be up to eight characters long.

SYSID(name)
Specifies the name of the remote IMS system. This is the name that is specified by the system programmer in the CONNECTION option of the DEFINE CONNECTION command that defines the link to the remote system. You need this option only if you are required to name the remote system explicitly.

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 that defines the remote IMS system (see Chapter 12, “Defining links to remote systems” on page 93).

LENGTH(value)
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. PROTECT is mandatory.

FMH
Specifies that the user data to be passed to the started task contains function management headers. This option is not normally used.

The RETRIEVE command

This section shows the format of the RETRIEVE command that is used to 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.

SET(pointer-ref)
Specifies the pointer reference to be set to the address of the data retrieved from IMS.

LENGTH(data-area)
Specifies the halfword binary length of the retrieved data.

For a RETRIEVE command with the INTO option, this 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, this must be a data area. On completion of the retrieval operation, the data area is set to the length of the data.

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 option of a subsequent START command.

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 option of the START command used to send the reply.

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 is raised if no data is available. If WAIT is specified, the ENDDATA condition is 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 and 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 later in this chapter.

The general command sequence for asynchronous SEND and RECEIVE application programs is shown in Figure 66 on page 205.

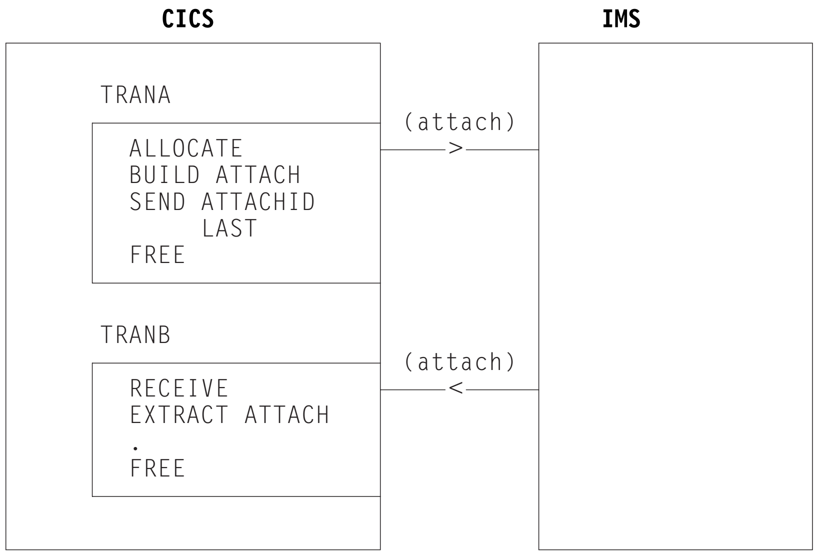


Figure 66. SEND and RECEIVE asynchronous processing – CICS front end

Distributed transaction processing

This section describes application programming for CICS-to-IMS distributed transaction processing (DTP). For further information about DTP, see the *CICS Distributed Transaction Programming Guide*.

CICS commands for CICS-to-IMS sessions

The EXEC CICS 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 is 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 and 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.

ISSUE SIGNAL SESSION(name) – used by a transaction that is in receive state to request an invitation to send (change-direction) from IMS.

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 and 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 client, or 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.

If CICS cannot find the named system, or no sessions are available, it raises the SYSIDERR condition. If CICS cannot find the named session or the session is out of service, CICS raises the SESSIONERR condition.

The PROFILE option allows you to specify a 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 uses the default profile DFHCICSA. This profile specifies INBFMH(ALL), which means that incoming function management headers are passed to your program and 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 a delay in acquiring a session is acceptable or not is dependent 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 parameter 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 has already been allocated when the transaction starts. You can omit the SESSION parameter from commands that relate to the principal facility. If, however, you want 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 need to know the meaning of the information that it carries.

The format of the BUILD ATTACH command is:

```
BUILD ATTACH
   ATTACHID(name)
  [PROCESS(ISCEDT|BASICEDT|name)]
  [RESOURCE(name)]
  [RPROCESS(name)]
  [RRESOURCE(name)]
  [QUEUE(name)]
  [IUTYPE(0|data-value)]
  [DATASTR(0|data-value)]
  [RECFM(data-value)]
```

The parameters 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 eight 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 (up to eight characters) 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.

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 eight 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.

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 data stream)
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 is 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 two-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 an 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 and 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 parameters 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'– (zero)
8–11	0000 – (user-defined data stream)
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' – (zero)
8–15	X'00' – multiple RU chains, MFS autopaged output
	X'01' – single RU chains, MFS nonpaged 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' – (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. For programming information about these commands, see the *CICS Application Programming 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 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 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 has been deferred, and the syncpointing activity causes the final transmission to occur with an added syncpoint request. The conversation is thus automatically involved in the syncpoint.

Freeing the session

The command used to free the session has the following format:

```
FREE SESSION(conversation-name)
```

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 some 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)

For programming information about the EXEC interface block (EIB), see the *CICS Application Programming 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, refer to “Command sequences for CICS-to-IMS sessions” on page 214.

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 command. You must acquire this name immediately after issuing the ALLOCATE command.

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 about the use of these fields is given in “Command sequences for CICS-to-IMS sessions” on page 214.

EIBRECV
Indicates that the conversation is in receive state and that the normal continuation is to issue a RECEIVE command.

EIBCOMPL
This field is used in conjunction with the RECEIVE NOTRUNCATE command; it is set when there is no more data available.

EIBSYNC
Indicates that the application must take a syncpoint or terminate.

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.

Information 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 about this subject is given under “Defining communication profiles” on page 163.

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 ensure that each transaction takes account of the requirements of the other. This helps to avoid errors during program development.

Conversation states

The protocols are based on the concept of several separate states. These states apply only to the particular conversation, not to your entire application program. In each state, there is a choice of commands that might most reasonably be issued. After the command has been issued, fields in the EIB can be tested to learn 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, when another set of commands becomes appropriate.

The states that are defined for this section are:

- State 1 – Session not allocated
- State 2 – Send state
- State 3 – Receive pending after SEND 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 starts 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. Here, the session is already allocated, and the transaction is in state 2. For transactions of this type, you must immediately obtain the session name from EIBTRMID so that you can name the session explicitly on later 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 figures help 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, make them in the order indicated.

The combination of the command issued and a particular positive test result lead to a new, resultant state, shown in the final column.

Other tests

The tests that are shown in the figures are those that are significant to the state of the conversation. Tests for other conditions that may 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

Figure 67. State 1 – session not allocated

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.

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 abends if SYNCPOINT fails)	2
FREE Equivalent to: SEND LAST WAIT FREE	—	1

Figure 68. State 2 – send state

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” on page 207.

STATE 3 CICS-to-IMS CONVERSATIONS RECEIVE PENDING after SEND INVITE		
Commands you can issue	What to test	New State
SYNCPOINT	(transaction abends if SYNCPOINT fails)	4

Figure 69. State 3 – receive pending after SEND 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

Figure 70. State 4 – receive state

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 saves the remaining data for retrieval by later RECEIVE NOTRUNCATE commands. EIBCOMPL is set when the last part of the data is passed back. If the NOTRUNCATE option is not specified, over-length data is indicated by the LENGERR condition, and the remaining data is discarded by CICS.

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 71. 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 72. 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 73. State 7 – free session

Part 5. Performance

Table 17. Performance road map

If you want to...	**Refer to...**
Control the length of intersystem queues	Chapter 23, “Intersystem session queue management” on page 221
Delete redundant shipped terminal definitions	Chapter 24, “Efficient deletion of shipped terminal definitions” on page 225

Chapter 23. Intersystem session queue management

This chapter describes how to control the number of queued requests for sessions on intersystem links (allocate queues).

Note: This chapter describes how to control queues for sessions on established connections. The specialized subject of using local queuing for function-shipped EXEC CICS START NOCHECK requests is described in “Local queuing of EXEC CICS START commands” on page 47.

Overview

In a perfect intercommunication environment, queues would never occur because work flow would be evenly distributed over time, and there would be enough intersystem sessions available to handle the maximum number of requests arriving at any one time. However, in the real world this is not the case, and, with peaks and troughs in the workload, queues do occur: queues come and go in response to the workload. The situation to avoid is an unacceptably high level of queuing that causes a bottleneck[10] in the work flow between interconnected CICS regions, and which leads to performance problems for the terminal end-user as throughput slows down or stops. This abnormal and unexpected queuing should be prevented, or dealt with when it occurs: a “normal” or optimized level of queuing can be tolerated.

For example, function shipping requests between CICS application-owning regions and connected file-owning regions can be queued in the issuing region while waiting for free sessions. Provided a file-owning region deals with requests in a responsive manner, and outstanding requests are removed from the queue at an acceptable rate, then all is well. But if a file-owning region is unresponsive, the queue can become so long and occupy so much storage that the performance of connected application-owning regions is severely impaired. Further, the impaired performance of the application-owning region can spread to other regions. This condition is sometimes referred to as “sympathy sickness”, although it should more properly be described simply as intersystem queuing, which, if not controlled, can lead to performance degradation across more than one region.

Methods of managing allocate queues

The following sections describe three methods for managing allocate queues, the first two straightforward, the third more sophisticated.

Using only connection definitions

For those intersystem links for which simple control requirements are adequate (perhaps those that carry non-critical traffic), you can specify the QUEUELIMIT and MAXQTIME options on the CONNECTION resource definitions.

[10] Bottleneck. A condition or state of the connection that slows down the rate of flow of traffic across the intercommunication link.

QUEUELIMIT defines the maximum number of allocate requests that CICS is to queue while waiting for free sessions on the connection. You can specify a number in the range 0 (that is, do not queue any requests) through 9999; or that all requests should be queued, if necessary, no matter what the length of the queue.

MAXQTIME defines the approximate time for which allocate requests should queue for free sessions on a connection that appears to be unresponsive. Its value is used only if a queue limit is specified on QUEUELIMIT, and if that limit is reached. You can specify a time in the range 0 (that is, the queue should be purged immediately after receipt of an allocate request that would exceed the queue limit) through 9999 seconds; or that requests should be queued for as long as necessary.

When an allocate request is received that would cause the QUEUELIMIT value to be exceeded, CICS calculates whether the queue's rate of processing means that a new request would take longer to satisfy than the maximum queuing time. If it would, CICS purges the queue. No further queuing takes place until the connection has freed a session. At this point, queuing begins again.

For information about the QUEUELIMIT and MAXQTIME options of the CEDA DEFINE CONNECTION command, see the *CICS Resource Definition Guide*.

Using the NOQUEUE option

A further method of controlling *explicit* allocate requests is to specify the NOQUEUE|NOSUSPEND option of the EXEC CICS ALLOCATE command. However, while this enables you to control specific requests, it takes no account of the state of the queue at the time the requests are issued. And it is of no use in controlling *implicit* allocate requests (where the session request is instigated by, for example, a function shipping request). For programming information about API options, see the *CICS Application Programming Reference* manual.

Using the XZIQUE global user exit

You can also control the queuing of allocate requests through an XZIQUE global user exit program. This allows you much more flexibility than simply setting a queue limit on the connection.

The XZIQUE exit enables you detect queuing problems (bottlenecks) early. Note that it provides more function than the XISCONA global user exit, which is invoked only for function shipping requests (including function shipped EXEC CICS START requests used for asynchronous processing). XZIQUE is invoked for transaction routing, DPL, asynchronous processing, and distributed transaction processing requests, as well as for function shipping. Compared with XISCONA, it receives more detailed information on which to base its decisions.

XZIQUE enables allocate requests to be queued or rejected, depending on the length of the queue. It also allows a connection on which there is a bottleneck to be terminated and then re-established.

Interaction with the XISCONA exit

There is no interaction between the XZIQUE and XISCONA global user exits. If you enable both exits, XISCONA and XZIQUE could both be invoked for function shipping requests, which is not recommended. You should ensure that only one of these exits is enabled. Because of its increased functionality and greater flexibility, it is recommended that you use XZIQUE rather than XISCONA.

When the XZIQUE exit is invoked

The XZIQUE global user exit is invoked, if it is enabled, at the following times:

- Whenever CICS tries to acquire a session with a remote system and there is no free session available. It is invoked whether or not you have specified the QUEUELIMIT option on the CONNECTION definition, and whether or not the limit has been exceeded. It is not invoked if the allocate request specifies NOQUEUE or NOSUSPEND.

  Requests for sessions can arise in a number of ways, such as explicit EXEC CICS ALLOCATE commands issued by DTP programs, or by transaction routing or function shipping requests.
- Whenever an allocate request succeeds in finding a free session, after the queue on the connection has been purged by a previous invocation of the exit program. In this case, your exit program can indicate that CICS is to continue processing normally, resuming queuing when necessary.

Uses of an XZIQUE global user exit program

When the exit is enabled, your XZIQUE global user exit program is able to check on the state of the allocate queue for a particular connection in the local system. Information is passed to the exit program in a parameter list, that is structured to provide data about non-specific allocate requests, or requests for specific modegroups, depending on the session request. Non-specific allocate requests are for MRO, LU6.1, and APPC sessions that do not specify a modegroup.

Using the information passed in the parameter list, your global user exit program can decide whether CICS is to:

- Queue the allocate request (only possible if the queue limit has not been reached).
- Reject the allocate request.
- Reject this allocate request and purge all queued requests for the connection.
- Reject this allocate request and purge all queued requests for the modegroup.

Your exit program could base its decision on, for example:

- The length of the allocate queue.
- Whether the number of queued requests has reached the limit set by the QUEUELIMIT option. If the queue limit has not been reached, you may decide to queue the request.

- The rate at which sessions are being allocated on the connection. If the queue limit has been reached but session allocation is acceptably quick, you may decide to reject only the current request. If the queue limit has been reached and session allocation is unacceptably slow, you may decide to purge the whole queue.

For details of the information passed in the XZIQUE parameter list, and advice about designing and coding an XZIQUE exit program, see the programming information in the *CICS Customization Guide*.

Chapter 24. Efficient deletion of shipped terminal definitions

This chapter describes how to delete redundant shipped terminal definitions. It contains the following topics:

Overview

In a transaction routing environment, terminal definitions can be "shipped" from a terminal-owning region (TOR) to an application-owning region (AOR) when they are first needed, rather than being statically defined in the AOR.

Note: The "terminal" could be an APPC device or system. In this case, the shipped definition would be of an APPC connection.

Shipped definitions can become redundant if:

- A terminal user logs off
- A terminal user stops using remote transactions
- The TOR is shut down
- The TOR is restarted, autoinstalled terminal definitions are not recovered, and the autoinstall user program, DFHZATDX, assigns a new set of termids to the same set of terminals

At some stage redundant definitions must be deleted from the AOR (and from any intermediate systems between the TOR and AOR[11]). This is particularly necessary in the last case above, to prevent a possible mismatch between termids in the TOR and the back-end systems.

The CICS Transaction Server for VSE/ESA Release 1 method of deleting redundant shipped definitions consists of two parts:

- Selective deletion
- A timeout delete mechanism

Selective deletion

Each time a terminal definition is installed, CICS Transaction Server for VSE/ESA Release 1 creates a unique "instance token" and stores it within the definition. Thus, if the definition is shipped to another region, the value of the token is shipped too. All transaction routing attach requests pass the token within the function management header (FMH). If, during attach processing, an existing shipped definition is found in the remote region, it is used *only if the token in the shipped definition matches that passed by the TOR*. Otherwise, it is deleted and an up-to-date definition shipped.

[11] For brevity, we shall refer to AORs and intermediate systems collectively as "back-end systems".

The timeout delete mechanism

You can use the timeout delete mechanism in your back-end systems, to delete shipped definitions that have not been used for transaction routing for a defined period[12]. Its purpose is to ensure that shipped definitions remain installed only while they are in use.

Timeout delete gives you flexible control over shipped definitions. CICS allows you to:

- Stipulate the minimum time a shipped definition must remain installed before being eligible for deletion
- Stipulate the time interval between invocations of the mechanism
- Reset these times online
- Cause the timeout delete mechanism to be invoked immediately

The parameters that control the mechanism allow you to arrange for a "tidy-up" operation to take place when the system is least busy. Your operators can use the CEMT transaction to modify the parameters online, or to invoke the mechanism immediately, should fine-tuning become necessary.

Implementing timeout delete

To use timeout delete in a CICS Transaction Server for VSE/ESA Release 1 system to which terminals are shipped, you need only specify two system initialization parameters:

DSHIPIDL={<u>020000</u>|hhmmss}
Specifies the minimum time, in hours, minutes, and seconds, that an *inactive* shipped terminal definition must remain installed in this region. When the CICS timeout delete mechanism is invoked, only those shipped definitions that have been inactive for longer than the specified time are deleted.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to prevent terminal definitions having to be reshipped because they have been deleted prematurely.

hhmmss Specify a 1 to 6 digit number in the range 0–995959. Numbers that have fewer than six digits are padded with leading zeros.

DSHIPINT={<u>120000</u>|0|hhmmss}
Specifies the interval between invocations of the CICS timeout delete mechanism. The timeout delete mechanism removes any shipped terminal definitions that have not been used for longer than the time specified by the DSHIPIDL parameter.

You can use this parameter in a transaction routing environment, on the application-owning and intermediate regions, to control:

- How often the timeout delete mechanism is invoked.
- The approximate time of day at which a mass delete operation is to take place, relative to CICS startup.

[12] Shipped definitions are not deleted if there is an automatic initiate descriptor (AID) associated with the terminal.

0	The timeout delete mechanism is not invoked. You might set this value in a terminal-owning region, or if you are not using shipped definitions.
hhmmss	Specify a 1 to 6 digit number in the range 1–995959. Numbers that have fewer than six digits are padded with leading zeros.

For details of how to specify system initialization parameters, see the *CICS System Definition Guide*.

After CICS startup you can use a CEMT or EXEC CICS INQUIRE DELETSHIPPED command to discover the current settings of DSHIPIDL and DSHIPINT. For flexible control over when mass delete operations take place, you can use a SET DELETSHIPPED command to reset the interval until the next invocation of the timeout delete mechanism. (The revised interval starts *from the time the command is issued*, **not** from the time the remote delete mechanism was last invoked, nor from CICS startup.) Alternatively, you can use a PERFORM DELETSHIPPED command to cause the timeout delete mechanism to be invoked immediately.

For information about the CEMT INQUIRE, PERFORM, and SET DELETSHIPPED commands, see the *CICS-Supplied Transactions* manual. For programming information about their EXEC CICS equivalents, see the *CICS System Programming Reference* manual.

Performance

Compared with pre-CICS Transaction Server for VSE/ESA Release 1 releases, the CICS Transaction Server for VSE/ESA Release 1 selective deletion and timeout delete mechanisms result in:

- A considerable reduction in the pathlength of some transactions.
- A reduction in the number of network flows, leading to better system performance, particularly across a complex of CICS Transaction Server for VSE/ESA Release 1 systems.
- Depending on your choice of DSHIPINT and DSHIPIDL settings, a possible reduction in the number of mass deletions of shipped definitions, and a scheduling of those that do take place for times when your system is lightly loaded.

Note that a poor choice of values for DSHIPINT and DSHIPIDL could result in unnecessary mass delete operations. Here are some suggestions for coding these parameters:

DSHIPIDL: In setting this value, you must consider the length of the work periods during which remote users access resources on this system. Do they access the system intermittently, all day? Or is their work concentrated into intensive, shorter periods?

By setting too low a value, you could cause definitions to be deleted and reshipped unnecessarily. It is also possible that you could cause automatic transaction initiation (ATI) requests to fail with the “terminal not known” condition. This condition occurs when an ATI request names a terminal that is not defined to this system. Usually, the terminal is not defined because it is owned by a remote system, you are using shippable terminals, and no prior transaction routing has taken place from it. By allowing temporarily inactive shipped definitions too short a life, you could increase the number of calls to the XALTENF and XICTENF global user exits that deal with the “terminal not known” condition.

DSHIPINT: You can use this value to control the time of day at which your mass delete operations take place. For example, if you usually warm-start CICS at 7 a.m., you could set DSHIPINT to 150000, so that the timeout delete mechanism is invoked at 10 p.m., when few users are accessing the system.

Warning: If CICS is recycled, perhaps because of a failure, the timeout delete interval is reset. Continuing the previous example, if CICS is recycled at 8:00 p.m., the timeout delete mechanism will be invoked at 11:00 a.m. the following day (15 hours from the time of CICS initialization). In these circumstances, you could use the SET DELETSHIPPED and PERFORM DELETSHIPPED commands to accurately control when a timeout delete takes place.

CICS provides statistics to help you tune the DFHIPIDL and DFHIPINT parameters. The statistics are available online, and are mapped by the DFHA04DS DSECT. For details of the statistics provided, see the *CICS Performance Guide*.

Migration considerations

For compatibility reasons, CICS Transaction Server for VSE/ESA Release 1 continues to support the old remote delete and remote reset mechanisms that were used in pre-CICS Transaction Server for VSE/ESA Release 1 releases. You can always use the new timeout delete mechanism on any CICS Transaction Server for VSE/ESA Release 1 back-end system. Whether the new selective deletion mechanism or the old-style remote delete and reset operates depends on the level of the front-end system. For example, consider the following combinations of front- and back-end systems.

Note: A “front-end” could be a TOR or an intermediate system. Likewise, a “back-end” could be an AOR or an intermediate system.

CICS TS VSE/ESA R1 front-end to CICS TS VSE/ESA R1 back-end

You can use timeout delete on the back-end system. Based on the instance tokens passed by the front-end system, the back-end uses selective deletion to remove redundant definitions singly, as they are referenced by routed transactions.

CICS TS VSE/ESA R1 front-end to pre-CICS TS VSE/ESA R1 back-end

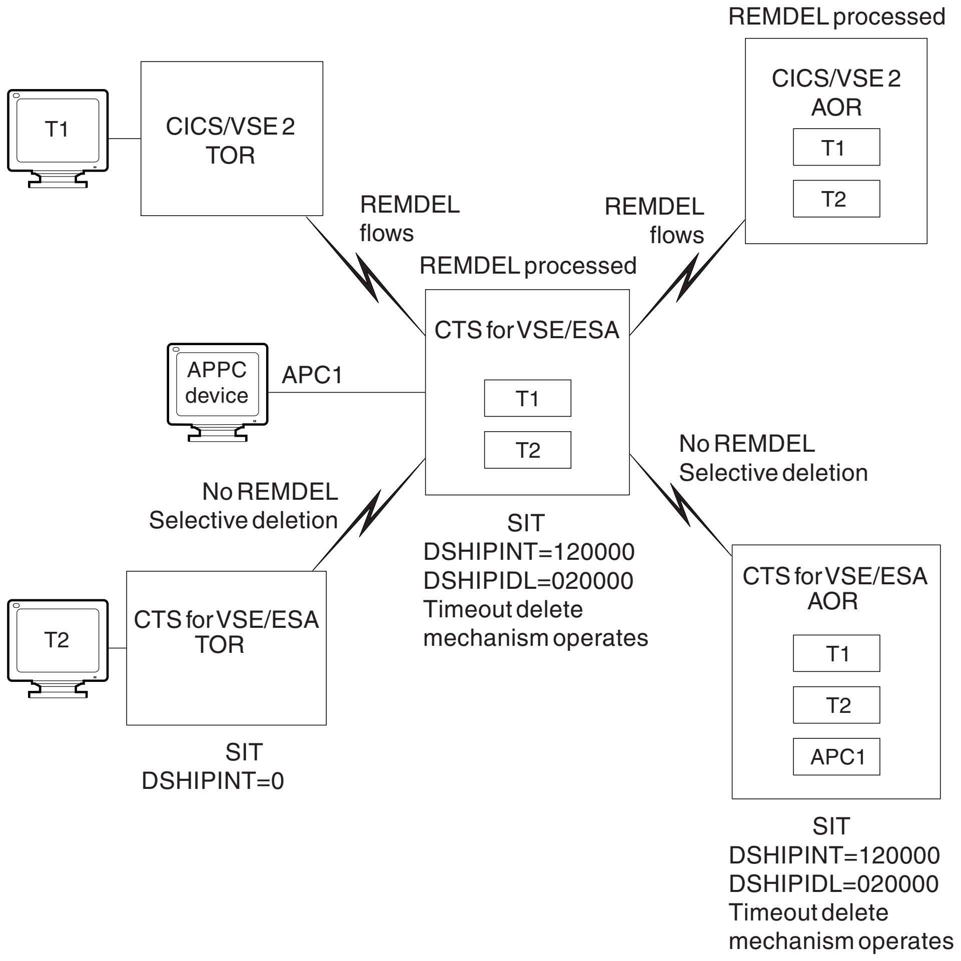
You cannot use timeout delete on the back-end system. The front-end system uses the old-style remote delete and remote reset mechanisms. This means that *all* shipped definitions in the back-end system—whether redundant or not—are deleted after a restart of the TOR or of an intermediate system.

Pre-CICS TS VSE/ESA R1 front-end to CICS TS VSE/ESA R1 back-end

You can use timeout delete on the back-end system. The front-end system uses the old-style remote delete and remote reset mechanisms, which are honored by the back-end system.

Note: If you migrate a pre-CICS Transaction Server for VSE/ESA Release 1 system to CICS Transaction Server for VSE/ESA Release 1, any other CICS Transaction Server for VSE/ESA Release 1 systems to which it is connected will not recognize the upgrade (and therefore continue to issue old-style remote delete and remote reset requests) until their connections to the upgraded system are reinstalled.

Figure 74 shows various combinations of front- and back-end systems. In the figure, old-style remote delete and remote reset requests are shown collectively as "REMDEL"s.



KEY:

REMDEL	Pre-CTS for VSE/ESA remote reset and remote delete requests
T1 T2	Remote terminal definitions
APC1	Remote APPC connection definition

Figure 74. Deletion of shipped terminal definitions in a mixed-release network

Part 6. Recovery and restart

Table 18. Recovery road map

If you want to...	**Refer to...**
Learn how to recover from a session or system failure	Chapter 25, “Recovery and restart in interconnected systems” on page 233
Learn about those aspects of the CICS extended recovery facility that affect intercommunication	Chapter 26, “Intercommunication and XRF” on page 251
Learn how the use of VTAM persistent sessions affects intercommunication	Chapter 27, “Intercommunication and VTAM persistent sessions” on page 253

Chapter 25. Recovery and restart in interconnected systems

This chapter describes those aspects of CICS recovery and restart that apply particularly in the intercommunication environment. It contains the following topics:

Introduction to intersystem recovery and restart

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 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 single major 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 causes 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, for example, 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 **in-doubt** period, as illustrated in Figure 75 on page 237.

If the intersystem session fails before the in-doubt period is reached, both sides back out in the normal way. After this period, both sides have committed their changes. If, however, the intersystem session fails during the in-doubt 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 option of the local transaction (see page 235); this action may be inconsistent with the action taken by the partner system.

For APPC sessions, CICS reduces the risk explained in the example by attempting resynchronization on a separate session (see “Action following failure during the in-doubt period” on page 239). For LUTYPE6.1 sessions, and also for APPC sessions if the resynchronization attempt fails, there are three possible courses of action that an application can take. (For MRO sessions, only the first two courses are available.)

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 option of the transaction definition

You can control, in some part, the action that CICS takes after failure during the in-doubt period by specifying transaction backout attributes when you define the transaction. This is done by means of the INDOUBT option of the CEDA DEFINE and ALTER TRANSACTION commands. The INDOUBT option of a transaction is honored when communication is lost with a partner and the task is in the in-doubt 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 option to the initiator, because they rely on all agents except the last committing if they are in doubt. APPC supports the INDOUBT option for the initiator and not-last agents. (For an explanation of the terms "last" and "not-last" as applied to agents, see page 237.) 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 INDOUBT option has the following format:

```
INDOUBT({BACKOUT|COMMIT|WAIT})
```

and the choices have the following meanings:

BACKOUT
Specifies that the transaction is to be backed out if a failure occurs during the in-doubt period. This is the default value.

Transaction backout is always performed for failures that occur outside the in-doubt period, irrespective of what is specified in the INDOUBT option. 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 in-doubt 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 APPC only) specifies that, if the session fails during the in-doubt period:

- Changes to recoverable temporary storage are to be neither committed nor backed out. 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.

  For this to happen, every change to recoverable temporary storage made by the transaction since the last syncpoint must be one of the following:

  – A new addition to a queue
  – A modification to a new addition to a queue
  – The creation of a new queue

WRITEQ TS (without REWRITE) and START PROTECT requests always qualify, as do WRITE TS REWRITE requests to new items on a queue. This is because CICS can back out these commands by erasing the added items, or commit them by unlocking the queue. WRITE TS REWRITE requests to items that existed before the start of the transaction (or before the last syncpoint) do not qualify.

If *some* changes to recoverable temporary storage do not qualify, *all* changes are committed at session failure.

- Interval-control START PROTECT requests are neither committed nor backed out.
- Changes to other recoverable resources are backed out.

This option is honored only if the local transaction has only one connection with a partner at the time of the syncpoint. In other circumstances, the BACKOUT option applies.

If session recovery is unsuccessful, the START commands are canceled but temporary storage queue changes are committed.

Syncpoint flows

The ways in which syncpoint requests and responses are exchanged on intersystem conversations are defined in the LUTYPE6.1 and APPC architectures. CICS multiregion operation uses the LUTYPE6.1 protocols. Although the formats of syncpoint flows for LUTYPE6.1 and APPC differ, the concepts of syncpoint exchanges are similar.

The flows involved in syncpoint exchanges are illustrated in Figure 75 on page 237. 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 the *CICS Distributed Transaction Programming Guide*.

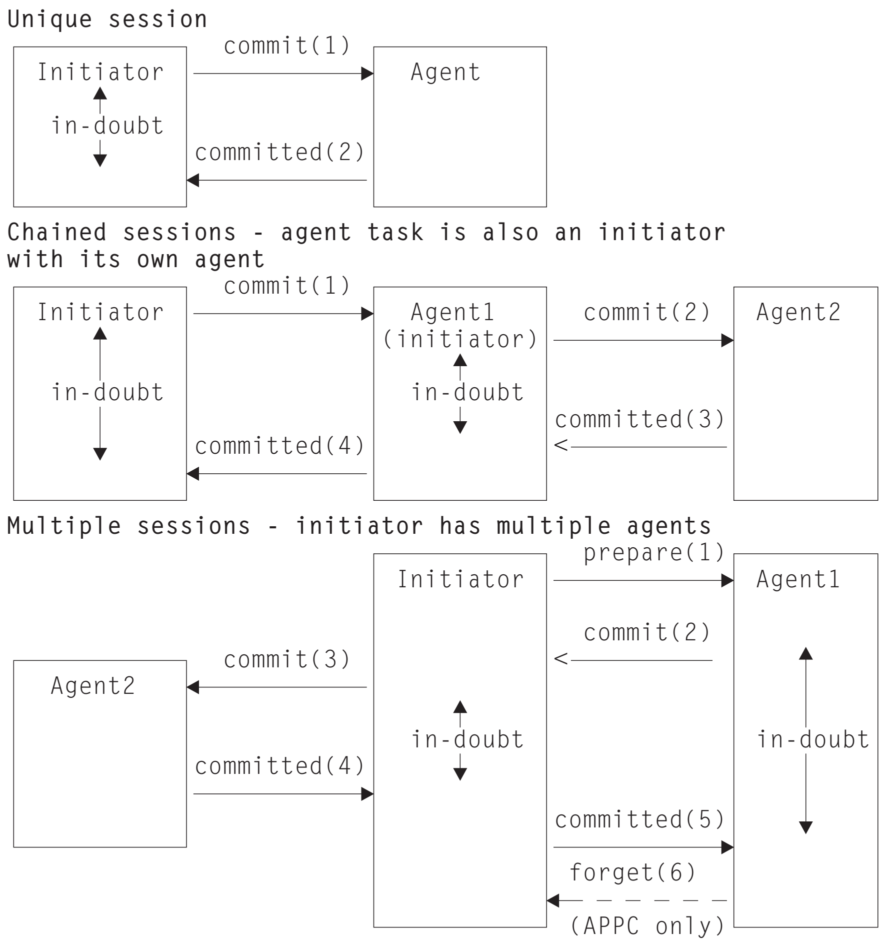


Figure 75. Syncpointing flows

In Figure 75, the numbers in brackets, for example, (1), show the sequence of the actions in each flow.

1. The initiator must be in send state on all its sessions.
2. When they issue their syncpoint, agents must be in send state on all sessions except the session on which they receive, prepare, or commit.

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 the last. (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 APPC conversations only, these agents respond “forget” to show that they do not require resynchronization.

Relationship between initiator and agent

Figure 76 shows the relationship between 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 syncpoint initiator.

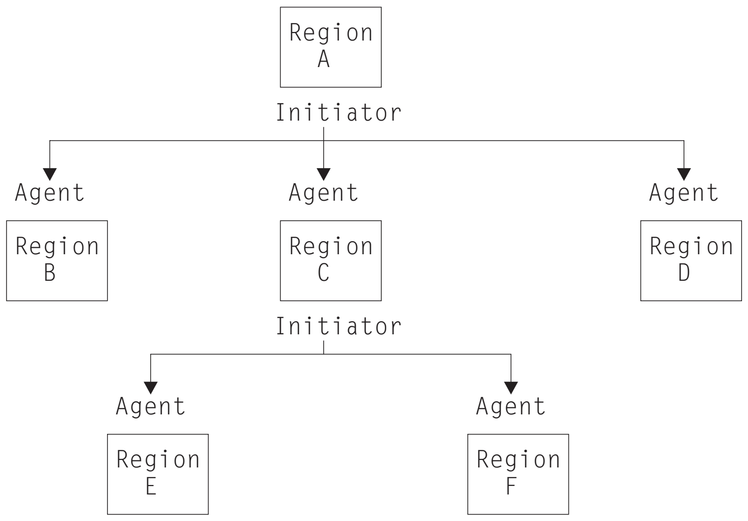


Figure 76. Relationship between initiator and agent regions

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 in-doubt 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:

Session failures
: These occur either between the CICS systems or between CICS and the terminal associated with a transaction. Unless they are capable of handling session failures, the transactions connected to the sessions are abended when they next try to use them. Resource recovery is performed as for unconnected transaction abends.

Total CICS system failures
: The failing system is recovered using emergency restart as for an unconnected system though there are extra features, described below, 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, unless the transaction can handle session failures, a remote system failure causes a local transaction abend.

Transaction abends
: These are recovered using dynamic transaction backout as for unconnected systems. The mirror and relay transactions are not special in this respect, they are recoverable like all other transactions.

  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 in-doubt period

This section discusses CICS actions following failure during the in-doubt period under the following headings:

- "APPC connections"
- "LUTYPE6.1 connections" on page 240
- "MRO connections" on page 240
- "Messages that can help recovery" on page 240
- "Restoring data integrity" on page 242

APPC connections

When an APPC session fails during the in-doubt 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-work states 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 handled, leads to an abend and (if necessary) backout, 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 of the connecting hardware. What happens after a total system failure depends on whether the failed CICS is using VTAM persistent session support:

- If the failed CICS uses persistent session support, its sessions are kept in "recovery pending" state until the system is restarted. Provided that restart takes place before the expiry of the persistent session delay interval, APPC open sessions are rebuilt. APPC busy sessions, however, are unbound. Thus, a partner CICS that is connected by an APPC link sees the failure simply as a session failure, and processes an in-doubt transaction accordingly. That is, it tries to contact the previously-failed system using a different session, as described above.

- If the failed CICS is not using persistent session support, or if it is not restarted within the delay interval, *all* its sessions are unbound. The partner CICS, having no alternative sessions to use, must abend the transaction and obey the INDOUBT option. It must then wait until the connection has been reestablished, after emergency restart, before it can determine and report the state of the distributed logical unit of work.

  When the intersystem session is recovered, unit-of-work states 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 back out any temporary storage changes locked due to the INDOUBT(WAIT) option.

For further information about persistent session support, see Chapter 27, "Intercommunication and VTAM persistent sessions" on page 253.

LUTYPE6.1 connections

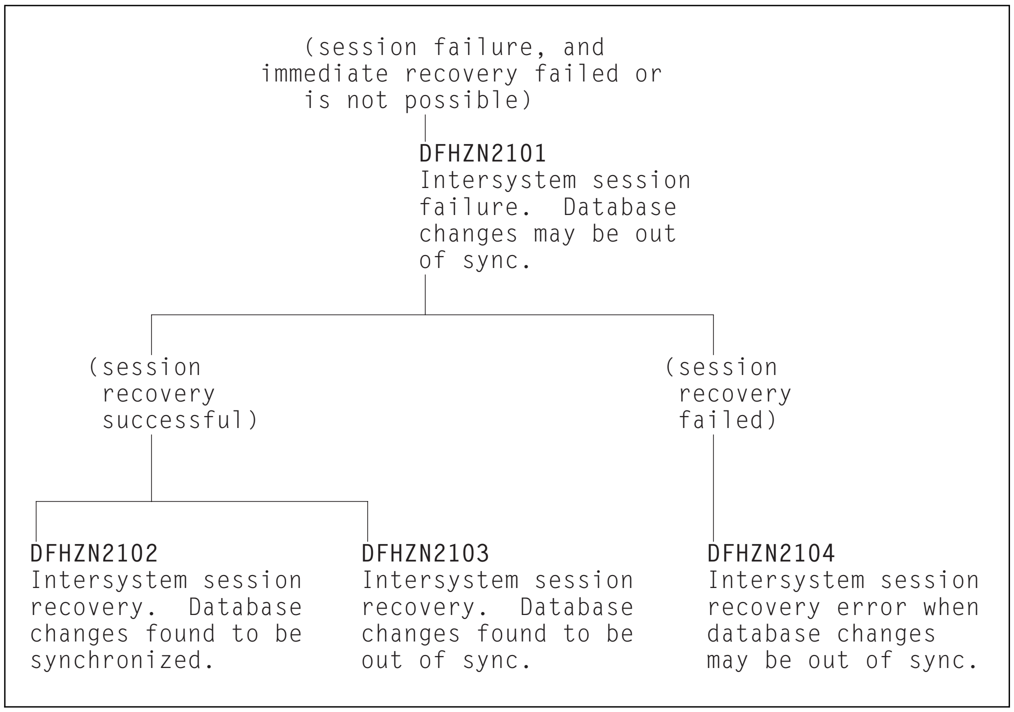
The absence of an LU services manager for LUTYPE6.1 connections makes immediate recovery impossible. Recovery is possible only after sessions have been reestablished. When the sessions are reestablished, the two sides exchange message sequence numbers to determine (and report to the CSMT log) whether 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 transaction definition.

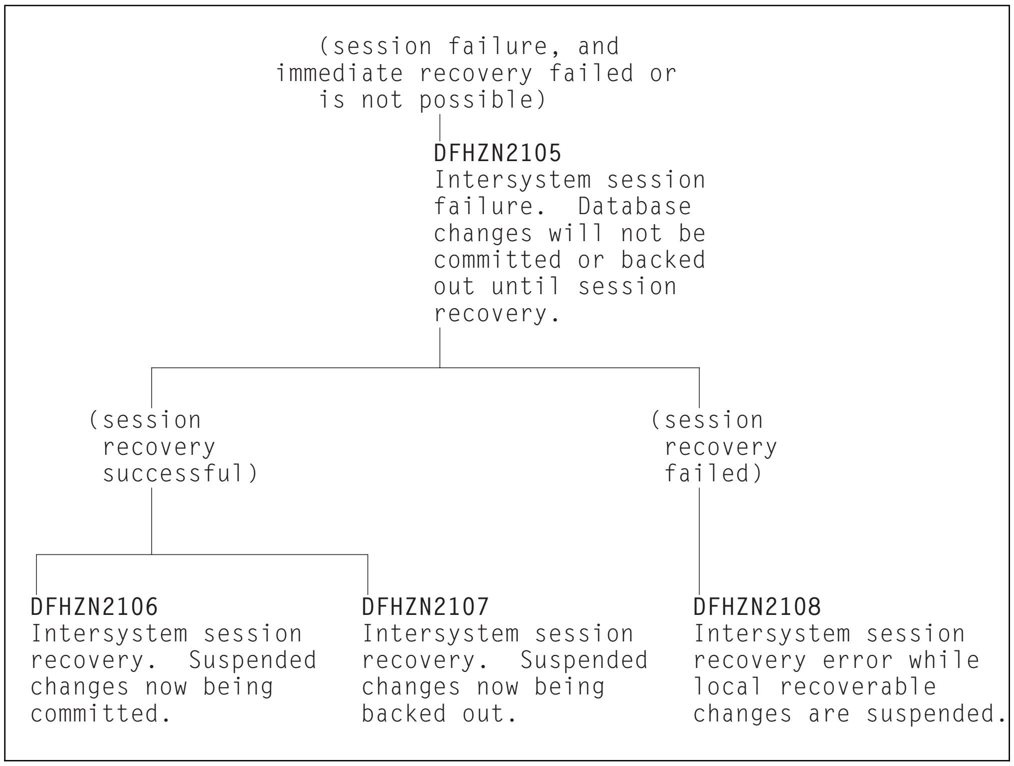
Messages that can help recovery

The messages associated with intersystem session failure and recovery are shown in two figures. Figure 77 on page 241 shows the messages associated with the INDOUBT(BACKOUT or COMMIT) attributes. Figure 78 on page 241 shows the messages associated with the INDOUBT(WAIT) attribute. Full details are in the *VSE/ESA Messages and Codes Volume 3* manual.



(session failure, and immediate recovery failed or is not possible)
DFHZN2101
Intersystem session failure. Database changes may be out of sync.
(session recovery successful)
(session recovery failed)
DFHZN2102
Intersystem session recovery. Database changes found to be synchronized.
DFHZN2103
Intersystem session recovery. Database changes found to be out of sync.
DFHZN2104
Intersystem session recovery error when database changes may be out of sync.

Figure 77. Session failure messages for INDOUBT(BACKOUT or COMMIT)



(session failure, and immediate recovery failed or is not possible)
DFHZN2105
Intersystem session failure. Database changes will not be committed or backed out until session recovery.
(session recovery successful)
(session recovery failed)
DFHZN2106
Intersystem session recovery. Suspended changes now being committed.
DFHZN2107
Intersystem session recovery. Suspended changes now being backed out.
DFHZN2108
Intersystem session recovery error while local recoverable changes are suspended.

Figure 78. 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 DFHZN2101 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 DFHZN2102, DFHZN2103, or DFHZN2104 message is issued for each session. For MRO and LUTYPE6.1 conversations, 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 DFHZN2101 message. After session recovery, a DFHZN2102, DFHZN2103, or DFHZN2104 message may be issued by the agent region.

As the following example shows, the system or application design can mean that there is no threat to data integrity even though DFHZN2103 or DFHZN2104 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 intersystem session fails during the in-doubt period, the local CICS region reports the possible threat to integrity with DFHZN2101 and takes unilateral action to commit or back out 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 DFHZN2102 or DFHZN2103. 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.

Restoring data integrity

If CICS messages indicate that database changes are or may be out of synchronization, 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 unit of work in the affected CICS regions, and determine what action is needed to synchronize the databases. For programming information about how to do this, see the *CICS Customization Guide*.

Recovery for APPC connections

This section describes recovery for APPC connections under the following headings:

- “Exchange-lognames process”
- “Pending units of work” on page 244
- “Connected system recovery – an example” on page 245

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 APPC 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 in-doubt period of a transaction, causing a failure of the connection between two CICS systems. INDOUBT(WAIT) is coded on both transaction definitions. 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 does 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 unit-of-work status information) are not the same ones that were in use at the preceding failure.

The exchange-lognames process affects only level-2 synchronization conversations. If it fails, level-2 synchronization conversations are not allowed on the link until the failure is resolved. This resolution can be achieved only by operator action. However, level-0 and level-1 synchronization traffic on the link is unaffected by the failure, and continues as normal. For information about synchronization levels, see “Synchronization levels” on page 18.

The CEMT INQ CONNECTION command can be used to determine whether the exchange-lognames process has completed successfully. The exchange-lognames status is shown only for APPC links that, before the failure, were carrying sync level 2 conversations (otherwise it is blank). It is ‘XOK’ if the process was successful. If it is shown as ‘XNOtdone’ (exchange lognames not done) and ‘ACQuired’, CICS does not allow any level-2 synchronization conversations. This may (depending on the design of your system) mean that the connection is not available to applications.

One or more of the following messages appear on the CSMT log:

DFHZN2110 ABNORMAL REPLY TO EXCHANGE LOG NAME COMMAND SENT TO SYSTEM: *sysid*
DFHZN2111 COLD/WARM RESTART MISMATCH WITH SYSTEM *sysid*
DFHZN2112 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

If 'XOK' and resynchronization is not complete, you cannot set the connection to NOTPENDING.

Warning: 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 *VSE/ESA Messages and Codes Volume 3* 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 units-of-work. 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.

Pending units of work

When an APPC session failure leaves a unit-of-work 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 units-of-work for the named system.

You should determine whether data integrity in your system is dependent on successful session recovery. You may find that intersystem 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 your conversation is dependent on session recovery and resynchronization and this has failed, you should investigate thoroughly to find out 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 CEMT SET CONNECTION NOTPENDING command 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.

Warning: This action prevents CICS from detecting 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 level-2 synchronization conversations are permitted.
- The connection is set to 'notpending' status.
- The normal resynchronization process is overridden by the deletion of all information describing the unresolved units of work.
- The unit-of-work status information containing the log name of the partner system is erased; the connection is as if the CICS system had been cold-started.
- If INDOUBT(WAIT) has been specified and temporary storage changes are suspended (DFHZN2105 has been issued), the changes are unilaterally committed. You can determine the status of data integrity and restore it as described in "Restoring data integrity" on page 242.

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 restoring data integrity.

Connected system recovery – an example

As an illustration of connected system recovery design, consider the following simple example:

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 a 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 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 a session or system failure during the in-doubt period of syncpoint processing.

For failure during the in-doubt period (and, for APPC sessions, following a failure of the resynchronization attempt), the change made to the stock file is backed out, and message DFHZN2101, 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 been 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 DFHZN2102 is issued. Otherwise, DFHZN2103 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, it appears to the conversation as if only the connection has failed. The failed system is usually emergency-restarted, and so its local resources are 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.

APPC unit-of-work states and LUTYPE6.1 message sequence numbers 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.

Note: If the failed system uses VTAM persistent session support, and is restarted within the persistent session delay interval, its APPC sessions are held in “recovery pending” state, and subsequently rebound without the need for network flows. The effect of this is described in “APPC connections” on page 239 and in Chapter 27, “Intercommunication and VTAM persistent sessions” on page 253.

In a busy system with a large number of parallel APPC sessions, it is possible that some sessions may fail to rebind after emergency restart. This reduces the number of parallel sessions available to communicating transactions and may therefore affect system performance. The situation can be detected by comparing the active and available counts returned on a CEMT INQUIRE MODENAME transaction. All the unbound sessions for that modename can be rebound by using a CEMT SET MODENAME ACQUIRED transaction from either end of the connection. Operation of the bound sessions is not interrupted.

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 is 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 leads 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 APPC conversations, 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 succeeding 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 can be retried, CICS does so transparently to the application program attempting to send a message.

For programming information about user-supplied DFHZNEP programs, see the *CICS 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 is detected by means of a time-out value specified on the transaction definition, which expires when a program has waited the specified period without a reply from the deadlocked transaction. CICS abends 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 is routed to all three systems and then deadlocks 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 it is not available).

In addition, some types of request may cause a transaction abend because incorrect 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 through VTAM) or AZI6 (for communication through 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 is unable to do so in the same way when those resources are remote.

Trace and dump facilities are defined 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 through MRO or through the VTAM application-to-application facility.

Detailed sequences and request formats for diagnosis of problems with CICS intercommunication can be found in the *CICS 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 26. Intercommunication and XRF

For further information about the extended recovery facility (XRF) of CICS Transaction Server for VSE/ESA, see the *CICS 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 12, "Defining links to remote systems" on page 93.

You can use the AUTOCONNECT option in your RDO CONNECTION resource definitions to cause CICS to try to reestablish the sessions following a takeover by the alternate CICS system.

Also, the bound or unbound status of some ISC session types can be tracked. In these cases, CICS can try 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 system initialization parameter. For information about system initialization parameters, see the *CICS System Definition Guide*.

MRO sessions: The status of MRO sessions cannot be tracked. Following a takeover by the alternate CICS system, CICS tries to reestablish MRO sessions according to the value specified for the INSERVICE option of the RDO CONNECTION resource definition.

LUTYPE6.1 sessions: Following a takeover, CICS tries to reestablish LUTYPE6.1 sessions in either of the following cases:

1. The AUTOCONNECT option of the RDO SESSIONS resource 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 RDO SESSIONS resource definition.

Single-session APPC devices: Following a takeover, CICS tries to reestablish single APPC sessions in either of the following cases:

1. The AUTOCONNECT option of the SESSIONS or TYPETERM RDO resource definition specifies YES.
2. The session is being tracked, and is bound when the active CICS fails. Single APPC sessions are tracked unless RECOVOPTION(NONE) is specified in the SESSIONS or the TYPETERM RDO resource definition (depending upon which form of definition is being used). Although RECOVOPTION has five possible values, for ISC there is a choice between NONE (no tracking) and *any one* of the other options (tracking).

Parallel APPC sessions: Following a takeover, CICS tries to reestablish the LU services manager sessions in either of the following cases:

- The AUTOCONNECT option of the CONNECTION definition specifies YES or ALL.
- 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.

As soon as the LU services manager sessions are reestablished, CICS tries to establish the sessions for any mode group that specifies autoconnection.

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.

Chapter 27. Intercommunication and VTAM persistent sessions

For definitive information about CICS support for VTAM persistent sessions, see the *CICS Recovery and Restart Guide*. This chapter looks at those aspects of persistent sessions that apply particularly to intersystem communication. For details of the link definitions required for persistent session support, refer to Chapter 12, "Defining links to remote systems" on page 93 and the *CICS Resource Definition Guide*. For details of the PSDINT system initialization parameter used to specify persistent session support, see the *CICS System Definition Guide*.

Comparison of persistent session support and XRF

XRF was introduced in CICS/VSE Version 2 to allow an alternate, partially initialized CICS system to take over control from an active CICS system which had failed. The use of VTAM persistent sessions provides an alternative to XRF. Persistent sessions allow you to restart a failed CICS in place, without the need for network flows to rebind CICS sessions. (Note that you cannot specify both XRF and CICS persistent session support for the same system.)

XRF provides availability of the system (through active and alternate systems) and availability for the user (through availability of the system and exploitation of backup sessions). Active and alternate pairs of systems require their own versions of some data sets (for example, auxiliary trace and dump data sets).

Persistent session support provides availability of the system (through restart in place of one system) and availability for the end user (through availability of the system and persistent sessions). Only one set of data sets is required. Only one system is required. Persistent session support has the following advantages over XRF:

- It supports all session types except MRO, LU6.1, and LU0 pipeline sessions. XRF does not support local terminals, MRO, or ISC (LU6.1 or LU6.2) sessions.
- It is easier to install and manage than XRF. It requires only a single system.

However, persistent session support does not retain sessions after a VTAM, VSE, or CPC failure. If you need to ensure rapid restarts after such a failure, you could use XRF rather than persistent sessions.

Interconnected CICS environment, recovery and restart

CICS systems can be interconnected via MRO, LU6.1, or LU6.2 connections and sessions.

MRO sessions

MRO connections do not have the ability to persist across CICS failures and subsequent emergency restarts.

LU6.1 sessions

If a CICS fails in a multisystem environment, all the LU6.1 sessions that are connected to it are held in recovery pending state until it is restarted with an emergency restart or until the expiry of the persistent session delay interval. In either case, the LU6.1 sessions are then unbound. They need to be reacquired before they can be used again.

Slightly different symptoms of the CICS failure may be presented to the systems programmer, or operator, depending on whether persistent session support is used. In systems without persistent session support, all the LU6.1 sessions unbind immediately after the failure.

In a system with persistent session support, the LU6.1 sessions are not unbound until the emergency restart (if this occurs within the persistent session delay interval) or the expiry of the persistent session delay interval. Consequently, these sessions may take a longer time to be unbound.

LU6.2 sessions

LU6.2 sessions that connect different CICS systems are capable of persistence across the failure of one or more of the systems and a subsequent emergency restart within the persistent session delay interval.

However, these sessions are unbound in certain circumstances, even if persistent sessions are supported in your system. The following sessions are unbound after a CICS failure and emergency restart, even if you have defined them to be persistent:

- Sessions for which no catalog entry is found. This applies to:
  - Autoinstalled LU6.2 parallel sessions.
  - Autoinstalled LU6.2 single sessions initiated by BIND requests.
  - Autoinstalled LU6.2 single sessions initiated by VTAM CINIT requests, if the AIRDELAY system initialization parameter is set to zero. (AIRDELAY specifies the interval that elapses after an emergency restart before autoinstalled terminal entries that are not in session are deleted.)

    In other words, the only autoinstalled LU6.2 sessions that are not unbound are single sessions initiated by CINIT requests, and then only if AIRDELAY is greater than zero.
- *All* sessions on an LU6.2 connection to a failing TOR, where, on one or more of the sessions, an AOR has function-shipped an ATI request to the TOR, because the request is associated with a terminal owned by the TOR. (ATI-initiated transaction routing is described on page 57.)
- *All* sessions on an LU6.2 connection, where, on one or more of the sessions, transaction routing via CRTE is taking place but there is no conversation in progress at the point of the failure. (Where a conversation is in progress, a DEALLOCATE(ABEND) is sent to the partner of the failing CICS.)

Effects on LU6.2 session control: After the failure of CICS in an LU6.2 interconnected environment, and a subsequent emergency restart within the persistent session delay interval, transaction CLS1 (CNOS) is not run **unless** one side of the connection had issued a CNOS request to zero or the connection was in the process of CNOS negotiation at the time of the failure.

The failing system runs transaction CLS2 (XLN, exchange log names) as soon as it can after emergency restart within the persistent session delay interval. CLS2 has to run before any further synclevel 2 conversations can be processed by either of the connected systems.

Effect on application programs

The use of VTAM persistent sessions has implications for DTP applications that use the APPC protocol. This is described in the *CICS Distributed Transaction Programming Guide*.

Part 7. Appendixes

Table 19. Appendixes road map	
If you want to...	**Refer to...**
Check for restrictions on the CICS intercommunication facilities	Appendix A, "Rules and restrictions checklist" on page 259
Understand how CICS implements the APPC architecture	Appendix B, "CICS mapping to the APPC architecture" on page 263
Check technical terms used in this manual	"Glossary" on page 275

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.

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 is 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.
  - MRO sessions.
  - IBM 2260 terminals.
  - Pipeline logical units with pooling.
  - VSE system consoles. (Messages entered through a console can be directed to any CICS system via the VSE/ESA MSG command.)
- The transaction CEOT is not supported by the transaction routing facility.
- The execution diagnostic facility (EDF) can be used in single-terminal mode to test a remote transaction.

  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 does 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.
- 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.)
- When transaction routing to or from APPC devices, CICS does not support CPI Communications conversations with sync level characteristics of CM_SYNC_POINT.
- TCTUAs are not shipped when the principal facility is an APPC parallel session.

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 that issued the ATI request. (See "Shipping terminals for automatic transaction initiation" on page 58.)

Acquiring LUTYPE6.1 sessions

- If an application tries to acquire 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(YES|ALL), 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 commands are supported only by APPC 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 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 transaction CEMT. The only terminals visible to this transaction are those owned by the system on which the master terminal transaction is actually running.

Installation and operations

- Module DFHIRP must be made SVA-resident; otherwise jobs and console commands may abend on completion.
- Do not install more than one APPC connection between an LU-LU pair.
- Do not install an APPC and an LUTYPE6.1 connection at the same time between an LU-LU pair.
- Do not install more than one MRO connection between the same two CICS regions.
- Do not install more than one generic EXCI connection on a CICS region.

Resource definition

- The PRINTER and ALTPRINTER options for a VTAM terminal must (if specified) name a printer owned by the same system as the one that owns 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.
- 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 IRC 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. CICS mapping to the APPC architecture

This appendix shows how the APPC programming language (described in the SNA publication, *Transaction Programmer's Reference Manual for LU Type 6.2*) is implemented by CICS. It contains the following topics:

- "Supported option sets."

  This is a table showing which APPC option sets are supported by CICS and which are not.

- "CICS implementation of control operator verbs" on page 265.

  This section describes how CICS implements the APPC control operator verbs. It includes tables showing how these verbs map to CICS commands.

- "CICS deviations from the APPC architecture" on page 274.

  This section describes the way in which the CICS implementation of APPC differs from the architecture described in the *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2* manual.

For information on how the CICS application programming interface for basic and unmapped conversations maps to the APPC verbs, see the *CICS Distributed Transaction Programming Guide*.

Supported option sets

Table 20 (Page 1 of 3). CICS support of APPC options sets

Set No.	Set name	Supported
101	Clear 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[13]	Receive immediate	Yes
108	Syncpoint services	Yes
109	Get TP name and instance identifier	No
110	Get conversation type	Yes
111	Recovery from program errors detected during syncpoint	Yes
201	Queued allocation of a contention-winner session	No
203	Immediate allocation of a session	Yes
204	Conversations between programs located at the same LU	No

[13] CICS programs support receive_immediate requests provided these requests are coded using the common programming Interface for communications.

Table 20 (Page 2 of 3). CICS support of APPC options sets

Set No.	Set name	Supported
211	Session-level LU-LU verification	Yes
212	User ID verification	Yes
213	Program-supplied user ID and password	No
214	User ID authorization	Yes
215	Profile verification and authorization	Yes
217	Profile pass-through	No
218	Program-supplied profile	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	No
249	Vote read-only response to a syncpoint operation	No
251	Extract transaction and conversation identity information	No
290	Logging of data in a system log	No
291	Mapped conversation LU services component	Yes
401	Reliable one-way brackets	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

Table 20 (Page 3 of 3). CICS support of APPC options sets

Set No.	Set name	Supported
612	Contention-winner automatic activation limit	No
613	Local maximum (LU, mode) session limit	Yes
616	CPSVCMG modename support	No
617	Session-level selective cryptography	No

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 display device. It is described in the *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
```

Programming information about these commands is given in the *CICS System Programming Reference* manual.

Some control operator verbs are supported by CICS resource definition. The definition of APPC links is described in “Defining APPC links” on page 101. Details of the resource-definition syntax are given in the *CICS Resource Definition Guide*.

With resource definition online, the CEDA transaction can be used to change some CONNECTION and SESSION options while CICS is running. With macro-level definition, the corresponding options are fixed for the duration of the CICS run.

Control operator verbs

The following tables show how APPC control operator verbs are implemented by CICS. See “Return codes for control operator verbs” on page 272 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(vble)	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(value1,)
MIN_CONWINNERS_SOURCE(vble)	MAXIMUM(,value2)
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.
RESOURCE(vble)	Connection RDO
LU_NAME(vble)	Passed internally
MODE_NAME(vble1,vble2)	Passed internally
RETURN_CODE	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(NO|YES)	CICS supports YES
DRAIN_TARGET(NO|YES)	CICS supports YES
FORCE(NO|YES)	Not supported
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_CONVERSATION_GROUP	**Not supported**

DEACTIVATE_SESSION	**Not supported**

DEFINE_LOCAL_LU	**DEFINE SESSIONS** + **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))	Total of MAX(nn) on all sessions
SECURITY(ADD USER_ID(vble))	In an external security manager
SECURITY(ADD PASSWORD(vble))	Not supported; defined in an ESM
SECURITY(ADD PROFILE(vble))	Not supported; defined in an ESM
SECURITY(DELETE USER_ID(vble))	Supported by redefining DFHSNT or in an ESM
SECURITY(DELETE PASSWORD(vble))	Not supported; defined in an ESM
MAP_NAME(ADD(vble))	Not supported
MAP_NAME(DELETE(vble))	Not supported
BIND_RSP_QUEUE_CAPACITY(YES|NO)	Not supported

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_MAX_RU_SIZE _LOWER_BOUND (vble)	Fixed at 8
SEND_MAX_RU_SIZE _UPPER_BOUND (vble)	SENDSIZE on SESSIONS
PREFERRED_RECEIVE_RU_SIZE (vble)	Not supported
PREFERRED_SEND_RU_SIZE (vble)	Not supported
RECEIVE_MAX_RU _SIZE_LOWER_BOUND (vble)	Fixed at 256
RECEIVE_MAX_RU _SIZE_UPPER_BOUND (vble)	RECEIVESIZE on SESSIONS
SINGLE_SESSION_REINITIATION OPERATOR	Not supported
SINGLE_SESSION_REINITIATION PLU	Not supported
SINGLE_SESSION_REINITIATION SLU	Not supported
SINGLE_SESSION_REINITIATION PLU_OR_SLU	Not supported
SESSION_LEVEL_CRYPTOGRAPHY (NOT_SUPPORTED)	Default
SESSION_LEVEL_CRYPTOGRAPHY (MANDATORY)	Not supported
SESSION_LEVEL_CRYPTOGRAPHY (SELECTIVE)	Not supported
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	MAXIMUM(,value2) on SESSIONS
SESSION_DEACTIVATED_TP_NAME(vble)	Not supported
LOCAL_MAX_SESSION_LIMIT(vble)	MAXIMUM(nn,) on SESSIONS

DEFINE_REMOTE_LU	DEFINE CONNECTION (CICS resource definition)
FULLY_QUALIFIED_LU_NAME(vble)	Cannot be specified
LOCALLY_KNOWN_LU_NAME(NONE)	Not supported
LOCALLY_KNOWN_LU_NAME(NAME(vble))	CONNECTION(name)
UNINTERPRETED_LU_NAME(NONE)	Defaults to CONNECTION(name)
UNINTERPRETED_LU_NAME(NAME(vble))	NETNAME on CONNECTION
INITIATE_TYPE(INITIATE_ONLY)	Not supported
INITIATE_TYPE(INITIATE_OR_QUEUE)	Not supported
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 or SESSKEY in ESM APPCLU profile
SECURITY_ACCEPTANCE(NONE)	ATTACHSEC(LOCAL)
SECURITY_ACCEPTANCE(CONVERSATION)	ATTACHSEC(VERIFY)
SECURITY_ACCEPTANCE (ALREADY_VERIFIED)	ATTACHSEC(IDENTIFY) or ATTACHSEC(PERSISTENT)

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)	Not supported; defined in an ESM
SECURITY_REQUIRED(CONVERSATION)	Not supported; defined in an ESM
SECURITY_REQUIRED(ACCESS(PROFILE))	Not supported
SECURITY_REQUIRED(ACCESS(USER_ID))	Not supported; defined in an ESM
SECURITY_REQUIRED (ACCESS(USER_ID_PROFILE))	Not supported
SECURITY_ACCESS (ADD(USER_ID(vble)))	Transaction can be redefined
SECURITY_ACCESS (ADD(PROFILE(vble)))	Transaction can be redefined
SECURITY_ACCESS (DELETE(USER_ID(vble)))	Transaction can be redefined
SECURITY_ACCESS (DELETE(PROFILE(vble)))	Transaction can be redefined
PIP(NO)	Specified for all TPs
PIP(YES(vble))	Specified on CONNECT PROCESS
PIP(NO_LU_VERIFICATION)	Default for all PIP data
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)	Not supported
PRIVILEGE(SESSION_CONTROL)	Not supported
PRIVILEGE(DEFINE)	Not supported
PRIVILEGE(DISPLAY)	Not supported
PRIVILEGE(ALLOCATE_SERVICE_TP)	Not supported
INSTANCE_LIMIT(vble)	Not supported
RETURN_CODE	Supported

DELETE	EXEC CICS DISCARD
LOCAL_LU_NAME	Not supported
REMOTE_LU_NAME	Not supported
MODE_NAME	Not supported
TP_NAME	DISCARD TRANSACTION( )
RETURN_CODE	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 * can be used.
LU_SESSION_LIMIT (vble) LU_SESSION_COUNT (vble) SECURITY (vble) MAP_NAMES (vble) REMOTE_LU_NAMES (vble) TP_NAMES (vble) BIND_RSP_QUEUE_CAPABILITY (vble) RETURN_CODE	MAXIMUM on INQ MODENAME ACTIVE on INQ MODENAME Not available Not supported INQ CONNECTION(*) INQ TRANSACTION(*) Not supported Supported

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) RETURN_CODE	This is CONNECTION name NETNAME on INQ CONNECTION Not supported SINGLESESS(Y|N) on CEDA VIEW Always YES Not available Not available CEDA VIEW SESSIONS with locally known LU name Supported

DISPLAY_MODE	CEMT INQUIRE MODENAME + CEMT INQUIRE TERMINAL
FULLY_QUALIFIED_LU_NAME (vble)	Cannot be specified.
MODE_NAME (vble)	MODENAME
LOCAL_MAX_SESSION_LIMIT (vble)	AVA on CEMT INQ MODENAME
CONVERSATION_GROUP_IDS (vble)	Not supported
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
PREFERRED_SEND_RU_SIZE (vble)	Not supported
PREFERRED_RECEIVE_RU_SIZE (vble)	Not supported
SINGLE_SESSION_REINITIATION (vble)	Not supported
SESSION_LEVEL_CRYPTOGRAPHY (vble)	Not available
SESSION_DEACTIVATED_TP_NAME	Not supported
CONWINNER_AUTO_ACTIVATE_LIMIT (vble)	Not available
LU_MODE_SESSION_LIMIT (vble)	MAXIMUM on INQ MODENAME
MIN_CONWINNERS (vble)	Not supported
MIN_CONLOSERS (vble)	Not supported
TERMINATION_COUNT (vble)	Not supported
DRAIN_LOCAL_LU (vble)	Not supported
DRAIN_REMOTE_LU (vble)	Not supported
LU_MODE_SESSION_COUNT (vble)	ACTIVE on INQ MODENAME
CONWINNERS_SESSION_COUNT (vble)	Not available
CONLOSERS_SESSION_COUNT (vble)	Not available
SESSION_IDS (vble)	INQ TERMINAL(*)
RETURN_CODE	Supported

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)	Not available
PIP (vble)	CICS TPs allow PIP YES and NO
DATA_MAPPING (vble)	Always NO
FMH_DATA (vble)	Always YES
PRIVILEGE (vble)	Not supported
INSTANCE_LIMIT (vble)	Not supported
INSTANCE_COUNT (vble)	CEMT INQ TRAN( )
RETURN_CODE	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 may 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 79 on page 273. In normal operation, the CICS master terminal operator may not wish to inspect the CSMT log when a command has been issued. 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 master terminal control 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 DFHZC4900. The explanatory text that accompanies the message varies and is summarized in Figure 79 on page 273. Refer to the *VSE/ESA Messages and Codes Volume 3* manual for a full description of the message. In certain cases, DFHZC4901 is also issued to give further information.

APPC RETURN_CODE	CICS message
OK	DFHZC4900 result = SUCCESSFUL
ACTIVATION_FAILURE_RETRY	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
ACTIVATION_FAILURE _NO_RETRY	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
ALLOCATION_ERROR	Checked by CEMT. If allocation fails, SYSTEM NOT ACQUIRED is returned to the operator.
COMMAND_RACE_REJECT	DFHZC4900 result = RACE DETECTED
LU_MODE_SESSION_LIMIT _CLOSED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
LU_MODE_SESSION_LIMIT _EXCEEDED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT _NOT_ZERO	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
LU_MODE_SESSION_LIMIT _ZERO	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0
LU_SESSION_LIMIT _EXCEEDED	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = (negotiated value)
PARAMETER_ERROR	Checked by CEMT
REQUEST_EXCEEDS_MAX _MAX_ALLOWED	Checked by CEMT
RESOURCE_FAILURE_NO _RETRY	The LU services manager transaction (CLS1) abends with abend code ATNI.
UNRECOGNIZED_MODE_NAME	DFHZC4900 result=MODENAME NOT RECOGNIZED

Figure 79. Messages triggered by CLS1

CICS deviations from the APPC architecture

This section describes the way in which the CICS implementation of APPC differs from the architecture described in the *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*.

There is one deviation:

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 incorrect combination is found (that is, PARALLEL-SESSIONS specified but CNOS not specified), CICS sends a negative response to the BIND request.

APPC 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 incorrect combination of these indicators.

APPC transaction routing deviations from APPC architecture

This single deviation applies only to APPC transaction routing:

- A transaction program cannot use ISSUE SIGNAL while in syncfree, syncsend, or syncreceive state. Attempting to do so may result in a state check.

Glossary

This glossary contains definitions of those terms and abbreviations that relate specifically to the contents of this book. It also contains terms and definitions from the *IBM Dictionary of Computing*, published by McGraw-Hill.

If you do not find the term you are looking for, refer to the Index or to the *IBM Dictionary of Computing*.

A

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 (ACF/VTAM). A set of programs that control communication between terminals and application programs running under VSE, OS/VS1, MVS, and OS/390.

Advanced Program-to-Program Communication (APPC). The general term chosen for the LUTYPE6.2 protocol under Systems Network Architecture (SNA).

alternate facility. An IRC or SNA session that is obtained by a transaction by means of an ALLOCATE command. Contrast with principal facility.

AOR. Application-owning region.

APPC. Advanced Program-to-Program Communication.

asynchronous processing. (1) A series of operations done separately from the task that requested them. For example, a print job requested by a transaction. (2) In CICS, an intercommunication function that allows a transaction executing on one CICS system to start a transaction on another system. The two transactions execute independently of each other. Compare with *distributed transaction processing*.

ATI. Automatic transaction initiation.

attach header. In SNA, a function management header that causes a remote process or transaction to be attached.

Automatic transaction initiation. The process whereby a transaction request made internally within a CICS system leads to the scheduling of the transaction.

B

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.

C

central processing complex (CPC). A single physical processing system, such as the whole of an ES/9000® 9021 Model 820, or one physical partition of such a machine. A physical processing system consists of main storage, and one or more central processing units (CPUs), time-of-day (TOD) clocks, and channels, which are in a single configuration. A CPC also includes channel subsystems, service processors, and expanded storage, where installed.

CICSplex. (1) A CICS complex. A CICSplex consists of two or more regions that are linked using CICS intercommunication facilities. The links can be either intersystem communication (ISC) or multiregion operation (MRO) links, but within a CICSplex are more usually MRO. Typically, a CICSplex has at least one terminal-owning region (TOR), more than one application-owning region (AOR), and may have one or more regions that own the resources that are accessed by the AORs. (2) The largest set of CICS regions or systems to be manipulated by a single CICSPlex SM entity.

CICSPlex System Manager (CICSPlex SM). An IBM CICS system-management product that provides a single-system image and a single point of control for one or more CICSPlexes.

compute-bound. The property of a transaction whereby the elapsed time for its execution is governed by its computational content rather than by its need to perform input/output.

conversation. A sequence of exchanges between transactions over a session, delimited by SNA brackets.

D

daisy-chain. In CICS intercommunication, the chain of sessions that results when a system requests a resource in a remote system, but the remote system discovers that the resource is in a third system and has itself to make a remote request.

data integrity. The quality of data that exists as long as accidental or malicious destruction, alteration. or loss of data are prevented.

Data Language/I (DL1). An IBM database management facility.

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 program link (DPL). A facility that allows a CICS client program to call a server program running in a remote CICS region, and to pass and receive data using a communications area.

distributed transaction processing (DTP). The distribution of processing between transactions that communicate synchronously with one another over intersystem or interregion links.

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.

DPL. Distributed program link.

DTP. 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.

E

EDF. Execution (command-level) diagnostic facility for testing command-level programs interactively at a terminal.

EIB. EXEC interface block.

EXCI. External CICS interface.

Extended Recovery Facility (XRF). XRF is a related set of programs that allow an installation to achieve a higher level of availability to end users. Availability is improved by having a pair of CICS systems: an active system and a partially initialized alternate system. The alternate system stands by to continue processing if failures occur on the active system.

External CICS interface (EXCI). An application programming interface (API) that enables a VSE/ESA client program (running outside the CICS address space) to call a program running in a CICS Transaction Server for VSE/ESA Release 1 system, and to pass and receive data using a communications area. The CICS program is invoked as if linked-to by another CICS program via a DPL request.

F

file control table (FCT). 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. Contrast with 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 shipping. The process, transparent to the application program, by which CICS accesses resources when those resources are actually held on another CICS system.

G

generalized data stream (GDS). The SNA-defined data stream format used for conversations on APPC sessions.

H

host computer. The primary or controlling computer in a data communication system.

I

IMS/VS. Information Management System/Virtual Storage.

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.

intercommunication facilities. A generic term covering intersystem communication (ISC) and multiregion operation (MRO).

interregion communication (IRC). The method by which CICS implements multiregion operation (MRO).

intersystem communication (ISC). Communication between separate systems by means of SNA networking facilities or by means of the application-to-application facilities of VTAM.

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. Interregion communication.

ISC. Intersystem communication.

L

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 (LU). A port through which a user gains access to the services of a network.

logical unit of work (LUW). A unit of work that can be regarded as a logically-related sequence of actions for the purposes of CICS error recovery mechanisms.

LU. Logical unit.

LUW. Logical unit of work.

LU-LU session. A session between two logical units in an SNA network.

M

macro. In CICS, an instruction similar in format to an assembler language instruction.

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.

mirror transaction. A transaction initiated in a CICS system in response to a function shipping request from another CICS system. The mirror transaction recreates the original request and the request is issued. The mirror transaction returns the acquired data to the originating CICS system.

MRO. Multiregion operation.

multiprogramming. Concurrent execution of application programs across partitions.

multiregion operation (MRO). Communication between CICS systems without the use of SNA networking facilities. CICS Transaction Server for VSE/ESA systems that use MRO must be in the same operating system. (CICS Transaction Server for OS/390, CICS/ESA, CICS/MVS, or CICS/OS/VS systems can use XCF/MRO to communicate across operating systems within the same MVS sysplex.)

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, or MVS/ESA™.

N

national language support (NLS). A CICS feature that enables the user to communicate with the system in the national language chosen by the user.

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.

NLS. National language support.

nonswitched connection. A connection that does not have to be established by dialing.

O

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.

P

partition. A fixed-size subdivision of main storage, allocated to a system task.

pipe. A one-way communication path between a sending process and a receiving process. In the external CICS interface (EXCI), each pipe maps on to one MRO session, where the client program represents the sending process and the CICS server region represents the receiving process.

principal facility. The terminal or logical unit that is connected to a transaction at its initiation. Contrast with alternate facility.

processor. Host processing unit.

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.

R

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 IRC or APPC link.

RU. Request unit.

S

SAA. Systems Application Architecture.

SCS. SNA character stream.

SDLC. Synchronous data link control.

security. Prevention of access to or use of data or programs without authorization.

session. In CICS intersystem communication, an SNA LU-LU session.

shippable terminal. A terminal whose definition can be shipped to another CICS system as and when the other system requires a remote definition of that terminal.

SIT. System initialization table.

SNA. Systems Network Architecture.

startup job stream. 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 transaction-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 APPC session.

syncpoint. Synchronization point. An intermediate point in an application program at which updates or modifications are logically complete.

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 (SIT). A table containing user-specified data that will control a system initialization process.

Systems Application Architecture (SAA). A set of common standards and procedures for working with IBM systems and data. SAA enables different software, hardware, and network environments to coexist. It provides bases for designing and developing application programs that are consistent across different systems.

Systems Network Architecture (SNA). The description of the logical structure, formats, protocols, and operational sequences for transmitting information units through, and controlling the configuration and operation of, networks. The structure of SNA allows the end users to be independent of, and unaffected by, the specific facilities used for information exchange.

T

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, OS/VS, MVS, or System/390®.) (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.

TCT. Terminal control table.

TCTTE. Terminal control table: terminal entry.

temporary storage control. The CICS element that provides temporary data storage facilities.

temporary storage table (TST). 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 (TCT). 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 name. 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 intercommunication facility that allows terminals or logical units connected to one CICS system to initiate and to communicate with transactions in another CICS system. Transaction routing is not possible over LU6.1 links.

transient data control. The CICS element that controls sequential data files and intrapartition data.

TST. Temporary-storage table.

unit-of-recovery descriptor (URD). 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.

V

VSE. Virtual Storage Extended.

VTAM. See *ACF/VTAM*.

X

XRF. Extended Recovery Facility.

Bibliography

CICS Transaction Server for VSE/ESA Release 1 library

Evaluation and planning	
Enhancements Guide	GC34-5763
Release Guide	GC33-1645
Migration Guide	GC33-1646
Report Controller Planning Guide	SC33-1941
General	
Master Index	SC33-1648
Trace Entries	SX33-6108
User's Handbook	SX33-6101
Glossary (softcopy only)	GC33-1649
Administration	
System Definition Guide	SC33-1651
Customization Guide	SC33-1652
Resource Definition Guide	SC33-1653
Operations and Utilities Guide	SC33-1654
CICS-Supplied Transactions	SC33-1655
Programming	
Application Programming Guide	SC33-1657
Application Programming Reference	SC33-1658
Sample Applications Guide	SC33-1713
Application Migration Aid Guide	SC33-1943
System Programming Reference	SC33-1659
Distributed Transaction Programming Guide	SC33-1661
Front End Programming Interface User's Guide	SC33-1662
REXX Guide	SC34-5764
Diagnosis	
Problem Determination Guide	GC33-1663
Messages and Codes Vol 3 (softcopy only)	SC33-6799
Diagnosis Reference	LY33-6085
Data Areas	LY33-6086
Supplementary Data Areas	LY33-6087
Communication	
Intercommunication Guide	SC33-1665
Internet Guide	SC34-5765
CICS Family: Interproduct Communication	SC33-0824
CICS Family: Communicating from CICS on System/390	SC33-1697
Special topics	
Recovery and Restart Guide	SC33-1666
Performance Guide	SC33-1667
Shared Data Tables Guide	SC33-1668
Security Guide	SC33-1942
External Interfaces Guide	SC33-1669
XRF Guide	SC33-1671
Report Controller User's Guide	SC34-5688
CICS Clients	
CICS Clients: Administration	SC33-1792
CICS Universal Clients Version 3 for OS/2: Administration	SC34-5450
CICS Universal Clients Version 3 for Windows: Administration	SC34-5449
CICS Universal Clients Version 3 for AIX: Administration	SC34-5348
CICS Universal Clients Version 3 for Solaris: Administration	SC34-5451
CICS Family: OO programming in C++ for CICS Clients	SC33-1923
CICS Family: OO programming in BASIC for CICS Clients	SC33-1671
CICS Family: Client/Server Programming	SC33-1435
CICS Transaction Gateway Version 3: Administration	SC34-5448

Books from VSE/ESA 2.5 base program libraries

VSE/ESA Version 2 Release 5

Book title	Order number
Administration	SC33-6705
Diagnosis Tools	SC33-6614
Extended Addressability	SC33-6621
Guide for Solving Problems	SC33-6710
Guide to System Functions	SC33-6711
Installation	SC33-6704
Licensed Program Specification	GC33-6700
Messages and Codes Volume 1	SC33-6796
Messages and Codes Volume 2	SC33-6798
Messages and Codes Volume 3	SC33-6799
Networking Support	SC33-6708
Operation	SC33-6706
Planning	SC33-6703
Programming and Workstation Guide	SC33-6709
System Control Statements	SC33-6713
System Macro Reference	SC33-6716
System Macro User's Guide	SC33-6715
System Upgrade and Service	SC33-6702
System Utilities	SC33-6717
TCP/IP User's Guide	SC33-6601
Turbo Dispatcher Guide and Reference	SC33-6797
Unattended Node Support	SC33-6712

High-Level Assembler Language (HLASM)

Book title	Order number
General Information	GC26-8261
Installation and Customization Guide	SC26-8263
Language Reference	SC26-8265
Programmer's Guide	SC26-8264

Language Environment for VSE/ESA (LE/VSE)

Book title	Order number
C Run-Time Library Reference	SC33-6689
C Run-Time Programming Guide	SC33-6688
Concepts Guide	GC33-6680
Debug Tool for VSE/ESA Fact Sheet	GC26-8925
Debug Tool for VSE/ESA Installation and Customization Guide	SC26-8798
Debug Tool for VSE/ESA User's Guide and Reference	SC26-8797
Debugging Guide and Run-Time Messages	SC33-6681
Diagnosis Guide	SC26-8060
Fact Sheet	GC33-6679
Installation and Customization Guide	SC33-6682
LE/VSE Enhancements	SC33-6778
Licensed Program Specification	GC33-6683
Programming Guide	SC33-6684
Programming Reference	SC33-6685
Run-Time Migration Guide	SC33-6687
Writing Interlanguage Communication Applications	SC33-6686

VSE/ICCF

Book title	Order number
Adminstration and Operations	SC33-6738
User's Guide	SC33-6739

VSE/POWER

Book title	Order number
Administration and Operation	SC33-6733
Application Programming	SC33-6736
Networking Guide	SC33-6735
Remote Job Entry User's Guide	SC33-6734

VSE/VSAM

Book title	Order number
Commands	SC33-6731
User's Guide and Application Programming	SC33-6732

VTAM for VSE/ESA

Book title	Order number
Customization	LY43-0063
Diagnosis	LY43-0065
Data Areas	LY43-0104
Messages and Codes	SC31-6493
Migration Guide	GC31-8072
Network Implementation Guide	SC31-6494
Operation	SC31-6495
Overview	GC31-8114
Programming	SC31-6496
Programming for LU6.2	SC31-6497
Release Guide	GC31-8090
Resource Definition Reference	SC31-6498

Books from VSE/ESA 2.5 optional program libraries

C for VSE/ESA (C/VSE)

Book title	Order number
C Run-Time Library Reference	SC33-6689
C Run-Time Programming Guide	SC33-6688
Diagnosis Guide	GC09-2426
Installation and Customization Guide	GC09-2422
Language Reference	SC09-2425
Licensed Program Specification	GC09-2421
Migration Guide	SC09-2423
User’s Guide	SC09-2424

COBOL for VSE/ESA (COBOL/VSE)

Book title	Order number
Debug Tool for VSE/ESA Fact Sheet	GC26-8925
Debug Tool for VSE/ESA Installation and Customization Guide	SC26-8798
Debug Tool for VSE/ESA User’s Guide and Reference	SC26-8797
Diagnosis Guide	SC26-8528
General Information	GC26-8068
Installation and Customization Guide	SC26-8071
Language Reference	SC26-8073
Licensed Program Specifications	GC26-8069
Migration Guide	GC26-8070
Migrating VSE Applications To Advanced COBOL	GC26-8349
Programming Guide	SC26-8072

DB2 Server for VSE

Book title	Order number
Application Programming	SC09-2393
Database Administration	GC09-2389
Installation	GC09-2391
Interactive SQL Guide and Reference	SC09-2410
Operation	SC09-2401
Overview	GC08-2386
System Administration	GC09-2406

DL/I VSE

Book title	Order number
Application and Database Design	SH24-5022
Application Programming: CALL and RQDLI Interface	SH12-5411
Application Programming: High-Level Programming Interface	SH24-5009
Database Administration	SH24-5011
Diagnostic Guide	SH24-5002
General Information	GH20-1246
Guide for New Users	SH24-5001
Interactive Resource Definition and Utilities	SH24-5029
Library Guide and Master Index	GH24-5008
Licensed Program Specifications	GH24-5031
Low-level Code and Continuity Check Feature	SH20-9046
Library Guide and Master Index	GH24-5008
Messages and Codes	SH12-5414
Recovery and Restart Guide	SH24-5030
Reference Summary: CALL Program Interface	SX24-5103
Reference Summary: System Programming	SX24-5104
Reference Summary: HLPI Interface	SX24-5120
Release Guide	SC33-6211

PL/I for VSE/ESA (PL/I VSE)

Book title	Order number
Compile Time Messages and Codes	SC26-8059
Debug Tool For VSE/ESA User's Guide and Reference	SC26-8797
Diagnosis Guide	SC26-8058
Installation and Customization Guide	SC26-8057
Language Reference	SC26-8054
Licensed Program Specifications	GC26-8055
Migration Guide	SC26-8056
Programming Guide	SC26-8053
Reference Summary	SX26-3836

Screen Definition Facility II (SDF II)

Book title	**Order number**
VSE Administrator's Guide	SH12-6311
VSE General Introduction	SH12-6315
VSE Primer for CICS/BMS Programs	SH12-6313
VSE Run-Time Services	SH12-6312

Notices

Programming Interface Information

This book is intended to help you understand how to get CICS systems to communicate with each other and with other systems. This book also documents General-use Programming Interface and Associated Guidance Information. General-use programming interfaces allow the customer to write programs that obtain the services of CICS.

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

Trademarks and service marks

Index

Sending your comments to IBM

CICS® Transaction Server for VSE/ESA™

Intercommunication Guide

SC33-1665-01

If you want to send to IBM any comments you have about this book, please use one of the methods listed below. Feel free to comment on anything you regard as a specific error or omission in the subject matter, and on the clarity, organization or completeness of the book itself.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail:

  IBM UK Laboratories
  Information Development
  Mail Point 095
  Hursley Park
  Winchester, SO21 2JN
  England

- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Email: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Program Number: 5648-054

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1665-01