CICS Transaction Server for z/OS

Intercommunication Guide

Version 3 Release 2

SC34-6829-04

CICS Transaction Server for z/OS

Intercommunication Guide

Version 3 Release 2

SC34-6829-04

Note!
Before using this information and the product it supports, be sure to read the general information under “Notices” on page 345.

This edition applies to Version 3 Release 2 of CICS Transaction Server for z/OS, program number 5655-M15, and to all subsequent versions, releases, and modifications until otherwise indicated in new editions.

Contents

Preface

What this book is about

This book is about:

- Multiregion operation (MRO): communication between CICS® regions in the same operating system, or in the same MVS™ sysplex, without the use of IBM® Systems Network Architecture (SNA) networking facilities.[1]
- Intersystem communication over SNA (ISC over SNA): communication between an IBM CICS Transaction Server for z/OS® region and other (CICS or non-CICS) systems or terminals that support the logical unit type 6.2 or logical unit type 6.1 protocols of SNA. Logical unit type 6.2 protocols are also known as Advanced Program-to-Program Communication (APPC). The remote systems may or may not be in the same MVS sysplex as CICS.
- IP interconnectivity (IPIC): communication between an IBM CICS Transaction Server for z/OS region and other (CICS or non-CICS) systems or terminals that support the Transport Control Protocol/Internet Protocol (TCP/IP). The remote systems may or may not be in the same MVS sysplex as CICS.

What is not covered by this book

The information in this book is predominantly, but not exclusively, about communication between CICS Transaction Server for z/OS, Version 3 Release 2 and other System/390® CICS or IMS™ systems. For supplementary information about communication between CICS TS for z/OS, Version 3.2 and non-System/390 CICS systems, see the *CICS Family: Communicating from CICS on System/390* manual.

Note: In this book, the phrase *System/390* is used as a generic term for computers of the System/370, System/390, and zSeries® families.

For an overview of the intercommunication facilities provided on other CICS products, see the *CICS Family: Interproduct Communication* manual .

For information about accessing CICS programs and transactions from the Internet, see the *CICS Internet Guide*. For information about accessing CICS programs and transactions from other non-CICS environments, see the *CICS External Interfaces Guide* .

For information about CICS support for the CICS Client workstation products, see the *CICS Family: Communicating from CICS on System/390* manual.

For information about the intercommunication aspects of using CICS business transaction services (BTS), see the *CICS Business Transaction Services* manual.

For information about the CICS Front End Programming Interface, see the *CICS Front End Programming Interface User's Guide*.

For information about distributed transaction programming, see the *CICS Distributed Transaction Programming Guide*.

1. The external CICS interface (EXCI) uses a specialized form of MRO link to support: communication between MVS batch programs and CICS; DCE remote procedure calls to CICS programs.

Who this book is for

This book is for customers involved in the planning and implementation of CICS intersystem communication over SNA (ISC over SNA), IP interconnectivity (IPIC), 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. If you plan to create an IPIC network, you will need a knowledge of TCP/IP.

Note: In this book, the term "MVS" refers to those services and functions that are provided by the Base Control Program (BCP) of z/OS. The BCP is a base element of z/OS.

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.

How this book is organized

This book is organized as follows:

Intercommunication concepts and facilities contains an introduction to CICS intercommunication and describes the facilities that are available. It is intended for evaluation and planning purposes.

Installing intercommunication support describes those aspects of CICS installation that apply particularly to intercommunication. It also contains some notes on IMS system definition. This part is intended to be used in conjunction with the *CICS Transaction Server for z/OS Installation Guide* and the *CICS System Definition Guide*.

Defining intercommunication resources provides guidance for resource definition. It tells you how to define links to remote systems, how to define remote resources, and how to define the local resources that are required in an intercommunication environment. It is intended to be used in conjunction with the *CICS Resource Definition Guide*.

Application programming in an intersystem environment describes how to write application programs that use the CICS intercommunication facilities. It is intended to be used in conjunction with the *CICS Application Programming Guide* and the *CICS Application Programming Reference*.

Performance in an intersystem environment describes those aspects of performance that apply particularly in the intercommunication environment. It is intended to be used in conjunction with the *CICS Performance Guide*.

Recovery and restart in an intersystem environment describes those aspects of recovery and restart that apply particularly in the intercommunication environment. It is intended to be used in conjunction with the *CICS Recovery and Restart Guide*.

Terminology

Unless specifically stated otherwise, in this book:

1. The term "*CICS*" means CICS Transaction Server for z/OS, Version 3 Release 2. Where other CICS products are meant, they are named explicitly.
2. The terms "*intersystem communication*" and "*ISC*" are generic names for mean intersystem communication over SNA (ISC over SNA) and IP interconnectivity (IPIC). Where either ISC over SNA or IPIC is meant, it is named explicitly.

   For an explanation of the two types of ISC, see "Communication between systems" on page 4.
3. The term "*IP connection*" means an IP interconnectivity connection.
4. The term "*MVS*" refers to those services and functions that are provided by the Base Control Program (BCP) of z/OS. The BCP is a base element of z/OS.

Summary of changes

This book is based on the CICS Intercommunication Guide for CICS Transaction Server for z/OS, Version 3 Release 1, SC34-6448-00. Changes from that edition are marked by vertical bars in the left margin.

Changes for CICS Transaction Server for z/OS, Version 3 Release 2

For information about changes that have been made in CICS Transaction Server for z/OS, Version 3 Release 2, please refer to *What's New* in the information center, or the following publications:

- *CICS Transaction Server for z/OS Release Guide*
- *CICS Transaction Server for z/OS Migration from CICS TS Version 3.1*
- *CICS Transaction Server for z/OS Migration from CICS TS Version 2.3*
- *CICS Transaction Server for z/OS Migration from CICS TS Version 2.2*
- *CICS Transaction Server for z/OS Migration from CICS TS Version 1.3*

Changes for CICS Transaction Server for z/OS, Version 3 Release 1

There were no major changes for this edition.

Changes for CICS Transaction Server for z/OS, Version 2 Release 3

The more significant changes for this edition were:

- References to the following CICS products were removed, because these products are no longer supported:
  - CICS/ESA Version 4
  - CICS/ESA Version 3
  - CICS/MVS Version 2

Part 1. Intercommunication concepts and facilities

This part of the manual describes the basic concepts of CICS intercommunication and the various facilities that are provided. It defines CICS **intercommunication**, and introduces the two types of intercommunication: **multiregion operation** and **intersystem communication**. It then describes the basic intercommunication facilities that CICS provides. These are:

- Function shipping
- Asynchronous processing
- Transaction routing
- Distributed program link (DPL)
- Distributed transaction processing (DTP).

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. This information is concerned with the role of CICS in a multiple-system environment, in which CICS can communicate with other systems that have similar communication facilities. This sort of communication is called *CICS intercommunication*.

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

Important: The information in this document is mainly about communication between CICS Transaction Server for z/OS and other System/390), CICS or IMS, systems. For supplementary information about communication between CICS Transaction Server for z/OS and non-System/390 CICS systems, see the *CICS Family: Communicating from CICS on zSeries* manual.

For information about CICS Transaction Server for z/OS's support for the CICS Client workstation products, see the *CICS Family: Communicating from CICS on zSeries* manual.

For information about accessing CICS programs and transactions from the Internet, see the *CICS Internet Guide*. For information about accessing CICS programs and transactions from other non-CICS environments, see the *CICS External Interfaces Guide*.

This section contains the following topics:

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 does not require the use of a network access method such as ACF/VTAM or TCP/IP. This form of communication is called **multiregion operation** (MRO). MRO can be used between CICS regions that reside:

- In the same z/OS image
- In the same z/OS systems complex (**sysplex**).

CICS Transaction Server for z/OS can use MRO to communicate with:

- Other CICS Transaction Server for z/OS systems
- CICS Transaction Server for OS/390® systems

Note: The external CICS interface (EXCI) uses a specialized form of MRO link to support:

- Communication between MVS batch programs and CICS

- DCE remote procedure calls to CICS programs

Communication between systems

For communication between CICS and non-CICS systems, or between CICS systems that are not in the same operating system or z/OS sysplex, you normally require a network access method to provide the necessary communication protocols. CICS TS for z/OS, Version 3.2 supports two such **intercommunication facilities**:

1. ACF/VTAM, which implements the IBM Systems Network Architecture (SNA)
2. Transport Control Protocol/Internet Protocol (TCP/IP)

The generic name for communication between systems over SNA is **intersystem communication (ISC)**. Communication between systems over TCP/IP is known as **IP interconnectivity (IPIC)**.

Note: Typically, ISC and IPIC are used to connect CICS and non-CICS systems, or CICS systems that are not in the same z/OS image or sysplex. These intercommunication facilities can also be used between CICS regions in the same z/OS image or sysplex. As an example, you might create an ISC connection between two CICS regions in the same sysplex if you required two connections between them and there was already an MRO connection.

ISC and IPIC offer different levels of function:

`Intersystem communication over SNA (ISC over SNA)`
ISC can be used between CICS and any other system that supports VTAM Advanced Program-to-Program Communication (APPC) or Logical Unit Type 6.1 (LUTYPE6.1) communications. For example, ISC over SNA connections can exist between CICS regions running in different z/OS sysplexes or on different operating system platforms, between CICS and any APPC device, and between CICS and IMS. All the base CICS intercommunication functions (described later) are supported.

CICS Transaction Server for z/OS can use ISC over SNA to communicate with:

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

`IP interconnectivity (IPIC)`
IPIC can be used between CICS and another CICS TS for z/OS, Version 3.2 (or later) region. The remote region may or may not be in the same z/OS sysplex. Currently, the only function supported is distributed program link (DPL).

Additional information about ISC over SNA and IPIC is in Chapter 3, “ISC and IPIC intercommunications facilities,” on page 19.

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.

For communicating with other CICS, IMS, or other systems, CICS provides these basic types of facility:

- Function shipping
- Asynchronous processing
- Transaction routing
- Distributed program link (DPL)
- Distributed transaction processing (DTP).

Note: The following intercommunication facilities, that support access to CICS programs and transactions from non-CICS environments, are not described in this book, but in Overview of CICS external interfaces , in the *CICS External Interfaces Guide*. or ../com.ibm.cics.ts.internet.doc/topics/dfhtl_part1.dita#dfhtl50 , in the *CICS Internet Guide*:

- The CICS bridge
- The external CICS interface
- Support for DCE remote procedure calls
- Support for ONC remote procedure calls
- The CICS Web Interface.

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

Table 1. Support for CICS basic intercommunication facilities, when communicating with other CICS, IMS, APPC, or TCP/IP systems

	IRC Inter region	**Intersystem communication over SNA (via ACF/VTAM)**				**Intersystem communication over TCP/IP**	
	MRO	**LUTYPE6.2 (APPC)**		**LUTYPE6.1**		**TCP/IP**	
Facility	**CICS**	**CICS**	**Non-CICS**	**CICS**	**IMS**	**CICS**	**Non-CICS**
Function Shipping	Yes	Yes	No	Yes	No	No	No
Asynchronous Processing	Yes	Yes	No	Yes	Yes	No	No
Transaction Routing	Yes	Yes	No	No	No	No	No
Distributed program link	Yes	Yes	No	No	No	Yes	No
Distributed transaction processing	Yes	Yes	Yes	Yes	Yes	No	No

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.

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 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.

DPL is supported over IPIC, as well as ISC over SNA, connections.

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.

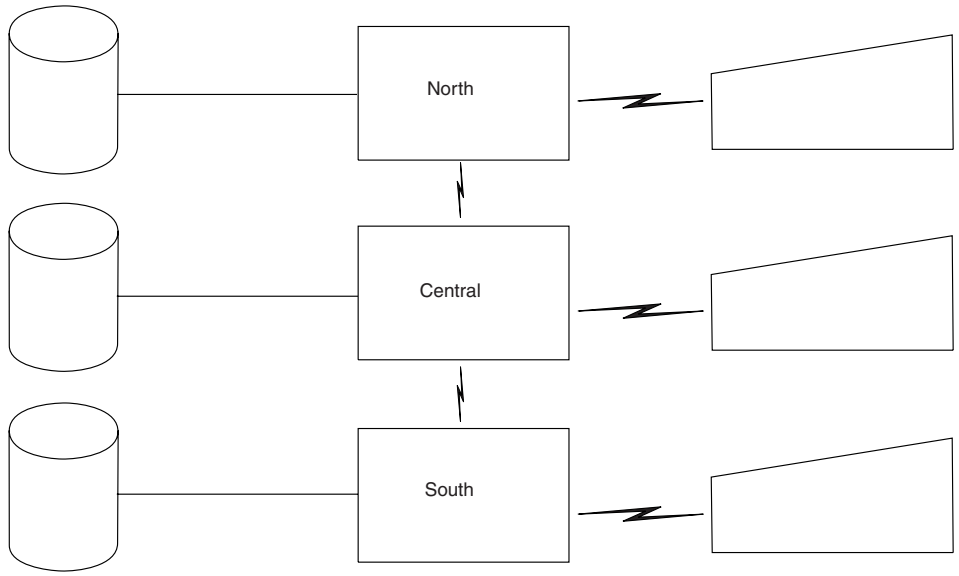
Multiregion operation makes it possible for two CICS regions to share selected system resources, and to present a “single-system” view to terminal operators. At the same time, each region can run independently of the other, and can be protected against errors in other regions. Various possible applications of MRO are described in Chapter 2, “Multiregion operation,” on page 11.

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

Note: You can also use IP interconnectivity to connect distributed systems, but currently the only base CICS intercommunication function it supports is distributed program link.

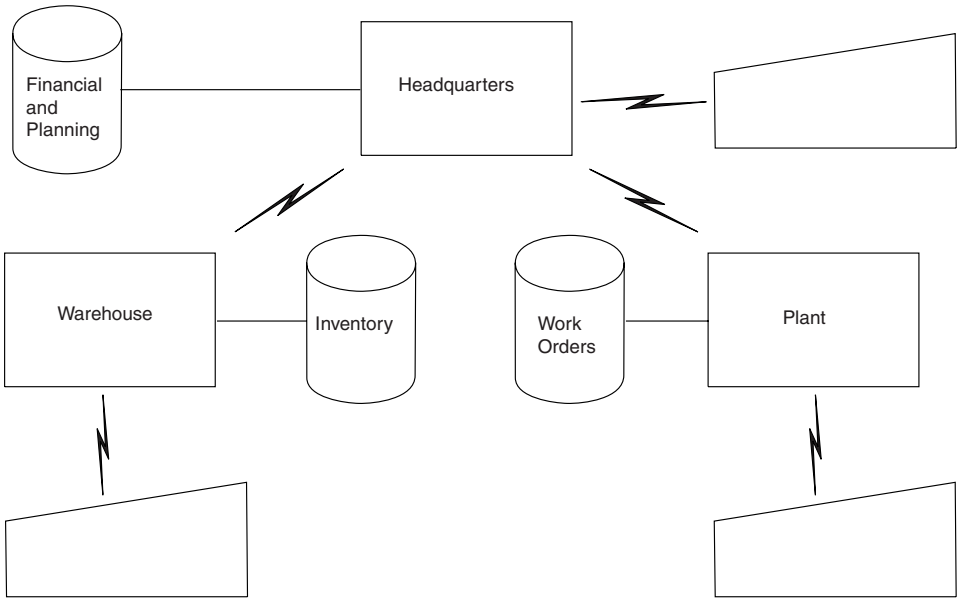
Figure 1 on page 8 shows some typical possibilities.

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. Examples of distributed resources (Part 1)

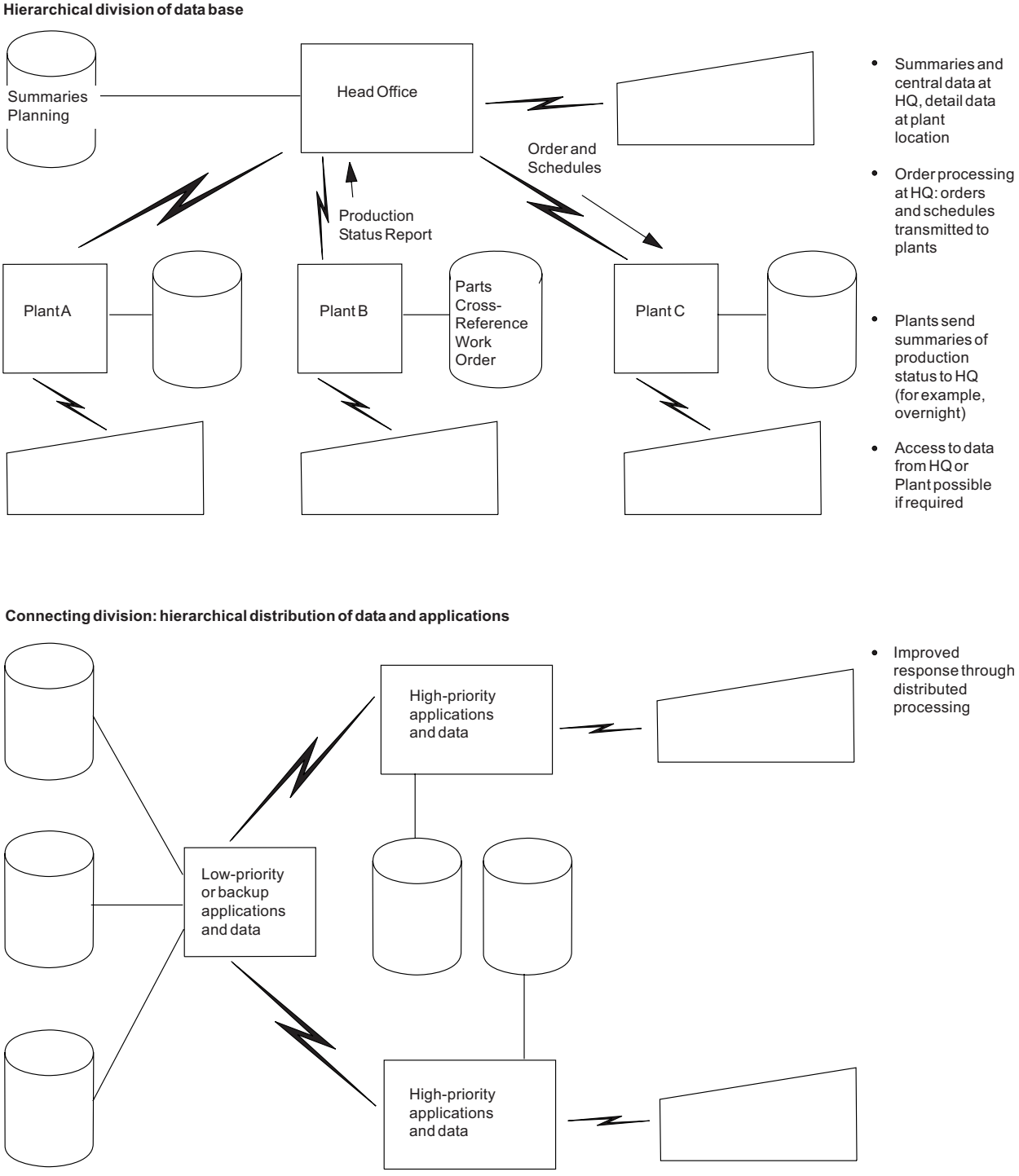


Figure 2. Examples of distributed resources (Part 2)

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.

Chapter 2. Multiregion operation

This chapter contains the following topics:

Overview of MRO

CICS multiregion operation (MRO) enables CICS systems that are running in the same MVS image, or in the same MVS sysplex, to communicate with each other. MRO does not support communication between a CICS system and a non-CICS system such as IMS.

Note: The external CICS interface (EXCI) uses a specialized form of MRO link to support:

- Communication between MVS batch programs and CICS
- DCE remote procedure calls to CICS programs.

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 can be implemented in three ways:

- Through support in CICS terminal control management modules and by use of a CICS-supplied interregion program (DFHIRP) loaded in the link pack area (LPA) of MVS. DFHIRP is invoked by a type 3 supervisor call (SVC). For convenience, we shall call this implementation of multiregion operation MRO(IRC), because you select it by specifying ACCESSMETHOD(IRC) on the CONNECTION definition: see "Choosing the access method for MRO" on page 147.
- By MVS cross-memory (XM) services, which you can select as an alternative to the CICS type 3 SVC mechanism: see "Choosing the access method for MRO" on page 147. Here, DFHIRP is used only to open and close the interregion links.
- By the cross-system coupling facility (XCF) of IBM MVS/ESA. XCF is required for MRO links between CICS regions in different MVS images of an MVS sysplex. It is selected dynamically by CICS for such links, if available. For details of the benefits of cross-system MRO, see "Benefits of XCF/MRO" on page 15.

CICS regions linked by MRO can be at different release levels; see "Multiregion operation" on page 3. *If an MVS image contains different releases of CICS, all using MRO to communicate with each other (or XCF/MRO to communicate with regions in other images in the sysplex), the DFHIRP module in the MVS LPA should be that from the most current CICS release in the image, or higher.* For full details of software and hardware requirements for XCF/MRO, see "Requirements for XCF/MRO" on page 108.

Installation of CICS multiregion operation is described in Chapter 10, "Installation considerations for multiregion operation," on page 107.

Facilities available through MRO

The intercommunication facilities available through MRO are:

- Function shipping
- Asynchronous processing
- Transaction routing
- Distributed program link
- Distributed transaction processing.

These are described under “Intercommunication facilities” on page 5.

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

Cross-system multiregion operation (XCF/MRO)

The cross-system coupling facility (XCF) is part of the MVS/ESA base control program, providing high performance communication links between MVS images that are linked in a sysplex (**sys**tems com**plex**) by channel-to-channel links, ESCON® channels, or coupling facility links.

The IRC provides an XCF access method that makes it unnecessary to use VTAM® to communicate between MVS images within the same MVS sysplex.

Each CICS region is assigned to an XCF group when it logs on to IRC, even if it is not currently connected to any regions in other MVS images. You specify the name of the XCF group on the XCFGROUP system initialization parameter. If XCFGROUP is not specified, the region becomes a member of the default CICS XCF group, DFHIR000.

When members of a CICS XCF group that are in different MVS images talk to each other, CICS selects the XCF access method dynamically, overriding the access method specified on the connection resource definition. The use of the MVS cross-system coupling facility enables MRO to function *between* MVS images in a sysplex environment, supporting all the usual MRO operations, [2] such as:

- Function shipping
- Asynchronous processing
- Transaction routing
- Distributed program link
- Distributed transaction processing.

Each CICS region can be a member of only one XCF group, which it joins when it logs on to IRC. The maximum size of an XCF group is limited by the MVS MAXMEMBER parameter, and there is an absolute limit of 2047 members. If this limit is a problem for you (because, for example, it limits the number of CICS regions you can have in your sysplex), you can create multiple XCF groups, each containing a different set of regions. You could, for example, have one XCF group for production regions and another for development and test regions. If you do need to have multiple XCF groups, it is recommended that:

2. XCF/MRO does not support accessing shared data tables across MVS images. Shared access to a data table, across two or more CICS regions, requires the regions to be in the same MVS image. To access a data table in a different MVS image, you can use function shipping.

- You put your production regions in a different XCF group from your development and test regions
- You do not create more XCF groups than you need: two, separated as described, may be sufficient
- You try not to move regions between XCF groups
- You try not to add or remove regions from existing XCF groups

Note that CICS regions can use MRO or XCF/MRO to communicate *only with regions in the same XCF group*. Members of different XCF groups cannot communicate via MRO (or XCF/MRO), *even if they are in the same MVS image*.

CICS regions linked by XCF/MRO can be at different release levels; see "Multiregion operation" on page 3. Depending on the versions of CICS installed in the MVS images participating in XCF/MRO, the versions of DFHIRP installed in the link pack areas of the MVS images can be different. If a single MVS image contains different releases of CICS, all using XCF/MRO to communicate with regions in other images in the sysplex, the DFHIRP module in the MVS LPA should be that from the most current CICS release in the image, or higher. However, note that the CICS TS for z/OS, Version 3.2 version of DFHIRP (required for multiple XCF group support) can be used only on z/OS Version 1.7 or later. For full details of software and hardware requirements for XCF/MRO, see "Requirements for XCF/MRO" on page 108.

Figure 3 on page 14 is a simple example of the use of XCF/MRO in a sysplex environment. In this example, there is only one CICS XCF group, DFHIR000. The members of DFHIR000 can communicate via XCF/MRO links across the two MVS images.

The MRO links between CICS1 and CICS2, and between CICS3 and CICS4, use either the IRC or XM access methods, as defined for the link. The MRO links between CICS regions on MVS1 and the CICS regions on MVS2 use the XCF method, which is selected by CICS dynamically.

In each MVS, the DFHIRP module in the LPA should be at the level of the highest CICS TS for z/OS release in the image.

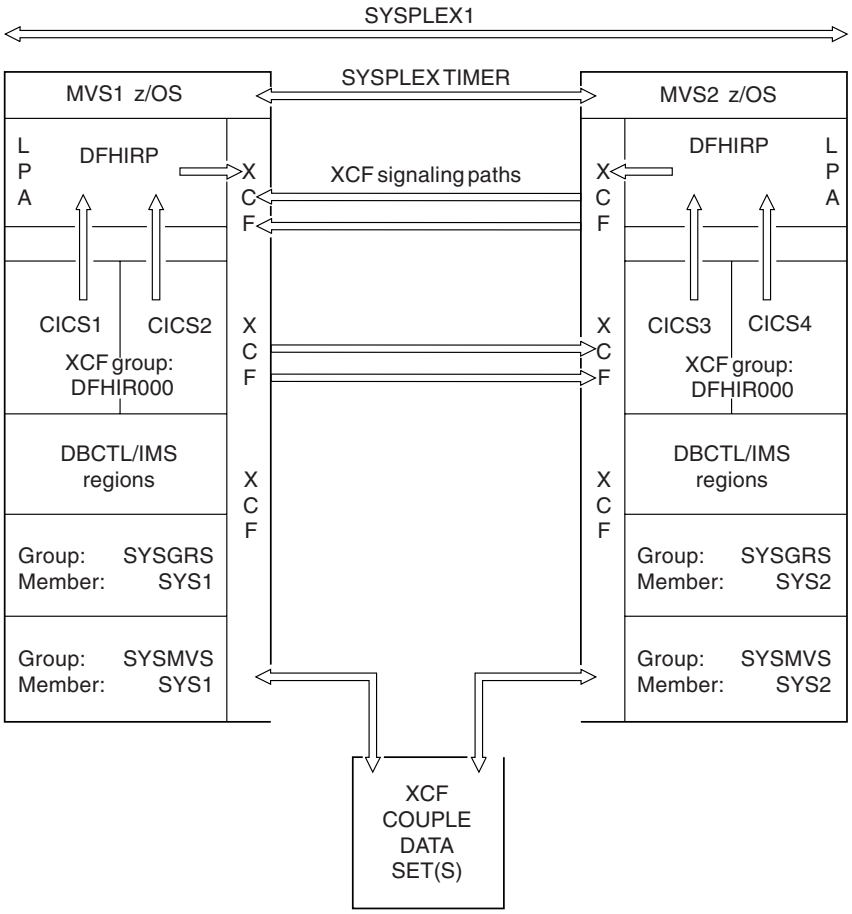


Figure 3. A sysplex (SYSPLEX1) containing a single CICS XCF group.

Figure 4 on page 15 is a slightly more complex example. In this case, there are two CICS XCF groups, DFHIR000 and DFHIR001. The members of each XCF group can communicate across the MVS images by means of XCF/MRO links.

To support multiple CICS XCF groups, both MVS images must be z/OS Version 1.7 or later and must use the CICS TS for z/OS, Version 3.2 or later version of DFHIRP. (Although z/OS has supported multiple XCF groups since Version 1.6, CICS TS for z/OS, Version 3.2 (required to join an XCF group other than DFHIR000) requires z/OS Version 1.7 or later.)

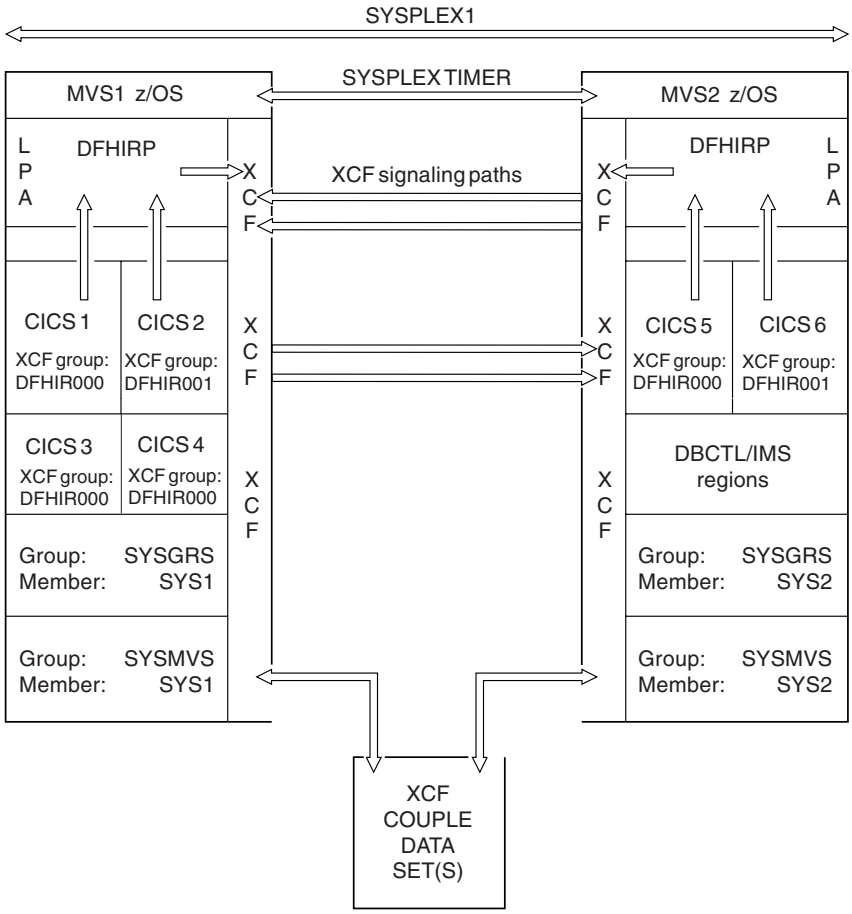


Figure 4. A sysplex (SYSPLEX1) containing two CICS XCF groups

Note that, in Figure 4:

- The members of the DFHIR000 XCF group in MVS1 (CICS 1, CICS 3, and CICS 4) use XCF/MRO, which is selected by CICS dynamically, to communicate with the member of the DFHIR000 XCF group in MVS2 (CICS 5). Similarly, CICS 2 in MVS1 uses XCF/MRO to communicate with CICS 6 in MVS 2; they are both members of the DFHIR001 group.
- CICS 1, CICS 3, and CICS 4 cannot use XCF/MRO to communicate with CICS 6, because CICS 6 is in a different XCF group. Similarly, CICS 2 cannot use XCF/MRO to communicate with CICS 5.
- Because they are in the same MVS image and the same XCF group, CICS 1, CICS 3, and CICS 4 can communicate with each other using either the MRO(IRC) or MRO(XM) access method, as defined for the links.
- CICS 5 cannot use any form of MRO to communicate with CICS 6, even though they are in the same MVS image, because they are in different XCF groups. Similarly, CICS 2 cannot use any form of MRO to communicate with CICS 1, CICS 3, or CICS 4.

Benefits of XCF/MRO

Some of the benefits of cross-system MRO using XCF links are:

- A low communication overhead between MVS images, providing much better performance than using ISC links to communicate between MVS systems. XCF/MRO thus improves the efficiency of transaction routing, function shipping, asynchronous processing, and distributed program link across a sysplex. (You

can also use XCF/MRO for distributed transaction processing, provided that the LUTYPE6.1 protocol is adequate for your purpose.)

- Easier connection resource definition than for ISC links, with no VTAM tables to update.
- Good availability, by having alternative processors and systems ready to continue the workload of a failed MVS or a failed CICS.
- Easy transfer of CICS systems between MVS images. The simpler connection resource definition of MRO, and having no VTAM tables to update, makes it much easier to move CICS regions from one MVS to another. You no longer need to change the connection definitions from CICS MRO to CICS ISC (which, in any event, can be done only if CICS startup on the new MVS is a warm or cold start).
- Improved price and performance, by coupling low-cost, rack-mounted, air-cooled processors (in an HPCS environment).
- Growth in small increments.
- Organizational benefits. Because regions in different XCF groups cannot communicate over MRO (or XCF/MRO), each group of regions is effectively insulated from the others. This can be useful if, for example, you want to prevent (possibly accidental) access to production regions from development or test regions.

Applications of multiregion operation

This section describes some typical applications of multiregion operation.

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

Reliable database access

You can use the storage protection and transaction isolation facilities of CICS Transaction Server for z/OS, Version 3 Release 2 to guard against unreliable applications that might otherwise bring down the system or disable other applications. However, you could use MRO to extend the level of protection.

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 7, “CICS transaction routing,” on page 55.

Workload balancing in a sysplex

In a sysplex, you can use MRO and XCF/MRO links to create a CICSplex consisting of sets of functionally-equivalent terminal-owning regions (TORs) and application-owning regions (AORs).

You can then perform workload balancing using:

- The VTAM generic resource function
- Dynamic transaction routing
- Dynamic routing of DPL requests
- The CICSPlex® System Manager (CICSPlex SM)
- The MVS workload manager.

A VTAM application program such as CICS can be known to VTAM by a generic resource name, as well as by the specific network name defined on its VTAM APPL definition statement. A number of CICS regions can use the same generic resource name.

A terminal user, wishing to start a session with a CICSplex that has several terminal-owning regions, uses the generic resource name in the logon request. Using the generic resource name, VTAM is able to select one of the CICS TORs to be the target for that session. For this mechanism to operate, the TORs must all register to VTAM under the same generic resource name. VTAM is able to perform workload balancing of the terminal sessions across the available terminal-owning regions.

The terminal-owning regions can in turn perform workload balancing using dynamic transaction routing. Application-owning regions can route DPL requests dynamically. The CICSPlex SM product can help you manage dynamic routing across a CICSplex.

For further information about VTAM generic resources, see the *VTAM Version 4 Release 2 Release Guide*. Dynamic routing of DPL requests is described in "Dynamically routing DPL requests" on page 88. Dynamic transaction routing is described in "Dynamic transaction routing" on page 57. For information on managing workloads using CICSPlex SM, see *CICSPlex System Manager Managing Workloads*. For information about the MVS workload manager, see the *CICS Performance Guide*.

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.

Note: If you are using DL/I databases, and want to split your system to avoid virtual storage constraints, consider using DBCTL, rather than CICS function shipping, to share the databases between your CICS address spaces.

Conversion from single-region system

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

CICS function 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 19, "Application programming for CICS function shipping," on page 231. 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 Transaction Server for z/OS Migration from CICS TS Version 3.1* manual. In addition, the restrictions that apply are given in Chapter 22, "Application programming for CICS transaction routing," on page 241.

It is always necessary to define an MRO link between the two regions and to provide local and remote definitions of the shared resources.

Chapter 3. ISC and IPIC intercommunications facilities

As noted in "Communication between systems" on page 4, CICS provides intercommunications facilities for:

Intersystem communication over SNA (ISC over SNA)
ISC over SNA implements the IBM Systems Network Architecture (SNA), which defines data formats and communication protocols for communication between systems in a multiple-system environment. It can be used between CICS and any other system that supports APPC or LUTYPE6.1 communications. It supports all the base CICS intercommunication functions.

IP interconnectivity (IPIC)
IPIC can be used between CICS and other CICS TS for z/OS, Version 3.2 (or later) regions; and between CICS TS and Java clients. On CICS-to-CICS links, the only one of the base CICS intercommunication functions currently supported is DPL.

This chapter contains the following topics:

Intersystem communication over SNA

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

This chapter contains the following topics:

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 5 on page 20.

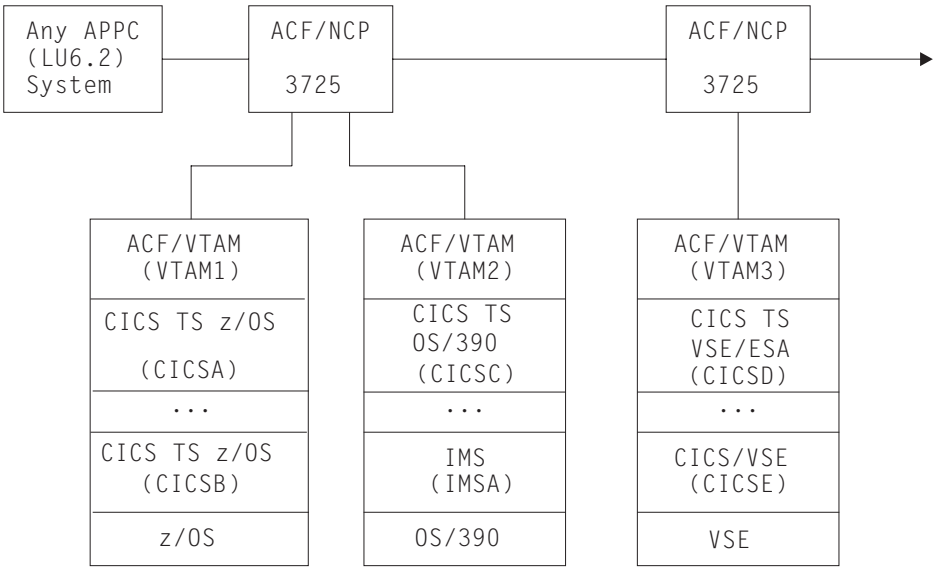


Figure 5. A possible configuration for intercommunicating systems

Single operating system

ISC within a single operating system (intrahost ISC) is possible through the application-to-application facilities of ACF/VTAM. In Figure 5, these facilities can be used to communicate between CICSA and CICSB, between CICSC and IMSA, and between CICSD and CICSE.

In an MVS system, you can use intrahost ISC for communication between two or more CICS systems (although MRO is a more efficient alternative) or between, for example, a CICS system and an IMS system.

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 5) 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 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 5, CICSD and CICSE can be connected to CICSA, CICSB, and CICSC in this way. Each participating system is appropriately configured for its particular location, using MVS or Virtual Storage Extended (VSE) 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 z/OS can connect to via ISC, see “Communication between systems” on page 4. For detailed information about using ISC to connect CICS Transaction Server for z/OS to other CICS products, see the *CICS Family: Communicating from CICS on zSeries* 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.

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

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

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

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 forerunner of LUTYPE6.2 (APPC).

LUTYPE6.1 sessions are supported by both CICS and IMS, so can be used for CICS-to-IMS communication. (For CICS-to-CICS communication, LUTYPE6.2 is the preferred protocol.)

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 “ACF/VTAM LOGMODE table entries for CICS” on page 112.)

Limited resources

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

Both single- and multi-session connections may use limited resources. For a multi-session connection, CICS does not unbind LU service-manager sessions until all modegroups in the connection have performed initial “change number of sessions” (CNOS) exchange. When CICS unbinds a session, CICS tries to balance the contention winners and losers. This may result in CICS resetting an unbound session to be neither a winner nor a loser.

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 155.) 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 13, “Defining links to remote systems,” on page 143.
- 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.

IP interconnectivity

Synchronization levels

IPIC connections support synchronization level 2. That is, they support full CICS syncpointing, including rollback.

System initialization parameters

IP interconnectivity requires CICS TCP/IP services to be activated. To activate them at CICS startup, specify TCPIP=YES as a system initialization parameter. (The default value of the TCPIP parameter is NO.)

For reference information about the TCPIP system initialization parameter, see TCPIP, in the *CICS System Definition Guide*.

Establishing intersystem sessions

How to define an IPIC connection between two CICS regions is described in “Defining IP interconnectivity links” on page 151.

Chapter 4. CICS function shipping

This chapter contains the following topics:

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 5, "Asynchronous processing," on page 37.

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 6 on page 26. 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.

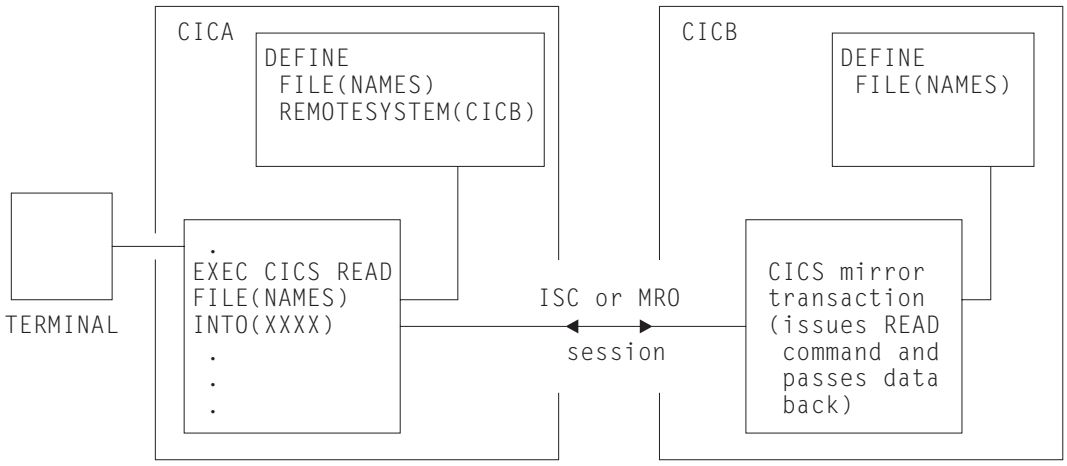


Figure 6. Function shipping

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. (Details are given in Chapter 16, “Defining remote resources,” on page 191.)

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 VSAM or BDAM files located on a remote CICS system. INQUIRE FILE, INQUIRE DSNAME, SET FILE, and SET DSNAME are not supported. Both read-only and update requests are allowed, and the files can be defined as protected in the system on which they reside. Updates to remote protected files 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 unit of work, even if the remote files are located on more than one connected CICS system.

Important:

Take care when designing systems in which remote file requests using physical record identifier values are employed, such as VSAM RBA,

BDAM, 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.

DL/I

Function shipping allows a CICS transaction to access IMS/ESA DM and IMS/VS DB databases associated with a remote CICS OS/390 system, or DL/I DOS/VS databases associated with a remote CICS/VSE system. (See Chapter 1, "Introduction to CICS intercommunication," on page 3 for a list of systems with which CICS Transaction Server for z/OS, Version 3 Release 2 can communicate.)

The IMS/ESA DM (DL/I) database associated with a remote CICS Transaction Server for z/OS 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 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 unit of work.

The PSB directory list (PDIR) 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.

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 definition of the queue in the requesting system defines it as being on the remote system. The definition of the queue in the remote system specifies its recoverability attributes, 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 trigger level 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 region. Requests that are to be function shipped to a resource-owning region may be queued if all bound contention winner sessions are busy, so that no sessions are immediately available. If the resource-owning region is unresponsive the queue can become so long that the performance of the issuing region is severely impaired. Further, if the issuing region is an application-owning region, its impaired performance can spread back to the terminal-owning region.

Note: "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.

The symptoms of this impaired performance are:

- The system reaches its maximum transactions (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 regions, 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 Intersystem communication program exits XISCONA and XISLCLQ, in the *CICS Customization Guide*. For information on the statistics records that are passed to your exit program, refer to ../com.ibm.cics.ts.performance.doc/topics/dfht3_stats_intro.dita , in 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 function shipping, DPL, transaction routing, and distributed transaction processing requests, whereas XISCONA is driven only for function shipping and DPL. If you enable both exits, XZIQUE and XISCONA could both be driven for function shipping and DPL requests, which is not recommended.

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

For further information about controlling intersystem queues, see Chapter 24, “Intersystem session queue management,” on page 267.

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 220). Details of the supplied mirror transactions are given in Chapter 17, “Defining local resources,” on page 217. 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 31.

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 7 on page 30), 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. For guidance about DL/I request processing, see the *CICS IMS Database Control Guide*.

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.

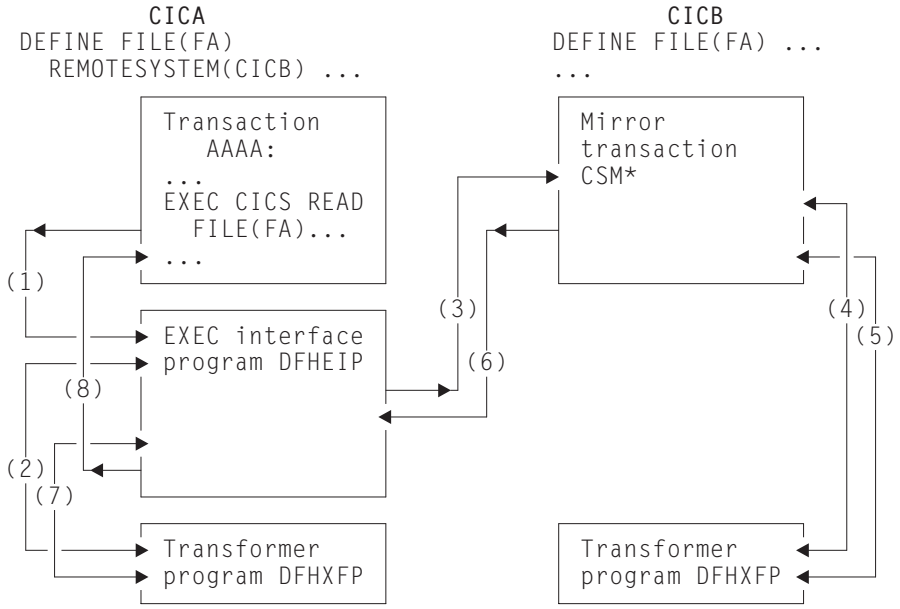


Figure 7. The transformer program and the mirror in function shipping

The mirror transaction uses the function-shipping transformer program, DFHXFP, to decode the formatted request (line 4 in Figure 7). 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 “ISC function shipping” on page 29). 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 Figure 9 on page 34 and Figure 10 on page 34 in “Function shipping examples” on page 33 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.

Note: Do not specify MROFSE=YES in the front-end region when long-running tasks may be used to function-ship requests. This is because a SEND session will be unavailable for allocation to other tasks when unused. Specifying MROFSE=YES may prevent the connection from being released, when contact has been lost with the back-end region, until the task terminates or issues a function-shipped request.

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.

Handling errors and failure of the mirror transaction

If the mirror task in the remote region encounters an error or abend, and that error or abend can be handled by the mirror program, then the error or abend is returned to the application program that issued the function-shipped request.

The remote mirror (server) task, and the task executing the program that issued the request (client task), will share a common transaction scope unless the request was one of the following requests.

- A function-shipped EXEC CICS START NOCHECK command.
- A distributed program link (DPL) request with SYNCONRETURN.
- A non-update request, for example, a file control read only.

If the server task performed recoverable work as part of such a common transaction scope then, even though an error or abend was encountered, that work will be committed or backed out under the control of the syncpoint processing of the client task. The default action is for the error or abend to cause abnormal termination of the client task and back out of all recoverable updates made by both the client and server programs

However, in common with purely local execution (ie when not using function-shipping or distributed program link), the application program that issued the request that was function-shipped might attempt to handle the error or abend. The handle logic will subsequently issue an **EXEC CICS SYNCPOINT** , **SYNCPOINT ROLLBACK**, **RETURN** or **ABEND** command. Attempting a **SYNCPOINT** or **RETURN**, (rather than a **SYNCPOINT ROLLBACK** or **ABEND**), despite being informed of the error or abend,

will attempt to commit the client program's local resource updates and those performed by the server transaction before the error or abend was encountered.

If the error or abend encountered by the mirror transaction can not be handled by the mirror program and this causes the termination and backout of the mirror transaction without sending a response to the client application, then CICS will force the client program's transaction to back out. Any explicit syncpoint attempt will fail and the local updates will be backed out. This is also true if a problem is encountered with the communications link between the client and server tasks.

If the client and server tasks do not share a common transaction scope, as described previously, then errors or abends that result in the termination of the server task, and problems with the communications link, do not force the client's transaction to back out.

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.

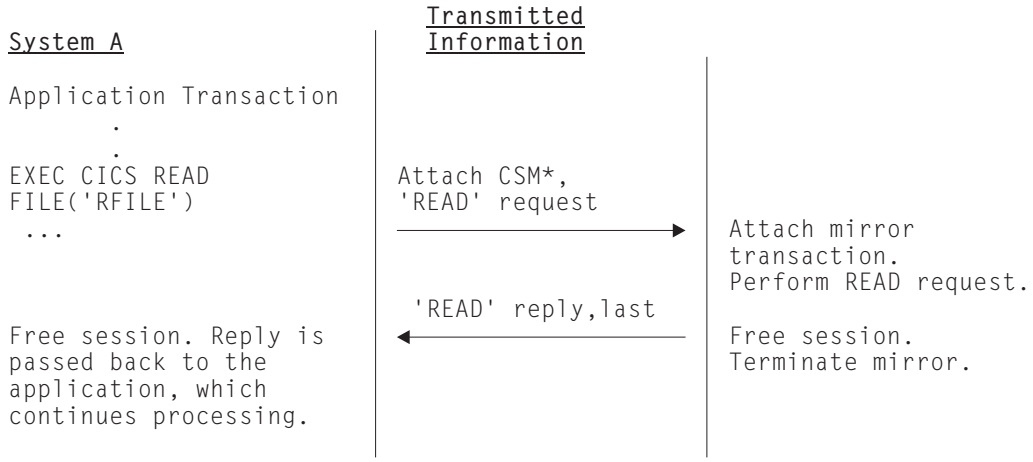


Figure 8. ISC function shipping—simple inquiry. Here no resource is being changed; the session is freed and the mirror task is terminated immediately.

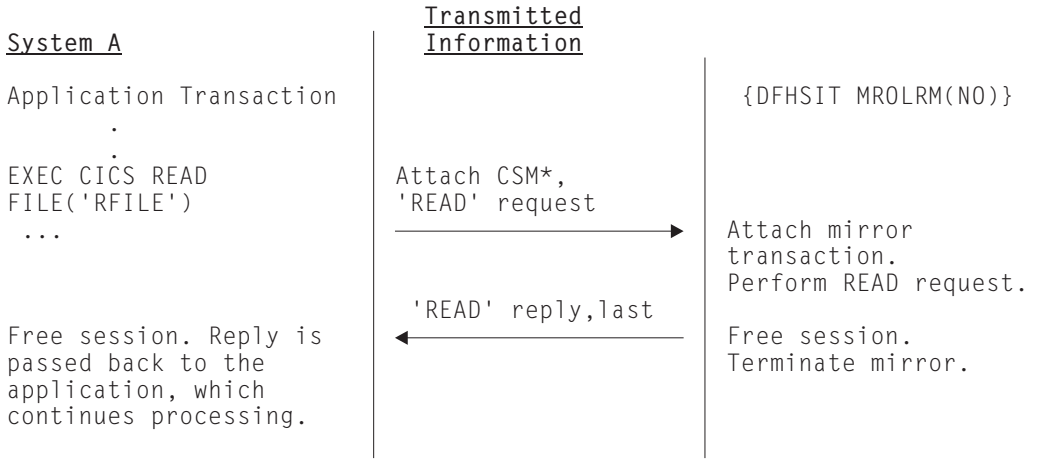


Figure 9. MRO function shipping—simple inquiry. Here no resource is being changed. Because long-running mirror tasks are not specified, the session is freed by System B and the mirror task is therefore terminated immediately.

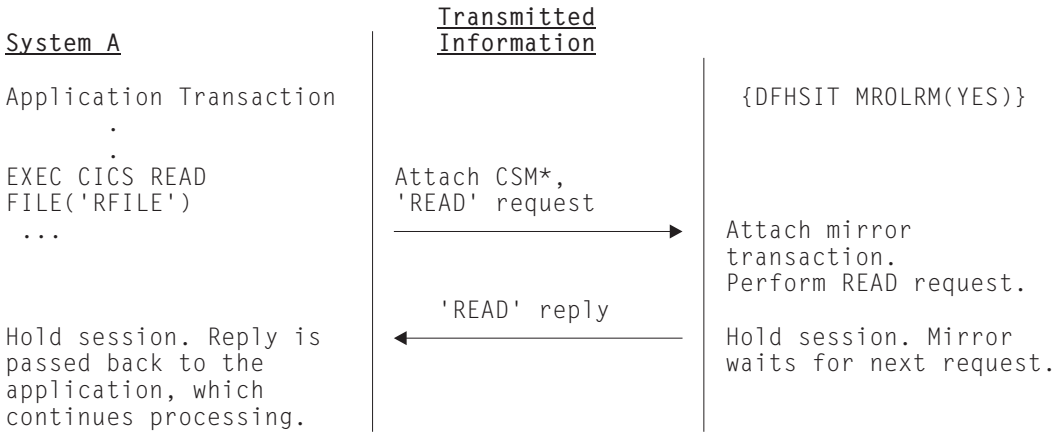


Figure 10. MRO function shipping—simple inquiry. Here no resource is being changed. However, because long-running mirror tasks are specified, the session is held by System B, and the mirror task waits for the next request.

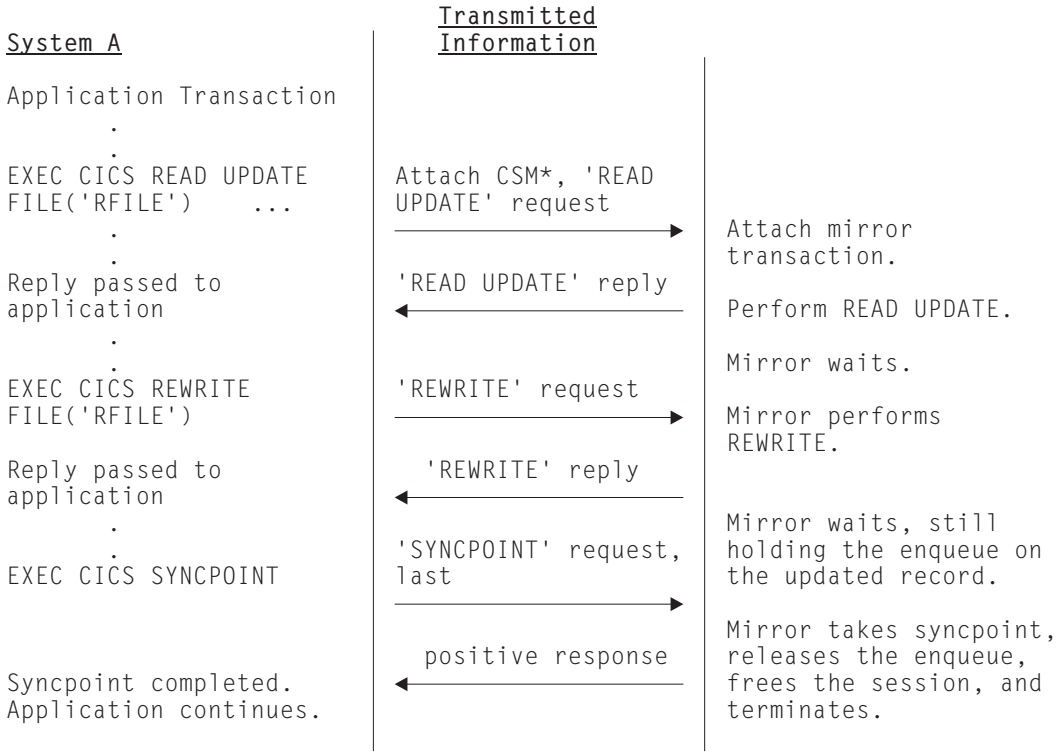


Figure 11. ISC or MRO function shipping—update. Because the mirror must wait for the REWRITE, it becomes long-running and is not terminated until SYNCPOINT is received. Note that the enqueue on the updated record would not be held beyond the REWRITE command if the file was not recoverable.

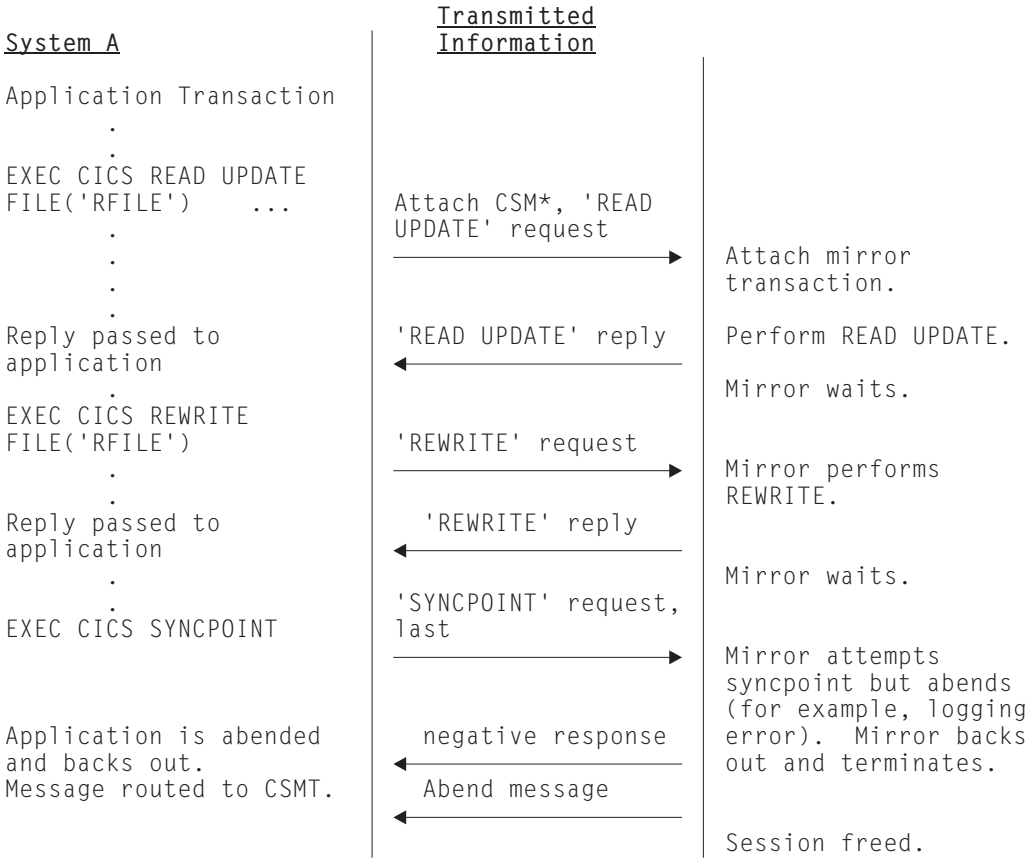


Figure 12. ISC or MRO function shipping—update with ABEND

Figure 12 is similar to Figure 11 on page 35, except that an abend occurs during syncpoint processing.

Chapter 5. Asynchronous processing

This chapter contains the following topics:

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 13.

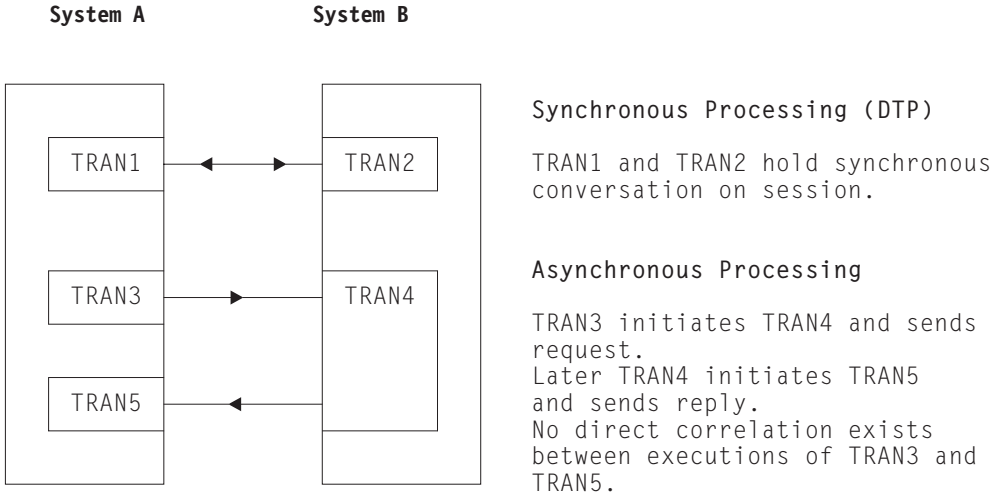


Figure 13. Synchronous and asynchronous processing compared

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 START and RETRIEVE.

   You can use the START command to schedule a transaction in a remote system in much the same way as you would in a single CICS system. This type of asynchronous processing is 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 START command for asynchronous processing, you can communicate only with systems that support the special protocol needed for function shipping; that is, CICS itself and IMS.

   A CICS transaction that is initiated by a remotely-issued start request can use the RETRIEVE command to retrieve any data associated with the request. 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 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 SEND LAST command answered by a single RECEIVE, this book refers to it elsewhere as the SEND/RECEIVE interface. The circumstances under which it is used are described in Chapter 23, “CICS-to-IMS applications,” on page 245.

The remainder of this chapter is devoted to asynchronous processing using START and RETRIEVE commands. Distributed transaction processing is described in Chapter 9, “Distributed transaction processing,” on page 95.

Asynchronous processing using START and RETRIEVE commands

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

- START
- CANCEL
- RETRIEVE.

Starting and canceling remote transactions

Note:

> For information about canceling dynamically-routed START commands, see “Canceling interval control requests” on page 76.

The interval control 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 START command on that system.

For CICS-to-CICS communication, you can include time-control information on the shipped START command in the normal way, by means of the INTERVAL or TIME 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 a START command specifies the time at which the remote transaction is to be initiated, not the time at which the request is to be shipped to the remote system.

A START command shipped to a remote CICS system can be canceled at any time up to its expiration time by shipping a CANCEL command to the same system. The particular START command 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 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 START command

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

- User data—specified in the FROM option.

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

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

  The QUEUE option cannot be used for CICS-to-IMS communication.
- 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 “Traditional routing of transactions started by ATI” on page 59.

  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 61.

  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 a 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 START command. An initiating transaction can obtain its local sysid by using an ASSIGN SYSID command, or its applid by using an 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 a 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 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 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 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 15 on page 47.

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 START commands” on page 42), or if the request is being shipped to IMS.

Including start request delivery in a unit of work

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

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 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 START command (or any function shipping request) for the same system.
- The transaction issues a 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 unit of work (UOW) 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 START NOCHECK PROTECT requests to IMS. Each request must be followed by a 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 “Intersystem queuing” on page 28.

For guidance information about controlling intersystem queues, see Chapter 24, “Intersystem session queue management,” on page 267.

Local queuing of 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 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 TRANSACTION definition attributes, in 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 Intersystem communication program exits XISCONA and XISLCLQ, in 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 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 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 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.

Important:

If a started transaction issues multiple RETRIEVE commands, or uses the WAIT option of the RETRIEVE command, *allow the ROUTABLE option of the transaction definition, in the region in which the START command is issued, to default to ROUTABLE(NO)*. If the transaction is defined as ROUTABLE(YES), multiple RETRIEVE or RETRIEVE WAIT commands may not work as you expect.

For information about the ROUTABLE option of the START command, see “Routing transactions invoked by START commands” on page 68.

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 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 229). If no session is available, the request is queued until there is one.

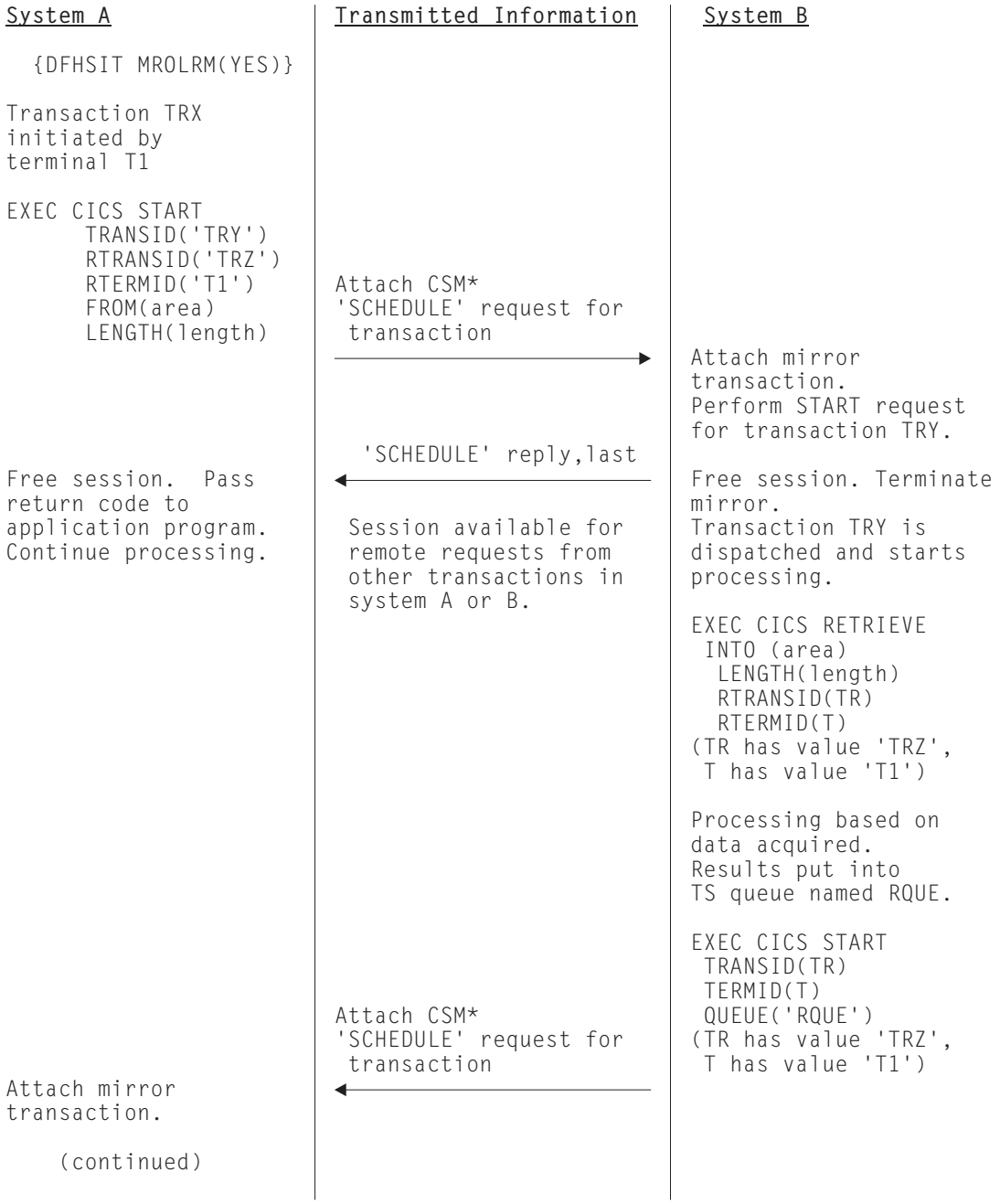
If the link to the system is an APPC link, CICS uses the modename associated with the transaction definition to select a class-of-service for the session.

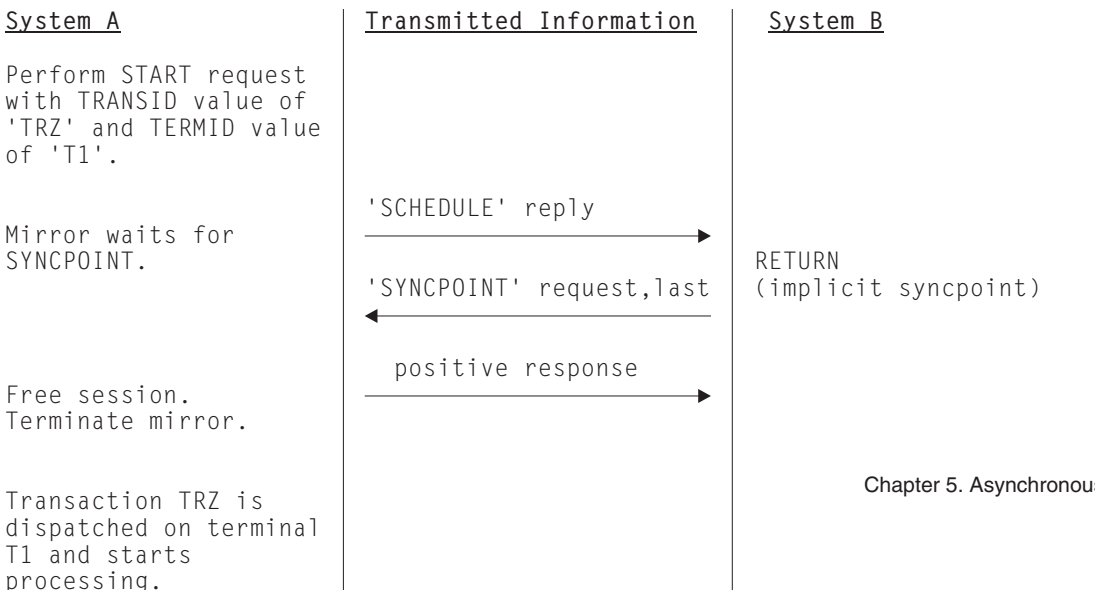
System programming considerations

This section discusses the CICS resources that must be defined for asynchronous processing.

- 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 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') FROM(area) LENGTH(length)
Attach CSM* 'SCHEDULE' request for transaction
Attach mirror transaction. Perform START request for transaction TRY.
'SCHEDULE' reply,last
Free session. Pass return code to application program. Continue processing.
Free session. Terminate mirror. Transaction TRY is dispatched and starts processing.
Session available for remote requests from other transactions in 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) QUEUE('RQUE') (TR has value 'TRZ', T has value 'T1')
Attach CSM* 'SCHEDULE' request for transaction
Attach mirror transaction.
(continued)


System A
Transmitted Information
System B
Perform START request with TRANSID value of 'TRZ' and TERMID value of 'T1'.
'SCHEDULE' reply
Mirror waits for SYNCPOINT.
RETURN (implicit syncpoint)
'SYNCPOINT' request,last
positive response
Free session. Terminate mirror.
Transaction TRZ is dispatched on terminal T1 and starts processing.

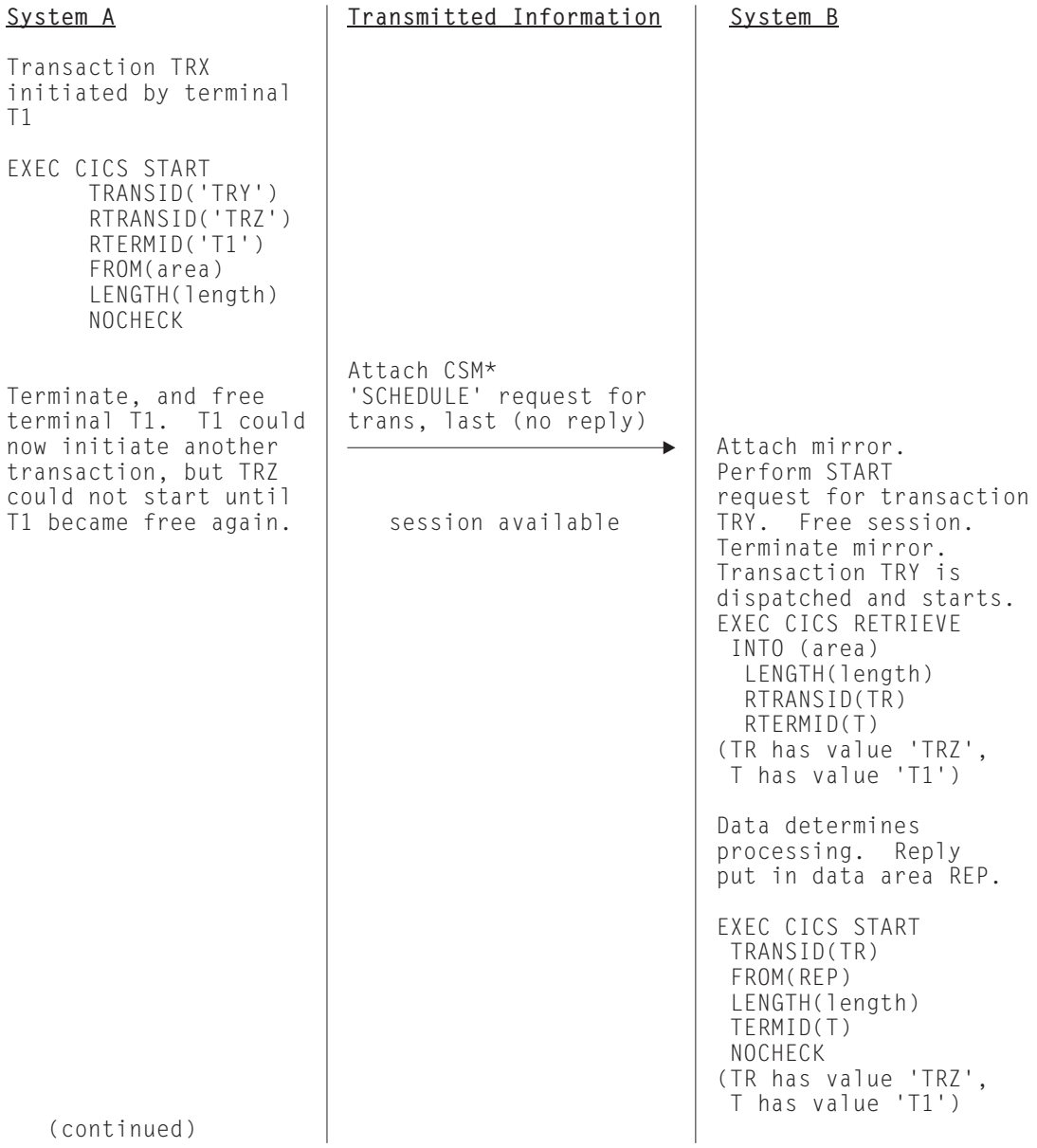


Figure 15. Asynchronous processing—remote transaction initiation using NOCHECK. This example shows an ISC connection, or an MRO connection without long-running mirrors.

Figure 15 on page 47 shows an ISC connection, or an MRO connection without long-running mirrors.

Chapter 6. Introduction to CICS dynamic routing

This chapter is an overview of the CICS dynamic routing interface. The information it contains is relevant to both Chapter 7, “CICS transaction routing,” on page 55 and Chapter 8, “CICS distributed program link,” on page 85.

What is dynamic routing?

In a CICSplex, resources (for example, transactions or programs) required by one region may be owned by another region (the resource-owning region). For example, you might have a terminal-owning region that requires access to transactions owned by an application-owning region.

Static routing
means that the location of the remote resource is specified at design time. Requests for a particular resource are always routed to the same region. Typically, when static routing is used, the location of the resource is specified in the installed resource definition.

Dynamic routing
means that the location of the remote resource is decided at run time. The decision is taken by a CICS-supplied user-replaceable **routing program**. The routing program may, at different times, route requests for a particular resource to different regions. This means, for example, that if you have several cloned application-owning regions, your routing program could balance the workload across the regions dynamically.

All the following can be dynamically routed:

- Transactions started from terminals.
- Transactions invoked by a subset of **EXEC CICS START** commands.
- CICS-to-CICS distributed program link (DPL) requests.
- Program-link requests received from outside CICS; for example, External Call Interface (ECI) calls received from CICS Clients.
- CICS business transaction services (BTS) processes and activities. BTS is described in the *CICS Business Transaction Services*.
- Method requests for enterprise beans and CORBA stateless objects. Enterprise beans are described in *Java Applications in CICS*.
- Bridge 3270 transactions.

Some further definitions are necessary:

Requesting region
The region in which a transaction or other routable request is issued. Here are some examples of what we mean by “requesting region”:

- For transactions started from terminals, it is the terminal-owning region (TOR).
- For transactions started by **EXEC CICS START** commands, it is the region in which the START command is issued.
- For “traditional” CICS-to-CICS DPL calls, it is the region in which the **EXEC CICS LINK PROGRAM** command is issued.
- For program-link calls received from outside CICS, it is the CICS region which receives the call.

- For BTS processes and activities, it is the region in which the **EXEC CICS RUN ACTIVITY ASYNCHRONOUS** command is issued.
- For method requests on enterprise beans or CORBA stateless objects:
  - If the method call is issued outside CICS; for example, by a remote (non-CICS) IIOP client. The requesting region is the listener region which receives the call.
  - If the method call is issued inside CICS; or example, by an enterprise bean object that calls a method of another enterprise bean. The requesting region is the region on which the call is issued.

Routing region
The region in which the routing program is invoked for route selection. With two exceptions, the requesting region and the routing region are always the same region. The exceptions are:

1. Some terminal-related START commands:
   - Because a terminal-related START command always runs in the terminal-owning region, the requesting region and the routing region might not be the same. This is fully explained in “Routing transactions invoked by START commands” on page 68.)
   - The routing region is always the TOR.
2. Some method requests for enterprise beans or CORBA stateless objects issued inside CICS:
   - An enterprise bean, program, or object on the local EJB/CORBA server calls a method of an object on a remote EJB/CORBA server. The requesting region is the local region on which the method call is issued. The routing region is the listener region on the remote EJB/CORBA server.

Target region
The region in which the routed transaction or request executes.

Two routing models

There are two possible dynamic routing models.

The “hub” model

The “hub” is the model that has traditionally been used with CICS dynamic transaction routing. A routing program running in a TOR routes transactions between several AORs. Usually, the AORs (unless they are AOR/TORs) do no dynamic routing. Figure 16 on page 51 shows a “hub” routing model.

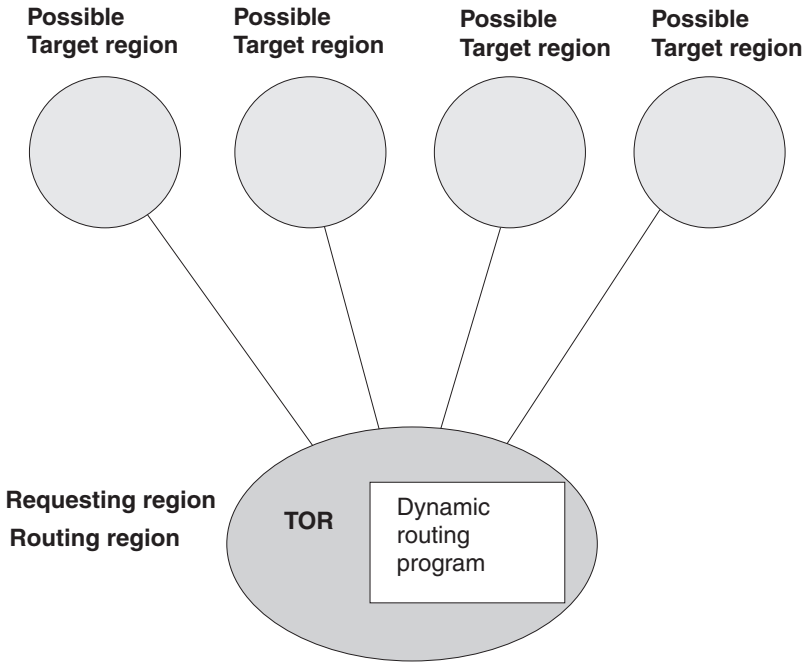


Figure 16. Dynamic routing using a "hub" routing model. One routing region (the TOR) selects between several target regions.

The "hub" model applies to the routing of:

- Transactions started from terminals.
- Transactions started by terminal-related START commands.
- Program-link requests received from outside CICS. (The receiving region acts as a "hub" or "TOR" because it routes the requests among a set of back-end server regions.)
- Bridge 3270 requests.

The "hub" model is a *hierarchical* system—routing is controlled by one region (the TOR); normally a routing program runs only in the TOR.

Advantage of the "hub" model

It is a relatively simple model to implement. For example, compared to the distributed model, there are few inter-region connections to maintain.

Disadvantages of the "hub" model

- If you use only one "hub" to route transactions and program-link requests across your AORs, the "hub" TOR is a single point-of-failure.
- If you use more than one "hub" to route transactions and program-link requests across the same set of AORs, you may have problems with distributed data. For example, if the routing program keeps a count of routed transactions for load-balancing purposes, each "hub"-TOR will need access to this data.

The distributed model

In the distributed model, each region may be both a routing region and a target region. A routing program runs in each region. Figure 17 on page 52 shows a distributed routing model.

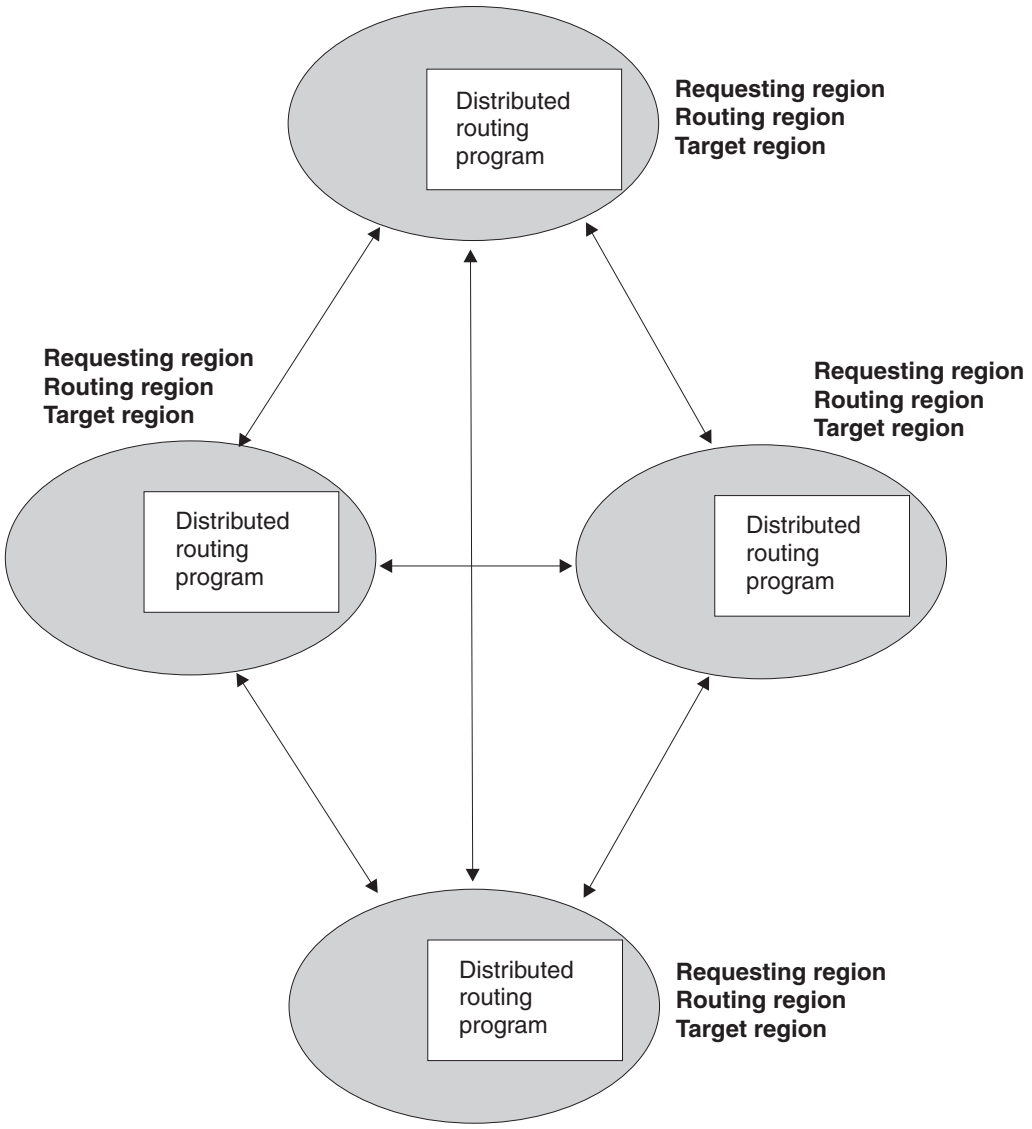


Figure 17. Dynamic routing using a distributed routing model. Each region may be both a routing region and a target region.

The distributed model applies to the routing of:

- CICS business transaction services processes and activities
- Method requests for enterprise beans and CORBA stateless objects
- Non-terminal-related START requests
- CICS-to-CICS DPL requests

The distributed model is a *peer-to-peer* system—each participating CICS region may be both a routing region and a target region. A routing program runs in each region.

Advantage of the distributed model

There is no single point-of-failure.

Disadvantages of the distributed model

- Compared to the “hub” model, there are a great many inter-region connections to maintain.

- You may have problems with distributed data. For example, any data used to make routing decisions must be available to all the regions. (CICSPlex SM solves this problem by using dataspaces.)

Two routing programs

There are two CICS-supplied user-replaceable programs for dynamic routing:

The dynamic routing program, DFHDYP
Can be used to route:
- Transactions started from terminals
- Transactions started by terminal-related START commands
- CICS-to-CICS DPL requests
- Program-link requests received from outside CICS
- Bridge 3270 requests

The distributed routing program, DFHDSRP
Can be used to route:
- CICS business transaction services processes and activities
- Method requests for enterprise beans and CORBA stateless objects
- Non-terminal-related START requests.

The two routing programs:

1. Are specified on separate system initialization parameters. You specify the name of your dynamic routing program on the DTRPGM system initialization parameter. You specify the name of your distributed routing program on the DSRTPGM system initialization parameter.
2. Are passed the same communications area. (Certain fields that are meaningful to one program are not meaningful to the other.)
3. Are invoked at similar points—for example, for route selection, route selection error, and (optionally) at termination of the routed transaction or program-link request.

Together, these three factors give you a great deal of flexibility. You could, for example, do any of the following:

- Use different user-written programs for dynamic routing and distributed routing.
- Use the same user-written program for both dynamic routing and distributed routing.
- Use a user-written program for dynamic routing and the CICSPlex SM routing program for distributed routing, or vice versa.

It is worth noting two important differences between the dynamic and distributed routing programs:

1. The dynamic routing program is only invoked if the resource (the transaction or program) is defined as DYNAMIC(YES). The distributed routing program, on the other hand, is invoked (for eligible non-terminal-related START requests, BTS activities, and method requests for enterprise beans and CORBA stateless objects) even if the associated transaction is defined as DYNAMIC(NO)—though it cannot route the request. What this means is that the distributed routing program is better able to monitor the effect of statically-routed requests on the relative workloads of the target regions.
2. Because the dynamic routing program uses the hierarchical “hub” routing model—one routing program controls access to resources on several target

regions—*the routing program that is invoked at termination of a routed request is the same program that was invoked for route selection*.

The distributed routing program, on the other hand, uses the distributed model, which is a peer-to-peer system; the routing program itself is distributed. *The routing program that is invoked at initiation or termination of a routed transaction is not the same program that was invoked for route selection*—it is the routing program on the target region.

Important:

If you intend to route from CICS Transaction Server for z/OS, Version 3 Release 2 to a CICS Transaction Server for OS/390, Version 1 Release 3 region (or vice versa), you must ensure that the PTF for CICS APAR PQ 75814 is applied to CICS Transaction Server for OS/390, Version 1 Release 3.

If you use CICSPlex SM for routing, the PTFs for each of the following CICSPlex SM APARs must be applied to each relevant CICSPlex SM release:

CICSPlex SM Version 1 Release 4
PQ80891

CICSPlex SM Version 2 Release 2
PQ80893

CICSPlex SM Version 2 Release 3
PQ81235

Chapter 7. 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 18 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**.

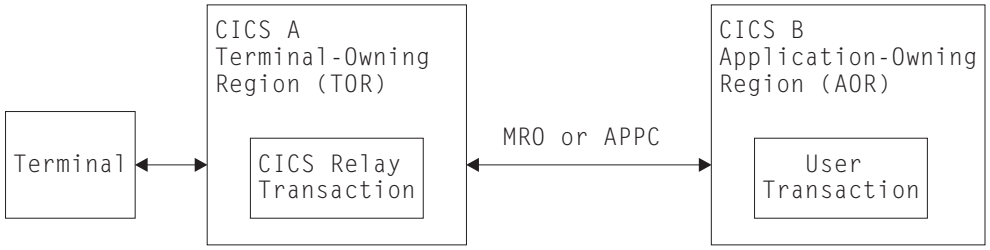


Figure 18. The elements of transaction routing

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

- EXCI connections and sessions
- IBM 7770 or 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- MVS 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 80.)

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 22, “Application programming for CICS transaction routing,” on page 241.

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. The two methods of routing transactions started by ATI are described in:
   - “Traditional routing of transactions started by ATI” on page 59
   - “Routing transactions invoked by START commands” on page 68.
3. A transaction can issue an 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 “Using the routing transaction (CRTE)” on page 81.

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 209.

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

Dynamic routing models:

Dynamic routing of terminal-initiated transactions uses the "hub" routing model described in "The "hub" model" on page 50.

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 routing program**, to intercept the terminal input data and specify that it be redirected to any transaction and system. The default dynamic 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 Writing a dynamic routing program, in the *CICS Customization Guide*. For information about system initialization parameters, see Specifying CICS system initialization parameters, in the *CICS System Definition Guide*.

When your routing program is invoked

CICS invokes the dynamic routing program:

- When a transaction defined as DYNAMIC(YES) is initiated.

  Note:

  1. If a transaction definition is not found, CICS uses the common transaction definition specified on the DTRTRAN system initialization parameter. See "Using a single transaction definition in the TOR" on page 213.
  2. If the transaction is defined as DYNAMIC(YES) in the target region, as well as in the routing region (TOR), the dynamic routing program is invoked, for routing, in the target region, as well as in the TOR. Thus, it is possible to "daisy-chain" routed requests from one region to another. Take care that this does not occur unintentionally.

  If the transaction was initiated from a terminal, the dynamic routing program can route the request.

  If the transaction was initiated by an EXEC CICS START command, the routing program may or may not be able to route the request—see "Routing transactions invoked by START commands" on page 68.
- 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.
- For routing of DPL requests, at all the points described in "Dynamically routing DPL requests" on page 88.

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 213.
- 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 dynamic 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. 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 24, "Intersystem session queue management," on page 267.)
- 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 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 Writing a dynamic routing program, in the *CICS Customization Guide*.

The CICS Interdependency Analyzer

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 are dependent on each other (that is, that contain *affinities*) so that it can route them consistently.

If you are planning to create a dynamic transaction routing environment, consisting perhaps of a mixture of CICS Transaction Server for z/OS, Version 3 Release 2 and earlier systems, you may find the CICS Interdependency Analyzer useful. It can be used to identify the causes of inter-transaction affinities in CICS Transaction Server for z/OS and CICS Transaction Server for OS/390 regions.

For more information about this utility, see the *CICS Interdependency Analyzer for z/OS User's Guide and Reference*.

For further information about transaction affinities, see Affinity , in 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. The output from the CICS Interdependency Analyzer can be used directly by CICSPlex SM.

Using CICSPlex SM, you could integrate workload balancing for transactions and DPL requests.

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

Traditional routing of transactions started by ATI

This section describes the “traditional” method of routing transactions that are started by automatic transaction initiation (ATI).

Important:

Wherever possible, you should use the enhanced method described in “Routing transactions invoked by START commands” on page 68. However, you cannot use the enhanced method to route:

- Transactions invoked by the trigger-level on a transient data queue
- Some transactions that are invoked by EXEC CICS START commands.

For these cases, you must use the “traditional” method described in this section.

Automatic transaction initiation is the process whereby a transaction request made internally within a CICS system or systems network leads to the scheduling of the transaction. ATI requests result from:

EXEC CICS START commands

A START command causes CICS interval control to initiate a transaction after a specified period of time (which may be zero) has elapsed.

Transient data queues

A transient data queue can be defined so that a transaction is automatically initiated when the number of records on the queue reaches a specified level.

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 19 on page 61, an application in AOR1 issues a 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 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. 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. We are talking here of “traditional” routing of transactions started by START commands. To find out how to use the ROUTABLE option of the transaction definition to specify enhanced routing, see “Routing transactions invoked by START commands” on page 68.

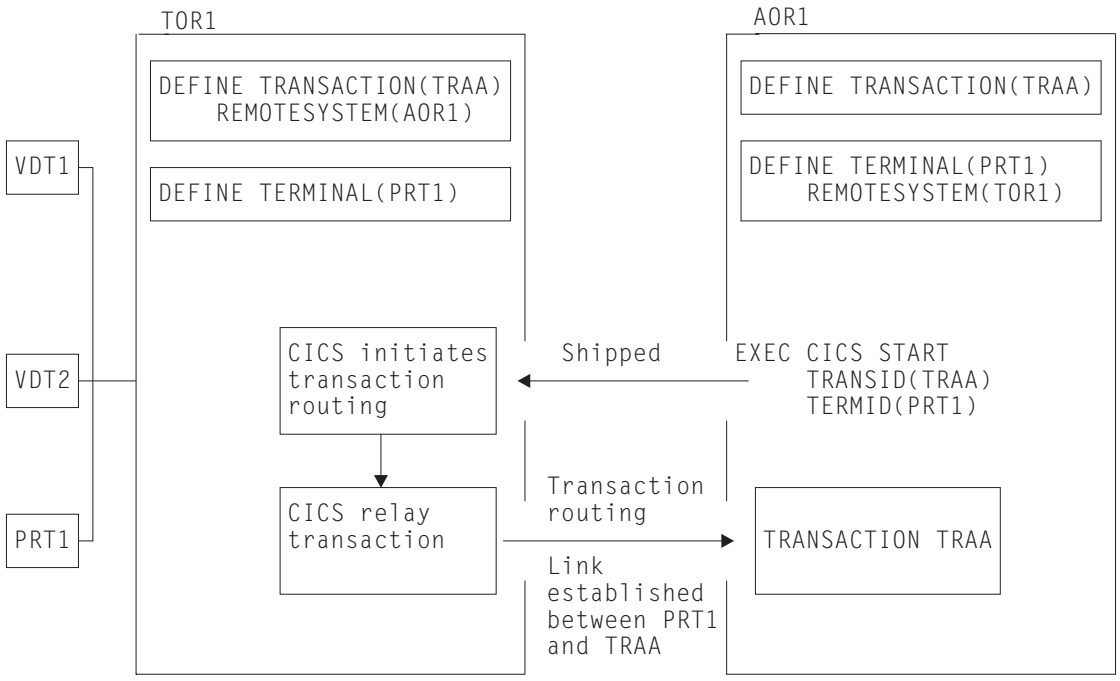


Figure 19. ATI-initiated transaction routing

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 several ways. For example:

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

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

Terminal-not-known condition

Important:

> The “terminal-not-known condition” frequently occurs, as the example in this section explains, because a terminal-related START command is issued in the terminal-owning region and function-shipped to the application-owning region, where the terminal is not yet defined. If you are able to use the enhanced routing method described in “Routing transactions invoked by START commands” on page 68, a START command issued in a TOR is not function-shipped to the AOR; thus the “terminal-not-known” condition does not occur.

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 Autoinstall, in 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 202.) 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 20 on page 63 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. Because X1 is defined as a remote transaction owned by CICB, CICA function-ships the 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.

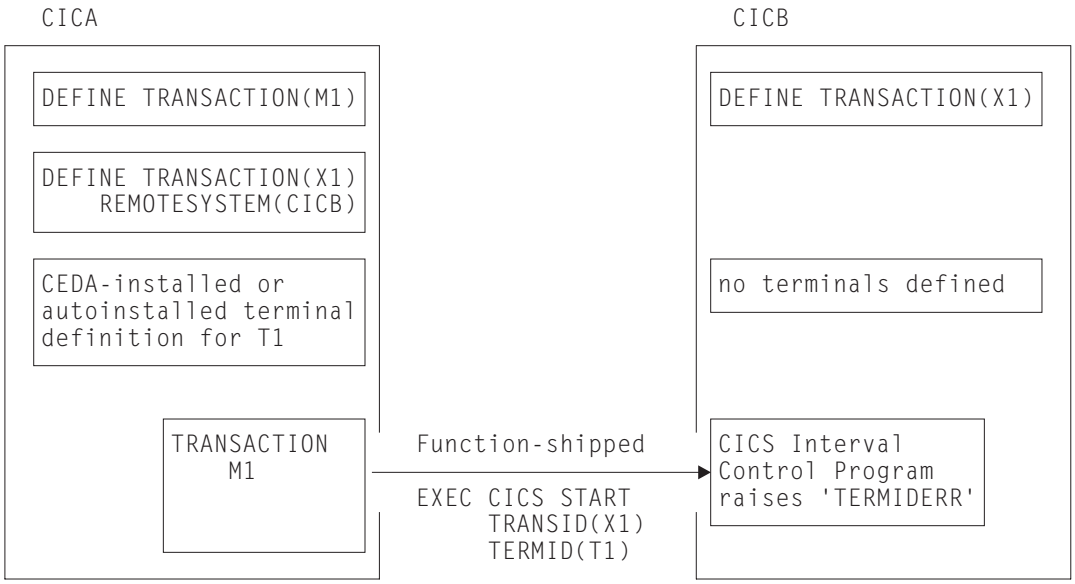


Figure 20. Failure of an ATI request in a system where the termid is unknown

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 a 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 a START command and from the transient data queue trigger mechanism. This means that a START command could result in an invocation of both exits.

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

- Whether the ATI request resulted from: a 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 20 on page 63. 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 21 on page 65:

1. The interval control program accepts the 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.

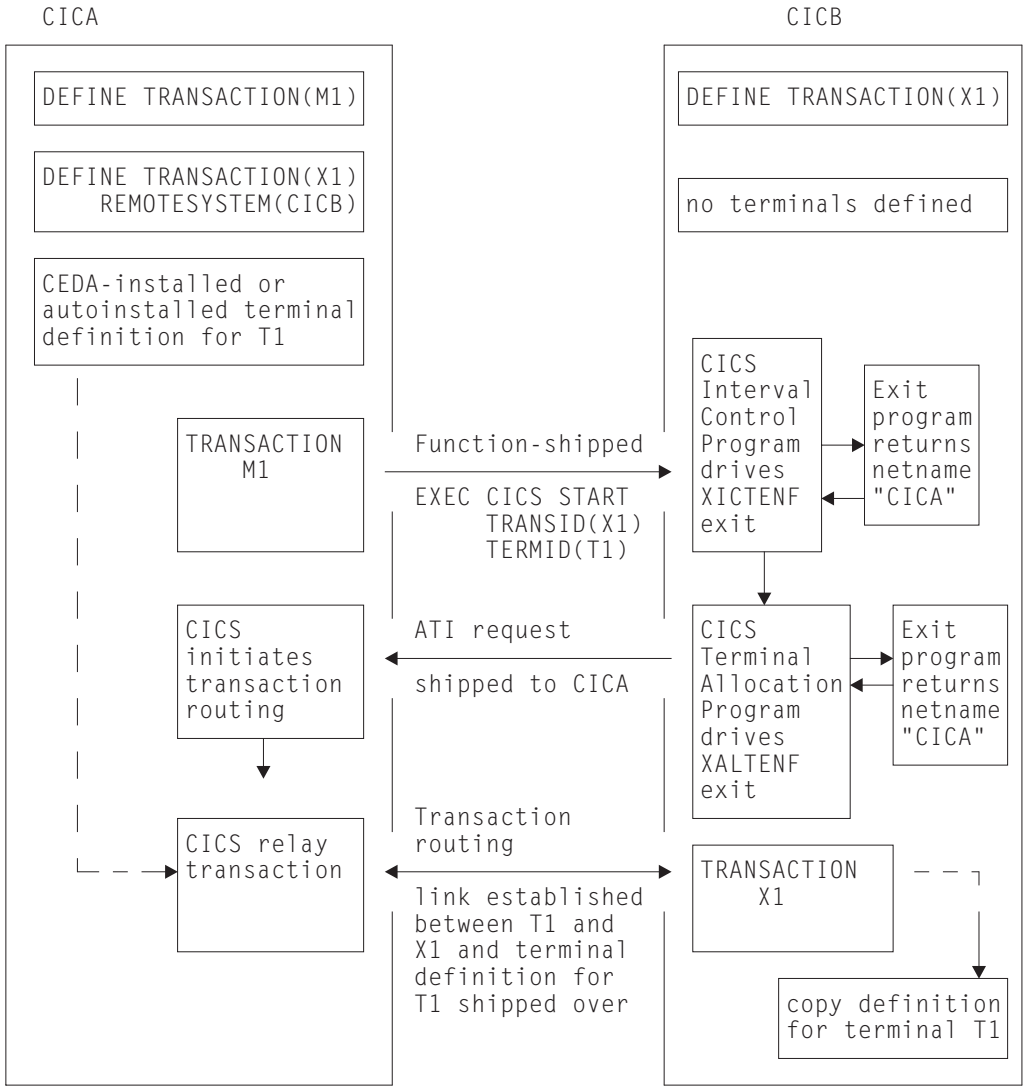


Figure 21. Resolving a 'terminal not known' condition on a START request

The example in Figure 21 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 Writing initialization and shutdown programs, in 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 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 'Terminal not known' condition exits XALTENF and XICTENF, in the *CICS Customization Guide*. A sample program is also available in the DFHXTENF member of library CICSTS32.CICS.SDFHSAMP.

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 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 on region TORA 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, rather than TORB, for transaction routing from terminal T1.

Figure 22 on page 67 illustrates what happens.

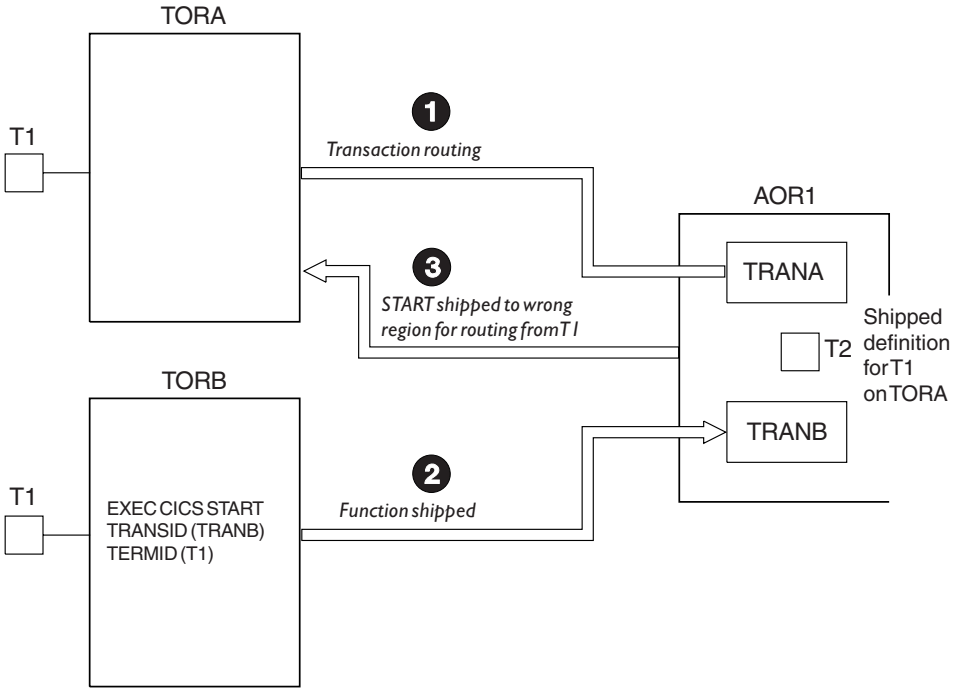


Figure 22. Function-shipped START request started against an incorrect terminal. Because a shipped definition of terminal T1 (owned by TORA) is installed on AOR1, the START request received from TORB is shipped to TORA, for routing, rather than to TORB.

There are two ways to prevent this situation:

1. **This is the preferred method**.

   Use the enhanced routing method described in "Routing transactions invoked by START commands" on page 68. A terminal-related START command issued in the terminal-owning region is *not* function-shipped to the AOR; thus it cannot be shipped back to the wrong TOR. Instead, the START executes directly in the TOR, and the transaction is routed as if it had been initiated from a terminal.

   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 in "Terminal aliases" on page 208. For information about writing an autoinstall program to control the installation of shipped definitions, see the *CICS Customization Guide*.

2. Use this method if you cannot use the enhanced routing method.

   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.

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

ATI and generic resources

An AOR can issue an EXEC CICS START request against a terminal that is owned by a VTAM generic resource, without knowing the member of the generic resource group to which the terminal is currently logged on. For details of using ATI with generic resources, see “Using ATI with generic resources” on page 134.

Routing transactions invoked by START commands

This section describes the preferred method of routing transactions that are invoked by EXEC CICS START commands. For convenience, we shall call the method described in this section the *enhanced* method. The enhanced method supersedes the “traditional” method described in “Traditional routing of transactions started by ATI” on page 59. Note, however, that the enhanced method cannot be used to route:

- Some transactions that are invoked by EXEC CICS START commands
- Transactions invoked by the trigger-level on a transient data queue.

In these cases, the “traditional” method must be used.

To specify that a transaction, if it is invoked by an EXEC CICS START command, is to be routed by the enhanced method described in this section, *define the transaction as ROUTABLE(YES) in the requesting region* (the region in which the START command is issued).

Advantages of the enhanced method

There are several advantages in using the enhanced method, where possible, rather than the “traditional” method:

Dynamic routing

Using the “traditional” method, you cannot route the started transaction dynamically. (For example, if the transaction on a terminal-related START command is defined as DYNAMIC(YES) in the terminal-owning region, your dynamic routing program is invoked for notification only—it cannot route the transaction.)

Using the enhanced method, you can route the started transaction dynamically.

Efficiency

Using the “traditional” method, a terminal-related START command issued in a TOR is function-shipped to the AOR that owns the transaction. The request is then shipped back again, for routing from the TOR.

Using the enhanced method, the two hops to the AOR and back are missed out. A START command issued in a TOR executes directly in the TOR, and the transaction is routed without delay.

Simplicity

Using the “traditional” method, when a terminal-related START command issued in a TOR is function-shipped to the AOR that owns the transaction the “terminal-not-known” condition may occur if the terminal is not defined in the AOR.

Using the enhanced method, because a START command issued in a TOR is not function-shipped to the AOR, the “terminal-not-known” condition does not

occur. The START command executes in the TOR directly, and the transaction is routed just as if it had been initiated from a terminal. If the terminal is not defined in the AOR, a definition is shipped from the TOR.

Terminal-related START commands

For a transaction invoked by a terminal-related START command to be eligible for the enhanced routing method, *all* of the following conditions must be met:

- The START command must be a member of the subset of eligible START commands—that is, it must meet all the following conditions:
  - The START command specifies the TERMID option, which names the terminal associated with the current task.
  - The principal facility of the task that issues the START command is a terminal. This is *not* the case if, for example, the program that issues the START command was linked-to by DPL; in this case, the principal facility is the intersystem session.
  - The principal facility of the task that issues the START command is *not* a surrogate Client virtual terminal.
  - The SYSID option of the START command does not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)
- The requesting region, the TOR, and the target region must all be CICS Transaction Server for OS/390, Version 1 Release 3 or later.

  Note: The requesting region and the TOR may be the same region.
- The requesting region and the TOR (if they are different) must be connected by either of the following:
  - An MRO link
  - An APPC parallel-session link.
- The TOR and the target region must be connected by either of the following:
  - An MRO link.
  - An APPC single- or parallel-session link. If an APPC link is used, at least one of the following must be true:
    1. Terminal-initiated transaction routing has previously taken place over the link. (The terminal-initiated transaction routing enables the TOR to determine whether or not the target region is a CICS Transaction Server for OS/390, Version 1 Release 3 or later system, and therefore eligible for enhanced routing.)
    2. CICSPlex SM is being used for routing.
- The transaction definition in the requesting region must specify ROUTABLE(YES).
- The transaction definition for the transaction to be started does not specify the REMOTESYSTEM option. (The remote region on which the transaction is to be started must not be specified explicitly.)
- If the requesting region and the TOR are different, the transaction definition in the requesting region must not specify the REMOTESYSTEM option. If the requesting region and the TOR are the same region, you may use REMOTESYSTEM in the transaction definition for static routing.
- If the transaction is to be routed dynamically, the transaction definition in the TOR must specify DYNAMIC(YES).

Important: When considering which START-initiated transactions are candidates for dynamic routing, you need to take particular care if the START command specifies any of the following options:

- AT, AFTER, INTERVAL, or TIME. (That is, there is a delay before the START is executed.)
- QUEUE.
- REQID.
- RTERMID.
- RTRANID.

You need to understand how each of the options of the START command is being used; whether, for example, it affects the set of regions to which the transaction can be routed.

START commands issued in an AOR

If a terminal-related START command is issued in an AOR, it is shipped to the TOR that owns the terminal named in the TERMID option. The START executes in the TOR.

Static routing:

The transaction definition in the AOR specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(NO). The dynamic routing program is not invoked.

If the transaction is eligible for enhanced routing, it is routed to the AOR named in the REMOTESYSYEM option of the transaction definition in the TOR. If REMOTESYSTEM is not specified, the transaction executes locally, in the TOR.

Note: If the transaction is ineligible for enhanced routing, it is handled in the "traditional" way described in "Traditional routing of transactions started by ATI" on page 59 - that is, CICS tries to route it back to the originating AOR for execution. If the REMOTESYSTEM option of the transaction definition in the TOR names a region other than the originating AOR, the request fails.

Figure 23 on page 71 shows the requirements for using the enhanced method to statically route a transaction that is initiated by a terminal-related START command issued in an AOR.

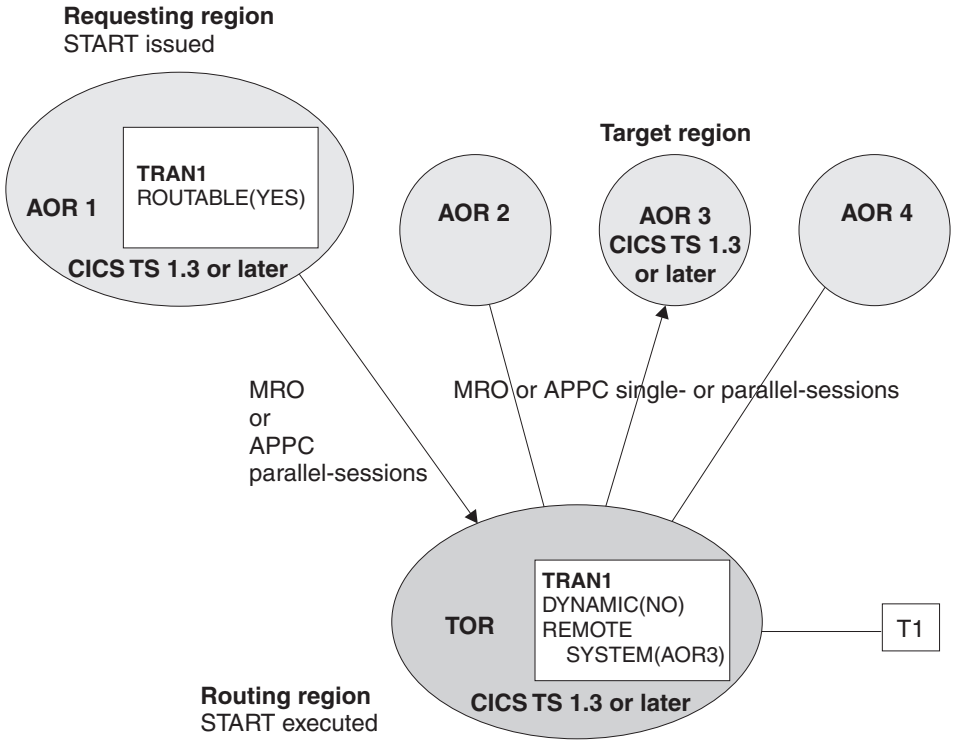


Figure 23. Static routing of a terminal-related START command issued in an AOR, using the enhanced method. The requesting region, the TOR, and the target region are all CICS TS OS/390, Version 1.3 or later. The requesting region and the TOR are connected by an MRO or APPC parallel-session link. The TOR and the target region are connected by an MRO or APPC (single- or parallel-session) link. The transaction definition in the requesting region specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(NO). The REMOTESYSTEM option names the AOR to which the transaction is to be routed.

Dynamic routing:

Dynamic routing models:

Dynamic routing of transactions invoked by terminal-related START commands uses the “hub” routing model described in “The “hub” model” on page 50.

The transaction definition in the AOR specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(YES). The dynamic routing program is invoked in the TOR. If the transaction is eligible for enhanced routing, the routing program can reroute the transaction to an alternative AOR—that is, to an AOR other than that in which the START was issued.

Note: If the transaction is ineligible for enhanced routing, the dynamic routing program is invoked for notification only—it cannot reroute the transaction. The transaction is handled in the “traditional” way—that is, it is routed back to the originating AOR for execution.

Figure 24 on page 72 shows the requirements for dynamically routing a transaction that is initiated by a terminal-related START command issued in an AOR.

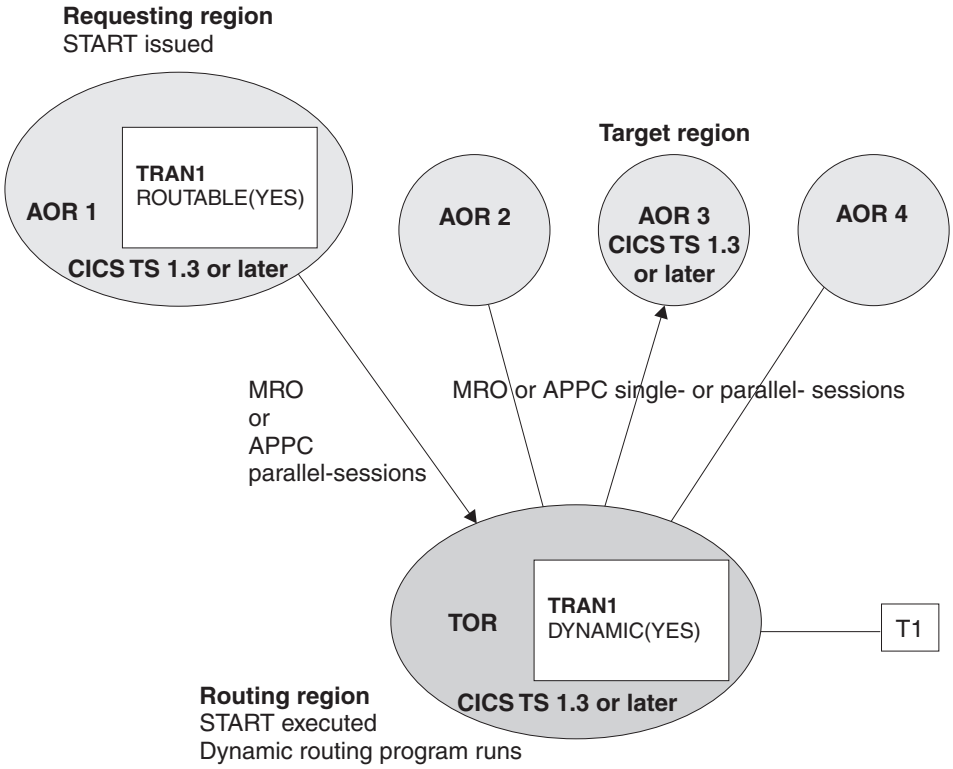


Figure 24. Dynamic routing of a terminal-related START command issued in an AOR. The requesting region, the TOR, and the target region are all CICS Transaction Server for OS/390, Version 1 Release 3 or later. The requesting region and the TOR are connected by an MRO or APPC parallel-session link. The TOR and the target region are connected by an MRO or APPC (single- or parallel-session) link. The transaction definition in the requesting region specifies ROUTABLE(YES). The transaction definition in the TOR specifies DYNAMIC(YES).

START commands issued in a TOR

Static routing: The transaction definition in the TOR specifies ROUTABLE(YES) and DYNAMIC(NO). The dynamic routing program is not invoked. If the transaction is eligible for enhanced routing (see the list of conditions for START commands to be eligible for enhanced routing):

1. The START executes in the TOR.
2. The transaction is routed to the AOR named in the REMOTESYSYEM option of the transaction definition. If REMOTESYSTEM is not specified, the transaction executes locally, in the TOR.

Note: If the transaction is ineligible for enhanced routing, the START request is handled in the "traditional" way described in "Traditional routing of transactions started by ATI" on page 59—that is, it function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition. If REMOTESYSTEM is not specified, the START executes locally, in the TOR.

Figure 25 on page 73 shows the requirements for using the enhanced method to statically route a transaction that is initiated by a terminal-related START command issued in a TOR.

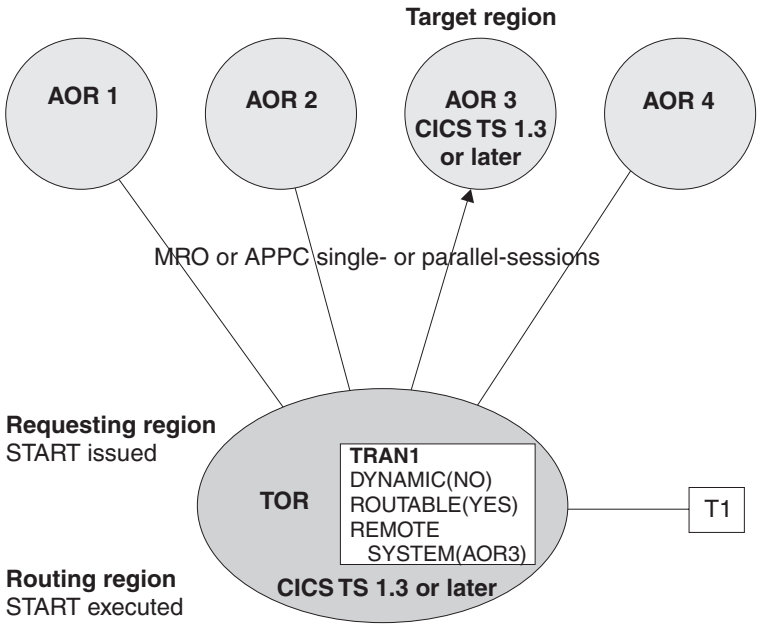


Figure 25. Static routing of a terminal-related START command issued in a TOR, using the enhanced method. The TOR and the target region are both CICS Transaction Server for OS/390, Version 1 Release 3 or later. The TOR and the target region are connected by an MRO or APPC (single- or parallel-session) link. The transaction definition in the TOR specifies DYNAMIC(NO) and ROUTABLE(YES). The REMOTESYSTEM option names the AOR to which the transaction is to be routed.

Dynamic routing:

Dynamic routing models:

Dynamic routing of transactions invoked by terminal-related START commands uses the "hub" routing model described in "The "hub" model" on page 50.

The transaction definition in the TOR specifies ROUTABLE(YES) and DYNAMIC(YES). The dynamic routing program is invoked. If the transaction is eligible for enhanced routing, the START is executed in the TOR, and the routing program can route the transaction.

Note: If the transaction is ineligible for enhanced routing, the dynamic routing program is invoked for notification only—it cannot route the transaction. The START request is handled in the "traditional" way—that is, it is function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition in the TOR. If REMOTESYSTEM is not specified, the START executes locally, in the TOR.

Figure 26 on page 74 shows the requirements for dynamically routing a transaction that is initiated by a terminal-related START command issued in a TOR.

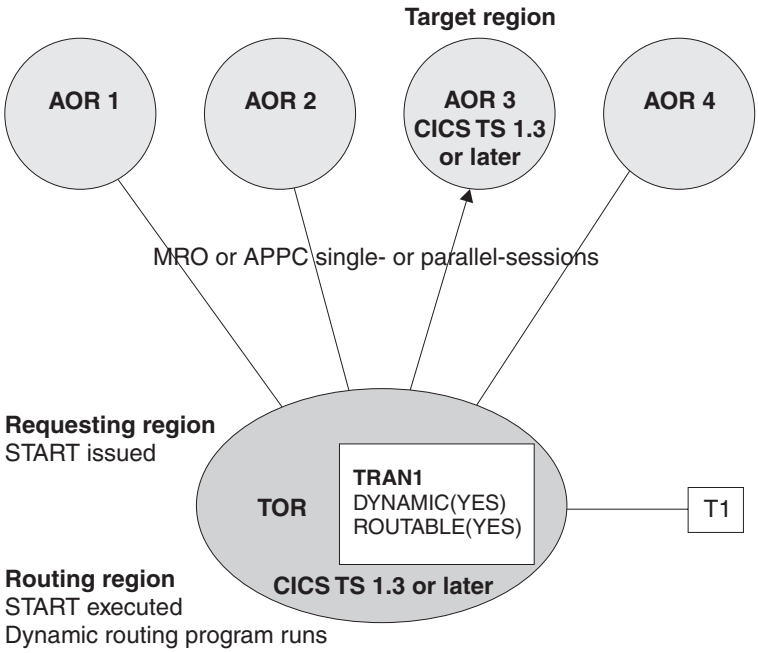


Figure 26. Dynamic routing of a terminal-related START command issued in a TOR. The TOR and the target region are both CICS Transaction Server for OS/390, Version 1 Release 3 or later. The TOR and the target region are connected by an MRO or APPC (single- or parallel-session) link. The transaction definition in the TOR specifies both DYNAMIC(YES) and ROUTABLE(YES).

Non-terminal-related START commands

For a non-terminal-related START request to be eligible for enhanced routing, *all* of the following conditions must be met:

- The requesting region is CICS Transaction Server for OS/390, Version 1 Release 3 or later.

  Note: In order for the distributed routing program to be invoked on the *target* region as well as on the requesting region, the target region too must be CICS Transaction Server for OS/390, Version 1 Release 3 or later. (For information about the points at which the distributed routing program is invoked, see Writing a distributed routing program, in the *CICS Customization Guide*.)
- The requesting region and the target region are connected by either of the following:
  - An MRO link.
  - An APPC single- or parallel-session link. If an APPC link is used, and the distributed routing program is to be invoked on the target region, at least one of the following must be true:
    1. Terminal-initiated transaction routing has previously taken place over the link. (The terminal-initiated transaction routing enables the requesting region to determine whether or not the target region is a CICS Transaction Server for OS/390, Version 1 Release 3 or later system.)
    2. CICSPlex SM is being used for routing.
- The transaction definition in the requesting region specifies ROUTABLE(YES).

In addition, if the request is to be routed dynamically:

- The transaction definition in the requesting region must specify DYNAMIC(YES).
- The SYSID option of the START command must not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)

Important: When considering which START-initiated requests are candidates for dynamic routing, you need to take particular care if the START specifies any of the following options:

- AT, AFTER, INTERVAL(non-zero), or TIME. That is, there is a delay before the START is executed.

  If there is a delay, the interval control element (ICE) created by the START request is kept in the requesting region with a transaction ID of CDFS. The CDFS transaction retrieves any data specified by the user and reissues the START request without an interval. The request is routed when the ICE expires, based on the state of the transaction definition and the sysplex at that moment.
- QUEUE.
- REQID.
- RTERMID.
- RTRANID.

You need to understand how these options are being used; whether, for example, they affect the set of regions to which the request can be routed.

Static routing

The transaction definition in the requesting region specifies ROUTABLE(YES) and DYNAMIC(NO). If the START request is eligible for enhanced routing (see above), the distributed routing program—that is, the program specified on the DSRTPGM system initialization parameter—is invoked for notification of the statically-routed request.

Note:

1. The distributed routing program differs from the dynamic routing program, in that it is invoked—for eligible non-terminal-related START requests where the transaction is defined as ROUTABLE(YES)—even when the transaction is defined as DYNAMIC(NO). The dynamic routing program is never invoked for transactions defined as DYNAMIC(NO). This difference in design means that you can use the distributed routing program to assess the effect of statically-routed requests on the overall workload.
2. If the request is ineligible for enhanced routing, the distributed routing program is not invoked.

Dynamic routing

Dynamic routing models:

Dynamic routing of non-terminal-related START requests uses the distributed routing model described in “The distributed model” on page 51.

The transaction definition in the requesting region specifies ROUTABLE(YES) and DYNAMIC(YES). If the request is eligible for enhanced routing, the distributed

routing program is invoked for routing. The START request is function-shipped to the target region returned by the routing program.

Note:

1. If the request is ineligible for enhanced routing, the distributed routing program is not invoked. Unless the SYSID option specifies a remote region explicitly, the START request is function-shipped to the AOR named in the REMOTESYSTEM option of the transaction definition in the requesting region; if REMOTESYSTEM is not specified, the START executes locally, in the requesting region.
2. If the request is eligible for enhanced routing, but the SYSID option of the START command names a remote region, the distributed routing program is invoked for notification only—it cannot route the request. The START executes on the remote region named on the SYSID option.
3. If you intend to route from CICS Transaction Server for z/OS, Version 3 Release 2 to a CICS Transaction Server for OS/390, Version 1 Release 3 region (or vice versa), you must ensure that the PTF for CICS APAR PQ 75814 is applied to CICS Transaction Server for OS/390, Version 1 Release 3.

   If you use CICSPlex SM for routing, the PTFs for each of the following CICSPlex SM APARs must be applied to each relevant CICSPlex SM release:

   CICSPlex SM Version 1 Release 4
   PQ80891
   CICSPlex SM Version 2 Release 2
   PQ80893
   CICSPlex SM Version 2 Release 3
   PQ81235

Canceling interval control requests:

To cancel a previously-issued START, DELAY, or POST interval control request, you use the CANCEL command. The REQID option specifies the identifier of the request to be canceled. If the request is due to execute on a remote region, you can use the SYSID option to specify that the CANCEL command is to be shipped to that region.

START and DELAY requests can be canceled only before any interval specified on the request has expired. If a START request is dynamically routed, it is kept in the local region until the interval expires, and can therefore be canceled by a locally-issued CANCEL command on which the SYSID option is unnecessary. However, in a distributed routing environment (in which each region can be both a requesting region and a target region), there may be times when you have no way of knowing to which region to direct a CANCEL command. For example, you might want to cancel a DELAY request which could have been issued on any one of a set of possible regions. To resolve a situation like this:

1. Issue a CANCEL command on which the REQID option specifies the identifier of the request to be canceled, and the SYSID option is not specified. The command executes locally.
2. Use an XICEREQ global user exit program based on the CICS-supplied sample program, DFH$ICCN. Your exit program is invoked before the CANCEL command is executed. DFH$ICCN:
   a. Checks:
      1) That it has been invoked for a CANCEL command.

2) That the SYSID option was not specified on the command.
3) That the identifier of the request to be canceled does not begin with 'DF'. ('DF' indicates a request issued internally by CICS.)
4) That the name of the transaction that issued the CANCEL command does not begin with 'C'—that is, that the transaction is not a CICS internal transaction, nor a CICS-supplied transaction such as CECI.

If one or more of these conditions are not met—for example, if it was invoked for a RETRIEVE command—DFH$ICCN does nothing and returns.

b. Instructs CICSPlex SM to:
1) Search every CICS region that it knows about for an interval control request with the identifier (REQID) specified on the CANCEL command.
2) On each region, cancel the first request (with the specified identifier) that it finds. Note that:
- Requests may be canceled on more than one region.
- If a particular region contains more than one request with the specified identifier, only the first request found by CICSPlex SM is canceled.
- You must ensure that CICSPlex SM has UPDATE access to the transaction ID of the transaction associated with the CANCEL request.

Note: For full details of DFH$ICCN's processing, see the comments in the sample program.

For details of the CANCEL command, see CANCEL, in the *CICS Application Programming Reference*. For general information about how to write an XICEREQ global user exit program, see Interval control EXEC interface program exits, in the *CICS Customization Guide*.

Allocation of remote APPC connections

A transaction running in the application-owning region can issue an 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 95.

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 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 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 27 on page 79 shows a network of three CICS systems chained together, of which the first is linked to an APPC terminal.

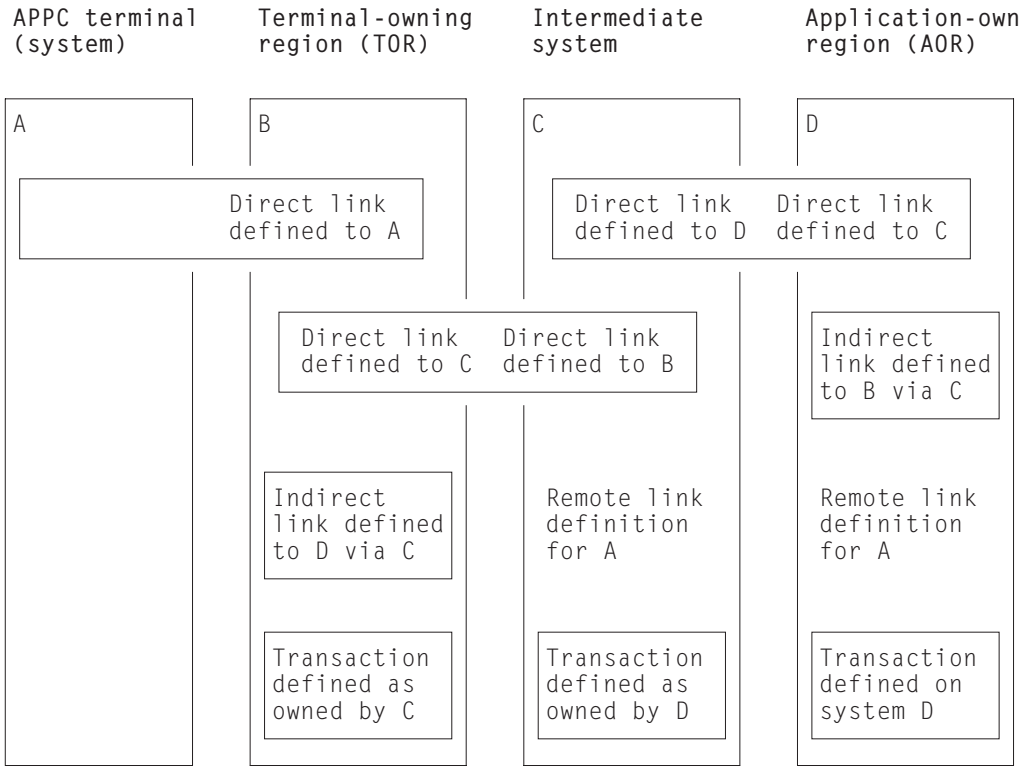


Figure 27. Transaction routing to an APPC terminal across daisy-chained systems

Note:

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 “Defining indirect links for transaction routing” on page 170.
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 BMS ROUTE command to route messages to remote terminals. For programming information about the BMS ROUTE command, see ROUTE, in the *CICS Application Programming Reference*. You cannot, however, route a message to a selected remote operator or operator class unless you also specify the terminal at which the message is to be delivered.

Table 2 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 ROUTE command (or a shipped terminal definition must already be available; see “Shipping terminal and connection definitions” on page 202). Note that the facility described in “Shipping terminals for automatic transaction initiation” on page 61 does not apply to terminals addressed by the ROUTE command.

Table 2. BMS message routing to remote terminals and operators

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

Using 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 RACF® 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 217.) 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 for transaction routing

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

1. Install MRO or ISC support, or both.
2. Define MRO or ISC links between the systems that are to be connected, as described in Chapter 13, “Defining links to remote systems,” on page 143.
3. Define the terminals and transactions that will participate in transaction routing, as described in Chapter 16, “Defining remote resources,” on page 191.
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 17, “Defining local resources,” on page 217.
5. If you want to use dynamic transaction routing, customize the supplied dynamic 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 24, “Intersystem session queue management,” on page 267.

Chapter 8. CICS distributed program link

This chapter describes CICS distributed program link (DPL).

It contains:

Overview of DPL

CICS distributed program link 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. Typically, entries in the CICS program definition tables specify that the named program is not in the local region (known as the **client region**) but in a remote region (known as the **server region**).

An illustration of a DPL request is given in Figure 28. 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 25) 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.

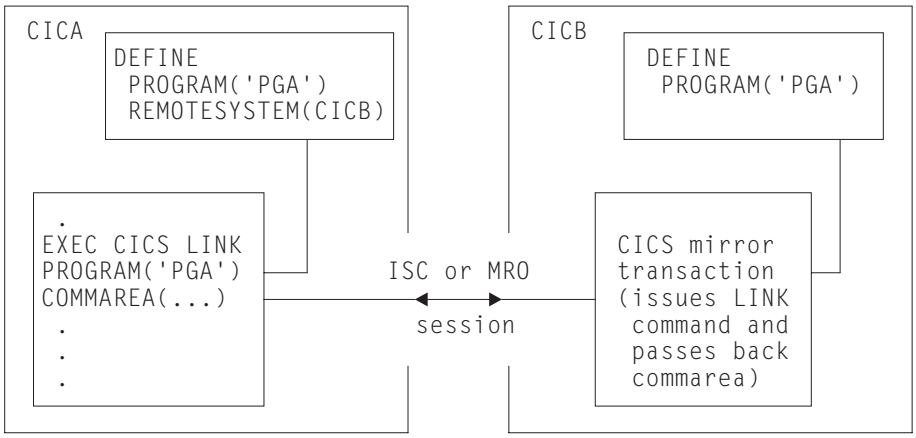


Figure 28. Distributed program link

The CICS recovery and restart facilities enable resources in remote 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.

A client program can run in a CICS intercommunication environment and use DPL without being aware of the location of the server program. CICS is made aware of the location of the server program in one of two ways. DPL requests can be routed to the server region either *statically* or *dynamically*.

Note: Provided that both the client and the server regions are CICS TS for z/OS, Version 3.2 or later, DPL is supported over IPIC connections, as well as over MRO and ISC over SNA connections. Support for DPL over TCP/IP is equivalent to that for DPL over MRO and DPL over SNA: for example, both two-phase commit and containers are supported.

If there is both an IPIC connection and an ISC over SNA connection between two CICS regions, and both are named the same, the IPIC connection takes precedence. That is, if remote region “CICB” is defined by both an IPCONN definition and a CONNECTION definition, CICS uses the IPCONN definition.

Statically routing DPL requests

Static routing means that the location of the server program is specified at design time, rather than at run-time. DPL requests for a particular remote program are always routed to the same server region. Typically, when static routing is used, the location of the server program is specified in the installed program resource definition. (Details are given in “Defining remote resources for DPL” on page 196.)

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) ...
```

Note: Although doing so may limit the client program's independence, the client program can name the remote system explicitly by using the SYSID option on the LINK command. If this option names a remote system, CICS routes the request to that system unconditionally. If the value of the SYSID option is “hard-coded”—that is, it is not deduced, from a range of possibilities, at run-time—this method is another form of static routing.

The local system can also be specified on the SYSID option. This means that the decision whether to link to a remote server program or a local one can be taken at run-time. This approach is a simple form of **dynamic routing**.

In the client region (CICA in Figure 29), 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.

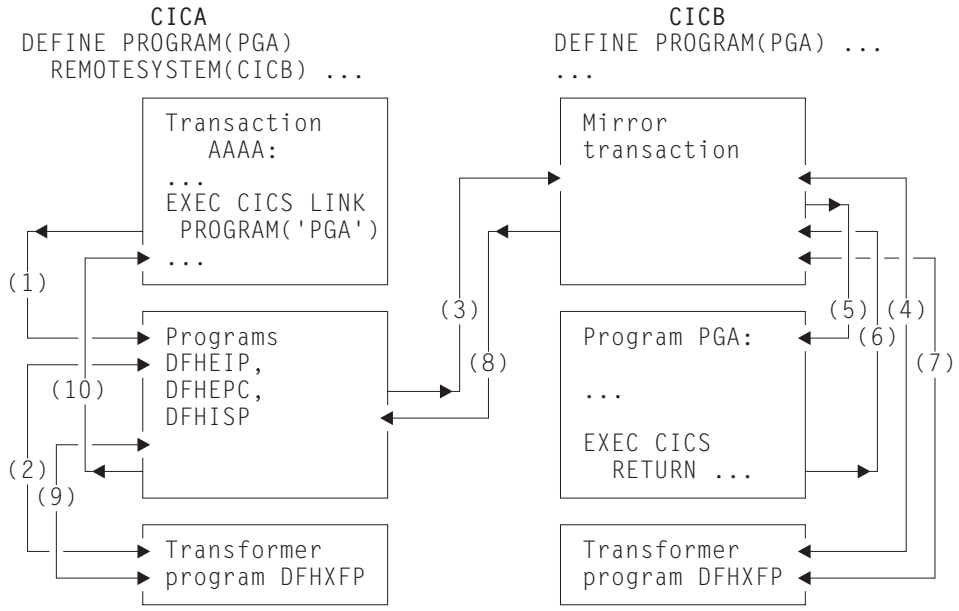


Figure 29. The transformer program and the mirror in DPL

As line (4) in Figure 29 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 communications area. 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.

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. However, a better way of doing this is to use the CICS dynamic routing program—see “Dynamically routing DPL requests.”

For programming information about writing XPCREQ and XPCREQC global user exit programs, see Program control program exits, in the *CICS Customization Guide*.

Dynamically routing DPL requests

Dynamic routing models:

Dynamic routing of DPL requests received from outside CICS uses the “hub” routing model described in “The “hub” model” on page 50.

Dynamic routing of CICS-to-CICS DPL requests uses the distributed routing model described in “The distributed model” on page 51. Note, however, that it is the *dynamic* routing program, not the distributed routing program, that is invoked for routing CICS-to-CICS DPL requests.

Dynamic routing means that the location of the server program is decided at run-time, rather than at design time. DPL requests for a particular remote program may be routed to different server regions. For example, if you have several cloned application-owning regions, you may want to use dynamic routing to balance the workload across the regions.

For eligible DPL requests, a user-replaceable program called the **dynamic routing program** is invoked. (This is the same dynamic routing program that is invoked for transactions defined as DYNAMIC—see “Dynamic transaction routing” on page 57.) The routing program selects the server region to which the program-link request is shipped.

The default dynamic 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 the dynamic routing program in particular, see Writing a dynamic routing program, in the *CICS Customization Guide*.

In the server region to which the program-link request is shipped, the mirror transaction is invoked in the way described for static routing.

Which requests can be dynamically routed?

For a program-link request to be eligible for dynamic routing, the remote program must either:

- Be defined to the local system as DYNAMIC(YES), *or*
- Not be defined to the local system.

Note: If the program specified on an EXEC CICS LINK command is not currently defined, what happens next depends on whether program autoinstall is active:

- If program autoinstall is inactive, the dynamic routing program is invoked.
- If program autoinstall is active, the autoinstall user program is invoked. The dynamic routing program is then invoked only if the autoinstall user program:
  - Installs a program definition that specifies DYNAMIC(YES), or
  - Does not install a program definition.

  For further information about autoinstalling programs invoked by EXEC CICS LINK commands, see “When definitions of remote server programs aren't required” on page 197.

As well as “traditional” CICS-to-CICS DPL calls instigated by EXEC CICS LINK PROGRAM commands, program-link requests received from outside CICS can also be dynamically routed. For example, all of the following types of program-link request can be dynamically routed:

- Calls received from:
  - The CICS Web Interface
  - The CICS Gateway for Java
- Calls from external CICS interface (EXCI) client programs
- External Call Interface (ECI) calls from any of the CICS Client workstation products
- Distributed Computing Environment (DCE) remote procedure calls (RPCs)
- ONC/RPC calls.

A program-link request received from outside CICS can be dynamically routed by:

- Defining the program to CICS Transaction Server for z/OS as DYNAMIC(YES)
- Coding your dynamic routing program to route the request.

When the dynamic routing program is invoked

For eligible program-link requests, [3] the dynamic routing program is invoked at the following points:

- Before the linked-to program is executed, to either:
  - Obtain the SYSID of the region to which the link should be routed.

    Note: The address of the caller's communication area (COMMAREA) is passed to the routing program, which can therefore route requests by COMMAREA contents if this is appropriate.
  - Notify the routing program of a statically-routed request. This occurs if the program is defined as DYNAMIC(YES)—or is not defined—but the caller specifies the name of a remote region on the SYSID option on the LINK command.

    In this case, specifying the target region explicitly takes precedence over any SYSID returned by the dynamic routing program.
- If an error occurs in route selection—for example, if the SYSID returned by the dynamic routing program is unavailable or unknown, or the link fails on the specified target region—to provide an alternate SYSID. This process iterates until either the program-link is successful or the return code from the dynamic routing program is not equal to zero.
- After the link request has completed, if reinvocation was requested by the routing program.
- If an abend is detected after the link request has been shipped to the specified remote system, if reinvocation was requested by the routing program.

Using CICSPlex SM to route requests

If you use the CICSPlex System Manager (CICSPlex SM) product to manage your CICSplex, you may not need to write your own dynamic routing program. 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 regions in the CICSplex can participate in dynamic routing.

---

3. By *program-link requests* we mean both “traditional” CICS-to-CICS DPL calls and requests received from outside CICS.

Using CICSPlex SM, you could integrate workload balancing for program-link requests with that for terminal-initiated transactions.

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

How CICS obtains the transaction ID

A transaction identifier is always associated with each dynamic program-link request. CICS obtains the transaction ID using the following sequence:

1. From the TRANSID option on the LINK command
2. From the TRANSID option on the program definition
3. 'CSMI', the generic mirror transaction. This is the default if neither of the TRANSID options are specified.

If you write your own dynamic routing program, perhaps based on DFHDYP, the transaction ID associated with the request may not be significant—you could, for example, code your program to route requests based simply on program name and available AORs.

However, if you use CICSPlex SM to route your program-link requests, the transaction ID becomes much more significant, because CICSPlex SM's routing logic is transaction-based. CICSPlex SM routes each DPL request according to the rules for its associated transaction as specified in the Transaction Group (TRANGRP), Workload Management Definition (WLMDEF) and Workload Management Specification (WLMSPEC) resource tables.

Note: The CICSPlex SM system programmer can use the EYU9WRAM user-replaceable module to change the transaction ID associated with a DPL request.

Daisy-chaining of DPL requests

Statically-routed DPL requests can be daisy-chained from region to region. For example, imagine that you have three CICS regions—A, B, and C. In region A, a program P is defined with the attribute REMOTESYSTEM(B). In region B, P is defined with the attribute REMOTESYSTEM(C). An EXEC CICS LINK PROGRAM(P) command issued in region A is shipped to region B for execution, from where it is shipped to region C.

Dynamically-routed DPL requests cannot be daisy-chained from region to region. Imagine two CICS regions, A and B. A program P is defined as DYNAMIC(YES)—or is not defined—in both regions. An EXEC CICS LINK PROGRAM(P) command is issued in region A. The dynamic routing program is invoked in region A and routes the request to region B. In region B, the dynamic routing program is not invoked, even though program P is defined as DYNAMIC(YES); P runs locally, in region B.

Limitations of DPL server programs

A DPL 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.

Attention: 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 20, “Application programming for CICS DPL,” on page 235.

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 24, “Intersystem session queue management,” on page 267.

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.

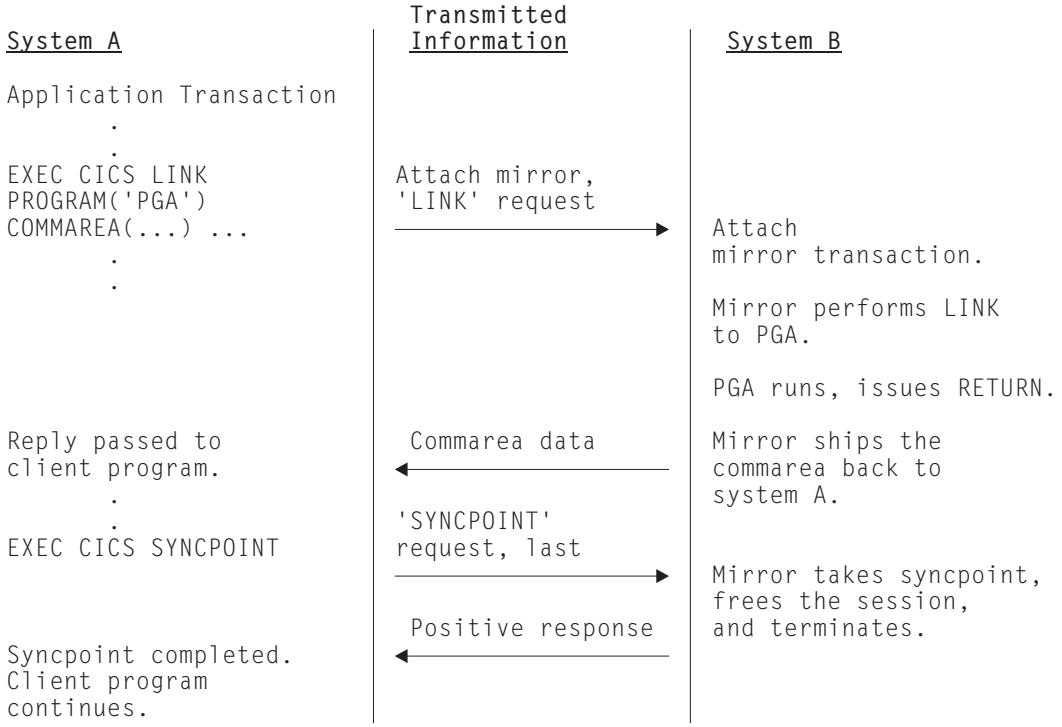


Figure 30. DPL with the client transaction issuing a syncpoint

Figure 30 shows a DPL request on which the client transaction issues a syncpoint. Because the mirror is always long-running, it does not terminate before SYNCPOINT is received.

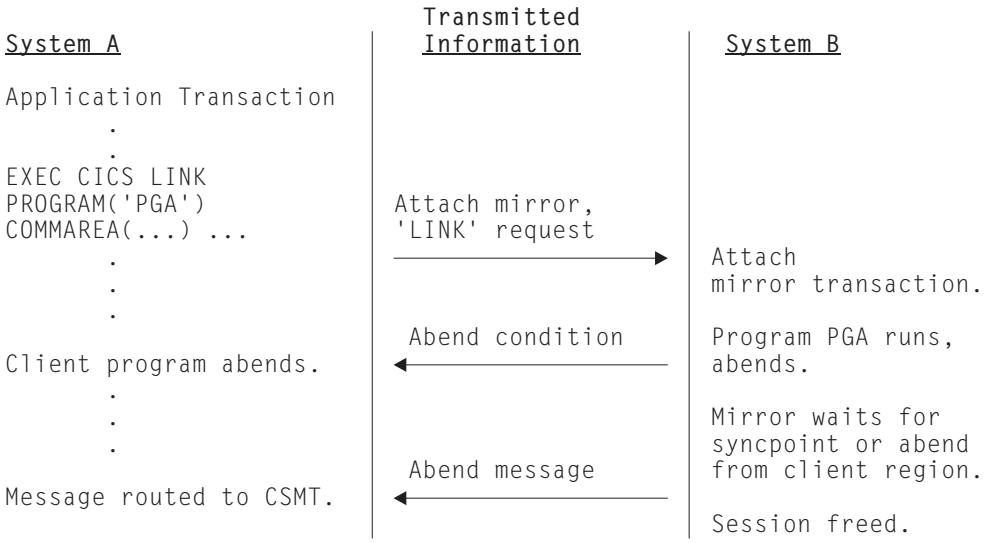


Figure 31. DPL with the server program abending

Figure 31 on page 93 shows a DPL request on which the server program abends.

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 means inefficient use of the links.
- The transactions associated with the point-of-sale devices are competing for sessions with the remote systems. This could mean 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 32 on page 98 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.

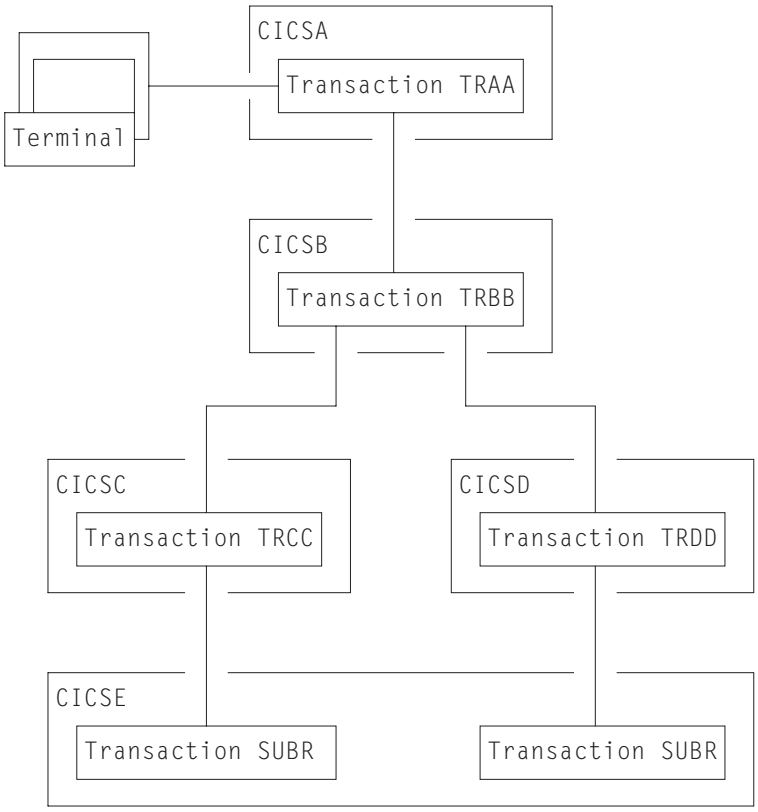


Figure 32. DTP in a multisystem configuration

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 in “Advantages over function shipping and transaction routing” on page 95, the DTP solution is to transmit the contents of the transient data queue

from the front end to the back end. The front end issues a SEND command for each record that it takes off the queue. The back end issues RECEIVE commands until it receives an indication that the transmission has ended.

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

Control flows and brackets

During a conversation, data passes over the link in both directions. A single transmission is called a **flow**. Issuing a SEND command does not always cause a flow. This is because the transmission of user data can be deferred; that is, held in a buffer until some event takes place. The 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 **unit of work** (UOW).

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 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. You cannot use MRO to communicate with a partner in a non-CICS system. Further, it supports communication between transactions running in CICS systems in different MVS images only if the MVS images are in the same MVS sysplex, and are joined by cross-system coupling facility (XCF) links. (For full details of the hardware and software requirements for XCF/MRO, see “Requirements for XCF/MRO” on page 108.)

For communication with a partner in another CICS system, where the CICS systems are either in the same MVS image, or in the same sysplex, 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 3 summarizes the main differences between the two protocols.

Table 3. 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 MVS image, or (using XCF/MRO) between MVS images in same sysplex	Communicates across multiple MVS images 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 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 4 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 4. 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.

Table 4. APPC conversations – mapped or basic? (continued)

Mapped	Basic
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 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 5 compares the two methods to help you to decide which API to use for a particular application.

Table 5. 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. Installing intercommunication support

This part of the manual discusses the installation requirements for a CICS system that is to participate in intersystem communication or multiregion operation. It describes:

- How to set up CICS for multiregion operation.
- How to set up CICS for intersystem communication. It also contains notes on the installation requirements of ACF/VTAM and IMS when these products are to be used with CICS in an intersystem communication environment.
- How to register your terminal-owning regions as members of a VTAM generic resource group, and things you need to consider when doing so.

Chapter 10. Installation considerations for multiregion operation

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

The information on MVS/ESA given in this chapter is for guidance only. Always consult the current MVS/ESA publications for the latest information.

Steps to install MRO

To install support for multiregion operation, you must:

1. Define CICS as an MVS subsystem
2. Ensure that the required CICS modules are included in your CICS system
3. Place some modules in the MVS link pack area (LPA).

Installing support for cross-system MRO (XCF/MRO) requires some additional administration. This is described in "Requirements for XCF/MRO" on page 108.

Adding CICS as an MVS subsystem

Multiregion operation with CICS Transaction Server for z/OS requires MVS/VS Subsystem Interface (SSI) support. You must therefore install CICS as an MVS subsystem. For information about how to do this, see Cross-domain considerations, in the *CICS Transaction Server for z/OS Installation Guide*.

Modules required for MRO

You must include the intersystem communication management programs in your system by specifying ISC=YES on the system initialization parameters.

MRO modules in the MVS link pack area

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

You must place the CICS Transaction Server for z/OS, Version 3 Release 2 versions of the following modules in the link pack area (LPA) of MVS.

- DFHCSVC – the CICS type 3 SVC module

  Multiregion operation requires the CICS interregion communication modules to run in supervisor state to transfer data between different regions. CICS achieves this by using a normal supervisor call to this startup SVC routine, which is in the pregenerated system load library (CICSTS32.CICS.SDFHLOAD).

  The SVC must be defined to MVS. For information about how to do this, see Defining the CICS SVCs to your MVS, in the *CICS Transaction Server for z/OS Installation Guide*.
- DFHIRP – the CICS interregion communication program.

MRO data sets and starter systems

To help you get started with MRO, a CICS job and a CICS startup procedure are supplied on the CICS distribution volume. For each MRO region, you must also create the CICS system data sets needed. See Creating the CICS data sets, in the *CICS Transaction Server for z/OS Installation Guide* for information about this.

Requirements for XCF/MRO

Communication across MVS images using XCF/MRO requires the MVS images to be joined in a sysplex.

A sysplex consists of multiple MVS images, coupled together by hardware elements and software services. In a sysplex, MVS images provide a platform of basic services that multisystem applications like CICS can exploit. As an installation's workload grows, additional MVS images can be added to the sysplex to enable the installation to meet the needs of the greater workload.

Usually, a specific function (one or more modules/routines) of the MVS application subsystem (such as CICS) is joined as a **member** (a member resides on one MVS image in the sysplex), and a set of related members is the **group** (a group can span one or more of the MVS images in the sysplex). A group is a complete logical entity in the sysplex. To use XCF to communicate in a sysplex, each CICS region joins an XCF group as a member, using services provided by DFHIRP.

Sysplex hardware and software requirements

For definitive information about installing and managing MVS systems in a sysplex, see the *MVS/ESA Setting Up a Sysplex* manual, GC28-1449.

Generating XCF/MRO support

1. Depending on the versions of CICS installed in the MVS images participating in XCF/MRO, the versions of DFHIRP installed in the images can be different. For all the MVS images containing CICS systems to be linked, ensure that the version of DFHIRP in the extended link pack area (ELPA) is at the required level. The DFHIRP module should be that from the most current CICS release in the image, or higher.

   Note: The CICS TS for z/OS, Version 3.2 version of DFHIRP (required for multiple XCF group support) can be used only on z/OS Version 1.7 or later. (Although z/OS has supported multiple XCF groups since Version 1.6, CICS TS for z/OS, Version 3.2 (required to join an XCF group other than DFHIR000) requires z/OS Version 1.7 or later.)
2. Ensure that each CICS APPLID is unique within the sysplex.
3. Ensure that the value of the MAXMEMBER MVS parameter, used to define the XCF couple data sets, is high enough to cater for the largest CICS XCF group. The maximum size of any XCF group within a sysplex is limited by this value. The theoretical maximum size of any XCF group is 2047 members.

   External CICS interface (EXCI) users that use an XCF/MRO link also join an XCF group. You should therefore set the value of MAXMEMBER high enough to allow all CICS regions and EXCI XCF/MRO users in the largest CICS XCF group to join the group concurrently.

   To list the CICS regions and EXCI users in an XCF group, use the MVS DISPLAY command. For example, to list the CICS regions and EXCI users in the DFHIR001 XCF group, use the command:

```
DISPLAY XCF,GROUP,DFHIR001,ALL
```

Attention:

Do not rely on the default value of MAXMEMBER, which may be too low to allow all the CICS regions and EXCI users in the largest XCF group to join the group. This is especially important if you have only a few CICS XCF groups.

Likewise, do not set a value much larger than you need, because this will result in large couple data sets for XCF. The larger the data set, the longer it will take to locate entries.

We suggest that you make the value of MAXMEMBER 10-15 greater than the combined number of CICS regions and EXCI users in the largest CICS XCF group.

Each CICS region joins an XCF group when it logs on to DFHIRP. Its member name is its APPLID (NETNAME) used for MRO partners. The XCF group name is specified on the XCFGROUP system initialization parameter. If XCFGROUP is not specified, the XCF group name defaults to DFHIR000.

At connect time, CICS invokes the IXCQUERY macro to determine whether the CICS region being connected to resides in the same MVS image. If it does, CICS uses IRC or XM as the MRO access method, as defined in the connection definition. If the partner resides in a different MVS image, CICS uses XCF as the access method, regardless of the access method defined in the connection definition.

Note: CICS regions can use MRO or XCF/MRO to communicate *only with regions in the same XCF group*. Members of different XCF groups cannot communicate via MRO (or XCF/MRO), *even if they are in the same MVS image*.

Further steps

Once you have installed MRO support, to enable CICS to use it you must:

1. Define MRO links to the remote systems. See “Defining links for multiregion operation” on page 145.
2. Define resources on both the local and remote systems. See Chapter 17, “Defining local resources,” on page 217 and Chapter 16, “Defining remote resources,” on page 191, respectively.
3. Specify that CICS is to log on to the IRC access method. See Installing MRO and ISC support, in the *CICS Transaction Server for z/OS Installation Guide*.

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.

The chapter contains the following topics:

- "Installing support for intersystem communication over SNA"
- "Installing support for IP interconnectivity" on page 117

Installing support for intersystem communication over SNA

This topic describes how to install support for intersystem communication over SNA (ISC over SNA). It is intended to be read in conjunction with the *CICS Transaction Server for z/OS Installation Guide*.

The topic also contains notes on installing ACF/VTAM and IMS when these products are used with ISC over SNA.

The topic contains the following topics:

- "Modules required for ISC"
- "ACF/VTAM definition for CICS"
- "Considerations for IMS" on page 112.

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

Modules required for ISC

You must include the intersystem communication programs in your system (by specifying 'YES' on the VTAM and ISC system initialization parameters). For information about specifying system initialization parameters, see Specifying CICS system initialization parameters, in the *CICS System Definition Guide*.

ACF/VTAM definition for CICS

When you define your CICS system to ACF/VTAM, include the following operands in 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 "ACF/VTAM LOGMODE table entries for CICS" on page 112.) You may omit this operand if you choose to add your MODEENT entries to the IBM default logon mode table (without renaming it).

AUTH=(ACQ,SPO,VPACE[,PASS])
ACQ is required to allow CICS to acquire LU type 6 sessions. SPO is required to allow CICS to issue the MVS MODIFY *vtamname* USERVAR command. (For further information about the significance of USERVARs, see the *CICS/ESA 3.3 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 option specifies LU type 6 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
APPC=NO is required for CICS. This setting is the default. If you do not use APPC=NO, you receive message DFHZC2400E, referencing the VTAM return code 1013.

For further information about the VTAM APPL statement, refer to the *OS/390 eNetwork Communications Server: SNA Resource Definition Reference* manual.

ACF/VTAM LOGMODE table entries for CICS

For APPC sessions, you can use the MODENAME option of the CICS DEFINE SESSIONS command (see “Defining APPC links” on page 155) to identify a VTAM logmode entry that in turn identifies the required entry in the VTAM class-of-service table. Every modename that you supply, when you define a group of APPC sessions to CICS, must be matched by a VTAM LOGMODE name. All that is required in the VTAM LOGMODE table are entries of the following form:

```
MODEENT LOGMODE=modename
MODEEND
```

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

```
MODEENT LOGMODE=SNASVCMG
MODEEND
```

If you plan to use autoinstall for single-session APPC terminals, additional information is required in the MODEENT entry. For programming 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 parameters.

Considerations for IMS

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

The following sections are intended to help you communicate effectively with the person responsible for installing the IMS system. They may also be helpful if you

have that responsibility. You should also refer to Chapter 13, “Defining links to remote systems,” on page 143, especially the section on defining compatible CICS and IMS nodes. For full details of IMS installation, refer to the *IMS/ESA Installation Guide*.

ACF/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 LU type 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 LU type 6 parallel session support.

For further information about the VTAM APPL statement, see the *OS/390 eNetwork Communications Server: SNA Resource Definition Reference* manual.

ACF/VTAM LOGMODE table entries for IMS: IMS allows the user to specify some BIND parameters in a VTAM logmode table entry. The CICS logmode table entry must match that of the IMS system. IMS uses (in order of priority) the mode table entry specified in:

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

Figure 33 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 33. 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 refer to “Defining compatible CICS and IMS nodes” on page 165. For full details of IMS installation, refer to the installation guide for the IMS product.

The COMM macro:

APPLID=name
Specifies the applid of the IMS system. For an IMS system generated without XRF support, this is usually the name that you should specify on the NETNAME option of DEFINE CONNECTION when you define the IMS system to CICS.

However, bear the following in mind:

- For an IMS system with XRF, the CICS NETNAME option should specify the USERVAR (that is, the generic applid) that is defined in the DFSHSBxx member of IMS.PROCLIB, not the applid from the COMM macro.
- If APPLID on the COMM macro is coded as NONE, and XRF is not used, the CICS NETNAME option should specify the label on the EXEC statement of the IMS startup job.
- If the IMS system is started as a started task, NETNAME should specify the started task name.

For an explanation of how IMS system names are specified, see page “System names” on page 165.

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 option of DEFINE SESSIONS.

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

EDTNAME=name
Specifies an alias for ISCEDT in the IMS system. For CICS-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|MULTn}
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 per 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. An output component defined as SINGLE1 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.

Installing support for IP interconnectivity

This topic describes how to install support for IP interconnectivity (IPIC). It is intended to be read in conjunction with the *CICS Transaction Server for z/OS Installation Guide*.

- Specify `TCPIP=YES` in the CICS system initialization table, or as a system initialization override.
- Define an IPIC connection between the two CICS regions to be connected, as described in "Defining IP interconnectivity links" on page 151.

Chapter 12. Installation considerations for VTAM generic resources

Important:

For brevity, in this chapter, the terms *CICS Transaction Server for z/OS* and *CICS TS for z/OS* are used to mean all the following CICS products:

- CICS Transaction Server for z/OS Version 3 Release 2
- CICS Transaction Server for z/OS, Version 3 Release 1
- CICS Transaction Server for z/OS, Version 2 Release 3
- CICS Transaction Server for z/OS, Version 2 Release 2
- CICS Transaction Server for OS/390, Version 1 Release 3

This chapter describes how, in a CICSplex containing a set of functionally-equivalent CICS terminal-owning regions (TORs), you can use the VTAM generic resource function to balance terminal sessions across the available TORs.

For an overview of VTAM generic resources, see “Workload balancing in a sysplex” on page 17.

Note: This chapter assumes some knowledge of tasks, such as defining connections to remote systems, that are described in later parts of this book. It may help your understanding of this chapter if you have already read Chapter 13, “Defining links to remote systems,” on page 143.

The chapter contains the following topics:

Prerequisites for VTAM generic resources

To use VTAM generic resources:

- You need ACF/VTAM Version 4 Release 2 or a later, upward-compatible, release.
- VTAM must be:
  - Running under an MVS that is part of a sysplex.

- Connected to the sysplex coupling facility. For information about the sysplex coupling facility, see the *MVS/ESA Setting Up a Sysplex* manual, GC28-1449.
- At least one VTAM in the sysplex must be an advanced peer-to-peer networking (APPN) network node, with the other VTAMs being APPN end nodes.

Planning your CICSplex to use VTAM generic resources

You can use the VTAM generic resource function to balance terminal session workload across a number of CICS regions. You do this by grouping the CICS regions into a single generic resource. Each region is a **member** of the generic resource. When a terminal user logs on using the name of the generic resource (the **generic resource name**), VTAM establishes a session between the terminal and one of the members, depending upon the session workload at the time. The terminal user is unaware of which member he or she is connected to. It is also possible for a terminal user to log on using the name of a generic resource member (a **member name**), in which case the terminal is connected to the named member.

APPC and LUTYPE6.1 connections do not log on in the same way as terminals. But they too can establish a connection to a generic resource by using either the generic resource name (in which case VTAM chooses the member to which the connection is made) or the member name (in which case the connection is made to the named member).

When you plan your CICSplex to use VTAM generic resources, you need to consider the following:

- Which CICS regions should be generic resource members?

  Note that:
  - Only CICS regions that provide equivalent functions for terminal users should be members of the same generic resource.
  - A CICS region that uses XRF cannot be a generic resource member.
  - In a CICSplex that contains both terminal-owning regions and application-owning regions (AORs), TORs and AORs should not be members of the same generic resource group.
- Should there be one or many generic resources in the CICSplex?

  If you have several groups of end users who use different applications, you may want to set up several generic resources, one for each group of users. Bear in mind that a single CICS region cannot be a member of more than one generic resource at a time.
- Will there be APPC or LUTYPE6.1[4] connections:
  - Between members of a generic resource?[5]
  - Between members of one generic resource and members of another generic resource?
  - Between members of a generic resource and systems which are not members of generic resources?

  In all these cases you will need to understand when you can use:
  - Connection definitions that specify the generic resource name of the partner system

4. You are recommended to use APPC in preference to LUTYPE6.1 for CICS-to-CICS connections.

5. You cannot use LUTYPE6.1 connections between members of a generic resource.

– Connection definitions that specify the member name of the partner system
– Autoinstall to provide definitions of the partner system.

Naming the CICS regions

Every CICS region has a network name, defined on a VTAM APPL statement, that uniquely identifies it to VTAM. You specify this name, or *applid*, on the APPLID system initialization parameter. If a region is a member of a generic resource, its applid and member name are one and the same.

A generic resource—a collection of CICS regions—has a generic resource name. Each CICS region that is to be a member of a generic resource specifies the generic resource name on its GRNAME system initialization parameter. Unlike network names, generic resource names do not have to be defined to VTAM. However, they must be distinct from network names, and must be unique within a network. The *System/390 MVS Sysplex Application Migration* manual suggests naming conventions for CICS generic resources.

When you start to use generic resources, you must decide how the generic resource name and the member names are to relate to the applids by which the member regions were known previously:

- If you have several TORs, you could continue to use the same applids for the TORs, and choose a new name for the generic resource. Terminal logon procedures will need to be changed to use the generic resource name, and so will connection definitions that are to use the generic resource name.
- If you have a single TOR, you could use its applid as the generic resource name, and give it a new applid. Changes to terminal logon procedures (and connection definitions) are minimized, but you need to change VTAM definitions, CONNECTION definitions in AORs connected using MRO, and RACF profiles that specify the old applid.

Generic resources and XRF

Because you cannot use XRF with VTAM generic resources, the concept of “specific” and “generic” CICS applids is not meaningful to regions that are members of a generic resource group. Each generic resource member has only one applid.

For a full explanation of the relationships between generic and specific CICS applids, VTAM APPL statements, and VTAM generic resource names, see “Generic and specific applids for XRF” on page 175.

Defining connections in a generic resource environment

The VTAM generic resource function can be used to balance session workload for APPC and LUTYPE6.1 connections. Connections differ from terminal sessions in the following ways:

- A connection can have multiple sessions. VTAM's generic resource support creates dependencies, or **affinities**, to ensure that—once the first session is established—subsequent sessions to a generic resource are with the same member as the first session.
- Either end of a connection can (in principle) establish the first session. Which end does (in practice) initiate the first session affects how connections should be defined in the generic resource environment.
- Connections that fail, and require resynchronization, must be reestablished between the same members. VTAM uses affinities to ensure that reconnections are made correctly.

Defining connections

When you define a connection to a generic resource, you have two possibilities for the NETNAME option of DEFINE CONNECTION:

1. Use the name (applid) of the generic resource member. This type of connection is known as a **member name connection**.
2. Use the name of the generic resource. This type of connection is known as a **generic resource name connection**.

It is important that you make the correct choice when you define connections to a generic resource:

- When CICS initiates a connection using a member name definition, VTAM establishes a session with the named member.
- When CICS initiates a connection using a generic resource name connection, VTAM establishes a connection to one of the members of the generic resource. Which member it chooses depends upon whether any affinities exist, and upon VTAM's session-balancing algorithms.

When a CICS Transaction Server for z/OS generic resource member sends a BIND request on a connection, the request contains the generic resource name and the member name of the sender. If the partner is also a CICS TS for z/OS generic resource, it can distinguish both names. Other CICS systems take the generic resource name from the bind, and attempt to match it with a connection definition.

It follows that the only time an LUtype 6 which is not itself a member of a CICS TS for z/OS generic resource can successfully use a member name to connect to a generic resource is when the generic resource member will never initiate any sessions. This is an unusual situation, and we therefore recommend that a connection from a system that is not a CICS TS for z/OS generic resource member to a generic resource should use the generic resource name.

Defining connections between GR members and non-GR members

When a generic resource member initiates a connection (that is, sends the first BIND) to another LUtype 6, it identifies itself to its partner with its generic resource name. Sessions initiated by the partner must then also use the generic resource name of the LU that initiates the connection.

Defining connections between members within a generic resource

You may want to define connections between members of a generic resource. You should always specify, on the NETNAME option of these CONNECTION definitions, the partner's member name and *not* the generic resource name.

Defining connections between CICS TS for z/OS generic resources

If you have two CICS TS for z/OS generic resources, you do not need to define and install member name connections for every possible connection between them. Instead, you can define and install a single generic resource name connection in each member that may initiate a connection with the partner generic resource. CICS then autoinstalls member name connections as they are required.

The only connection definition required in a CICS region that does not initiate connections is one that can be used as an autoinstall template. If there is a generic

resource name connection installed, it is used as the template, so we suggest that you define generic resource name connections for this purpose.

Generating VTAM generic resource support

To generate VTAM generic resource support for your CICS TORs, you must:

1. Use the GRNAME system initialization parameter to define the generic resource name under which CICS is to register to VTAM. To comply with the CICS naming conventions, it is recommended that you pad the name to the permitted 8 characters with one of the characters #, @, or $.

   For example:

   ```
   GRNAME=CICSH###
   ```

   For details of the GRNAME system initialization parameter, see GRNAME, in the *CICS System Definition Guide*. The CICS naming conventions are described in the *System/390 MVS Sysplex Application Migration* manual.
2. Use an APPL statement to define the attributes of each participating TOR to VTAM. The attributes defined on each individual APPL statement should be identical. The name on each APPL statement must be unique. It identifies the TOR individually, within the generic resource group.
3. Shut down each terminal-owning region cleanly before registering it as a member of the generic resource. “Cleanly” means that CICS must be shut down by means of a CEMT PERFORM SHUTDOWN NOSDTRAN command. A CEMT PERFORM SHUTDOWN IMMEDIATE is *not* sufficient; nor is a CICS failure followed by a cold start. You should specify NOSDTRAN to prevent the possibility of the shutdown assist transaction force closing VTAM or performing an immediate shutdown. (The default shutdown assist transaction, DFHCESD, is described in Shutdown assist program (DFHCESD), in the *CICS Operations and Utilities Guide*.)

   If CICS has *not* been shut down cleanly before you try to register it as a member of a generic resource, VTAM may (due to the existence of persistent sessions) fail to register it, and issue a return code-feedback (RTNCD-FDB2) of X'14', X'86'. (VTAM RTNCD-FDB2s are described in the *OS/390 eNetwork Communications Server: SNA Programming* manual.) To correct this, you must restart CICS (with the same APPLID), and use a CEMT PERFORM SHUTDOWN NOSDTRAN command to shut it down cleanly. Alternatively, if you have written a batch program to end affinities (see page “Writing a batch program to end affinities” on page 132), you might be able to use it to achieve the same effect. As part of its processing, the skeleton program described on page “Writing a batch program to end affinities” on page 132 opens the original VTAM ACB with the original APPLID, unbinds any persisting sessions, and closes the ACB.

Note:

1. If your CICSplex comprises separate terminal-owning regions and application-owning regions, you should not include TORs and AORs in the same generic resource group.
2. You cannot use VTAM generic resources with XRF. If you specify 'YES' on the XRF system initialization parameter, any value specified for GRNAME is ignored.
3. If you specify a valid generic resource name on GRNAME, you should specify only *name1* on the APPLID system initialization parameter. (If you do specify both *name1* and *name2* on the APPLID parameter, CICS ignores *name1* and uses *name2* as the VTAM applid.)

For detailed information about generating VTAM generic resource support, see the *OS/390 eNetwork Communications Server: SNA Network Implementation*.

Migrating a TOR to a generic resource

This section describes how to manage existing terminals and connections when migrating a TOR to membership of a CICS Transaction Server for z/OS generic resource. How to establish connections between two CICS TS for z/OS generic resources is described separately in "Setting up inter-sysplex communications between generic resources" on page 126.

Note: For the purposes of this discussion, a "terminal-owning region" is any CICS region that owns terminals and is a candidate to be a member of the generic resource.

Recommended methods

In general, we advise that:

- For simplicity, you first create a generic resource consisting of only one member. Do not add further members until the single-member generic resource is functioning satisfactorily.
- Because all members of a generic resource should be functionally equivalent, you create additional members by cloning the first member. (A situation in which you might choose to ignore this advice is described below.)

There are two recommended methods for migrating a TOR to a generic resource. Which you use depends on whether there are existing LU6 connections.

No LU6 connections

If there are no LU6 (that is, APPC or LU6.1) connections to your terminal-owning region, we recommend that you choose a new name for the generic resource and retain your old applid. Non-LU6 terminals can log on by either applid or generic resource name, hence they are not affected by the introduction of the generic resource name. You can then gradually migrate the terminals to using the generic resource name. Later, you can expand the generic resource by cloning the first member-TOR.

Note: If you have several existing TORs that are functionally similar, rather than cloning the first member you might choose to expand the generic resource by adding these existing regions, using their applids as member-names.

LU6 connections

If there are LU6 (APPC or LU6.1) connections to your terminal-owning region[6] , we recommend that they log on using the generic resource name. However, you will probably want to migrate to generic resource without requiring all your LU6 network partners to change their logon procedures. One option is to use the applid of your existing terminal-owning region as the new generic resource name. Because this requires you to choose a new applid, it is also necessary to change the CONNECTION definitions of MRO-connected application-owning regions and RACF profiles that specify the old applid. Note, however, that you do not need to change the APPL profile to which the users are authorized—CICS passes the GRNAME to

6. Not counting connections to other members of the generic resource.

RACF as the APPL name during signon validation, and the old applid is now the GRNAME. The recommended migration steps are:

1. Configure your CICSplex with a single terminal-owning region.
2. Set the generic resource name to be the current applid of that terminal-owning region.
3. Change the current applid to a new value.
4. Change CONNECTION definitions in MRO partners to use the new applid for the terminal-owning region.
5. Change RACF profiles that specify the old applid.
6. Restart the CICSplex.

   At this point:

   - Non-LU6 terminals can log on using the old name (without being aware that they are now using a VTAM generic resource). They will, of course, be connected to the same TOR as before because there is only one in the generic resource set.
   - LU6 connections log on using the old name (thereby conforming to the recommendation that they should connect by generic resource name).
7. Install new cloned terminal-owning regions with the same generic resource name and the same connectivity to the set of AORs.

   At this point:

   - Autoinstalled non-LU6 terminals start to exploit session balancing.
   - Autoinstalled APPC sync level 1 connections start to exploit session balancing.
   - Because of affinities, existing LU6.1 and APPC sync level 2 connections continue to be connected to the original terminal-owning region (by generic resource name).
   - Special considerations apply to non-autoinstalled terminals and connections, and to LU6 connections used for outbound requests. These are described in “Dealing with special cases” on page 139.

Removing a TOR from a generic resource

There are several ways to remove a region from a generic resource:

- Issue a SET VTAM CLOSED command to close the VTAM ACB.
- Shut down CICS. If you want to remove the region permanently, you must remove the generic resource name from the GRNAME system initialization parameter before restarting CICS.
- Issue a SET VTAM DEREGISTERED command to remove the region *dynamically*—that is, without closing the VTAM ACB or shutting down CICS. This may be useful if, for example, you need to apply minor maintenance to a TOR.

  When a TOR is dynamically removed from a generic resource, any terminals which are logged on are gradually redirected to the remaining generic resource members, as they log off and back on again.

  To re-register CICS with the generic resource, you must close and reopen the VTAM ACB.

  For details of the SET VTAM DEREGISTERED command, see SET VTAM, in the *CICS System Programming Reference* manual and CEMT SET VTAM, in the *CICS Supplied Transactions* manual.

Important:

If you remove a region from a generic resource:

- You should end any affinities that it owns. If you do not, VTAM will not allow the affected APPC and LU6.1 partners to connect to other members of the generic resource. See “Ending affinities” on page 131.
- The region that has been removed should not try to acquire a connection to a partner that knows it by its generic resource name, unless the partner has ended its affinity to the removed region.

Moving a TOR to a different generic resource

To move a region from one generic resource to another, you must:

1. End any affinities that it owns. See “Ending affinities” on page 131.
2. Shut it down cleanly. See “Generating VTAM generic resource support” on page 123.

   If CICS is *not* shut down cleanly before you try to register it as a member of the new generic resource, VTAM may fail to register it, and issue a RTNCD-FDB2 of X'14', X'86'. To correct this, you must restart CICS with the *original* GRNAME and APPLID, and use a CEMT PERFORM SHUTDOWN NOSDTRAN command to shut it down cleanly. Alternatively, if you have written a batch program to end affinities, you might be able to use it to achieve the same effect. As part of its processing, the skeleton program described on page “Writing a batch program to end affinities” on page 132 opens the original VTAM ACB with the original GRNAME, unbinds any persisting sessions, and closes the ACB.
3. Specify the name of the alternative generic resource on the GRNAME system initialization parameter, and restart CICS.

Setting up inter-sysplex communications between generic resources

This section describes communications between CICS Transaction Server for z/OS generic resources in partner sysplexes. You must use APPC parallel-session connections for links between CICS TS for z/OS generic resources.

Establishing connections between CICS TS for z/OS generic resources

Assume that you have two sysplexes, SYSPLEXL and SYSPLEXR, and that these contain the CICS TS for z/OS generic resource groups CICSL and CICSR, respectively (see Figure 34 on page 128). The steps involved in establishing connections between CICSL and CICSR are as follows:

1. On each member of CICSL that is to initiate a connection to CICSR, statically define and install an APPC parallel-session connection in which the NETNAME is the generic resource name of CICSR—that is, define a *generic resource name connection*. Similarly, on each member of CICSR that is to initiate a connection to CICSL, statically define and install an APPC parallel-session connection in which the NETNAME is the generic resource name of CICSL.

   Note: You should not install any predefined connections other than generic resource name connections.

   The first attempt by any member of CICSL to acquire a connection to CICSR (or vice versa) uses a generic resource name connection.
2. The CICSR member to which VTAM sends the bind request searches for the generic resource name connection definition for CICSL. (If none exists, it autoinstalls one, subject to the normal rules for autoinstalling connections.)

3. Subsequent connections that VTAM happens to route to the same member of CICSR from different members of CICSL are autoinstalled on the CICSR member, using the CICSL member name as the NETNAME; that is, CICS autoinstalls *member name connections*. Similarly, subsequent connections to the same member of CICSL from different members of CICSR are autoinstalled on the CICSL member, using the CICSR member name as the NETNAME. The example in "Example" makes this clearer.

   The template used for autoinstalling these further connections can be any installed connection. CICS uses the generic resource name connection as the default template.

   If you decide to use a template other than the default for member name connections, remember that use of the sessions for these connections is initiated by the partner, so consider defining the MAXIMUM option with no contention winners. [7] (This is useful because the member name is not known to the applications in the system in which the member name connection is autoinstalled. They use the GR name for outbound requests. Therefore the member name connection is not used for outbound requests and so does not need to have any sessions defined as winners. By allowing the partner system to have all the sessions as winners, the overhead of bidding for loser sessions is avoided.)

   A template is a normal installed connection defined with CONNECTION and SESSIONS that can be used solely as a template, or as a real connection. It is used as a model from which to autoinstall further connections.

Example

In Figure 34 on page 128 through Figure 37 on page 130, each generic resource uses the partner sysplex's generic resource name when initiating a connection. All generic resource members are able to initiate connections; that is, they all have a generic resource name connection (a predefined connection entry in which the NETNAME is the generic resource name of the partner sysplex). The connections are APPC parallel-session synclevel 2 links.

7. The MAXIMUM option of DEFINE SESSIONS is described in "Defining groups of APPC sessions" on page 157.

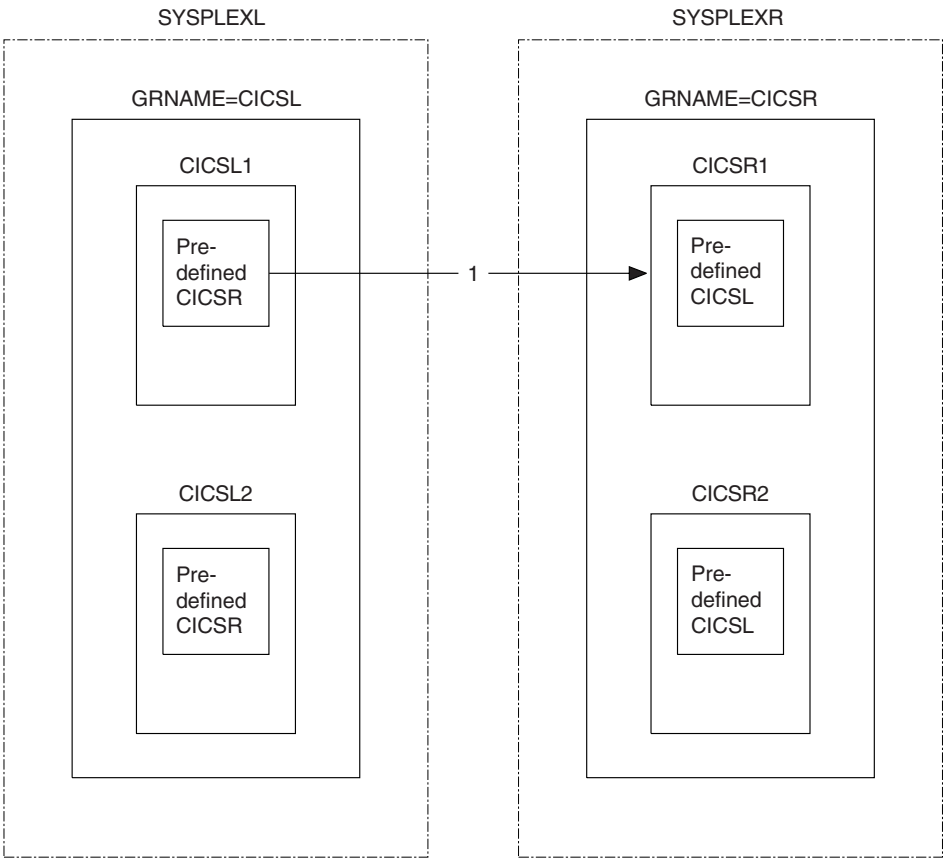


Figure 34. The figure shows two sysplexes, SYSPLEXL and SYSPLEXR. Each contains a CICS generic resource group. The CICSL1 member of the CICSL group attempts to acquire a connection to a member of the CICSR group in SYSPLEXR.

In Figure 34, the first bind that flows from CICSL1 to CICSR is routed to whichever member of CICSR VTAM decides is the most lightly loaded. In this example it goes to CICSR1. The predefined connections for the generic resource names CICSR and CICSL in CICSL1 and CICSR1 are used.

Affinities are created at SYSPLEXL and SYSPLEXR, associating CICSL1 with CICSR1. When you need to end these affinities, you may or may not need to do so explicitly—see “Ending affinities” on page 131 and “APPC connection quiesce processing” on page 292. Until the affinities are ended, whenever CICSL1 tries to reconnect to CICSR, VTAM routes the request to CICSR1; and whenever CICSR1 tries to reconnect to CICSL, VTAM routes the request to CICSL1.

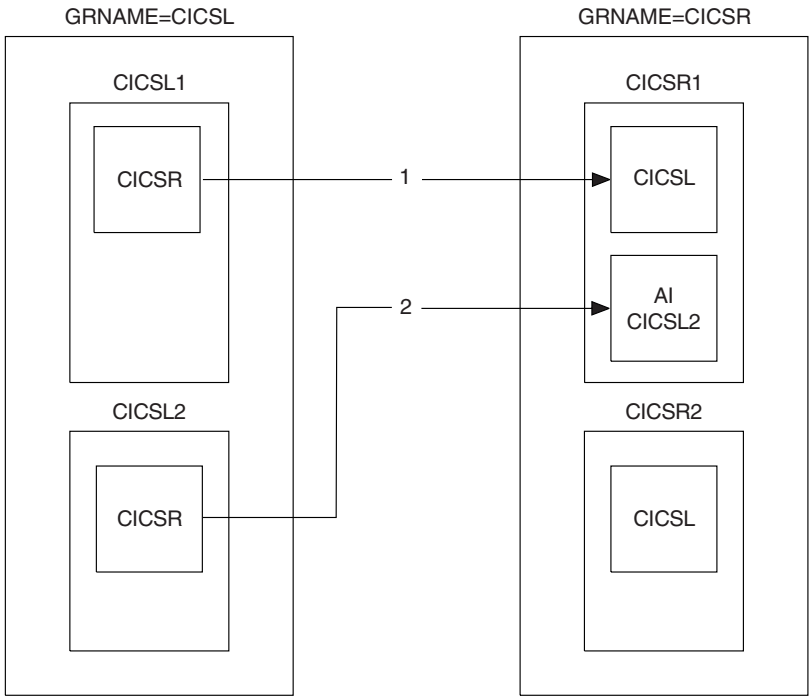


Figure 35. Second flow, CICSL2-CICSR

Figure 35 shows a bind flow from CICSL2 to CICSR. In this example VTAM has, once again, chosen to route it to CICSR1, but it could have gone to one of the other members of CICSR.

The predefined connection for CICSR in CICSL2 is used. CICSR1 looks for the connection entry for CICSL. It is already in use, so a new connection is autoinstalled using the member name CICSL2.

Affinities are created at SYSPLEXL and SYSPLEXR, associating CICSL2 with CICSR1. If you need to end these affinities, you may or may not need to do so explicitly.

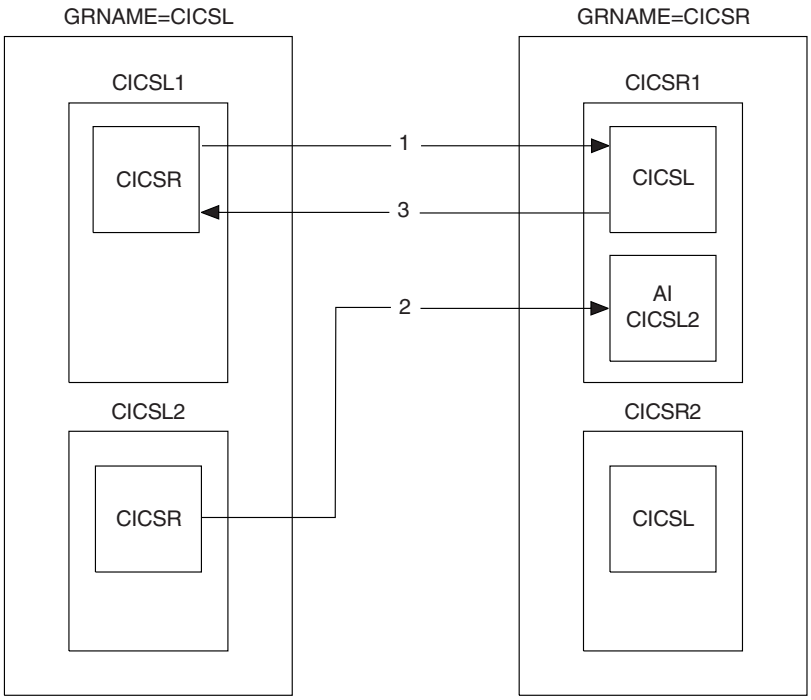


Figure 36. Third flow, CICSR1-CICSL

Figure 36 shows a third flow, this time from CICSR1 to CICSL. The existing affinity forces it to CICSL1.

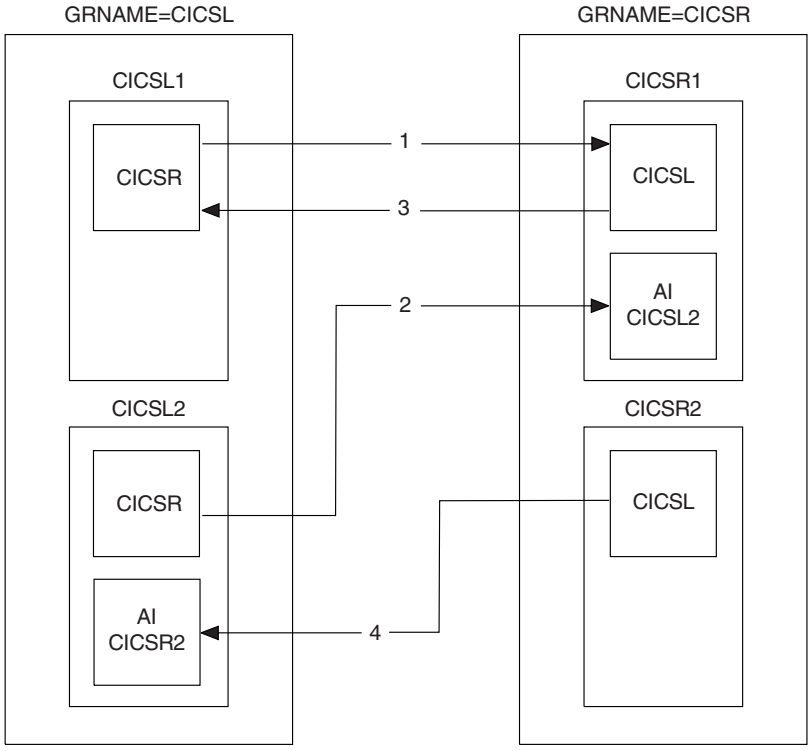


Figure 37. Fourth flow, CICSR2-CICSL

Figure 37 on page 130 shows a fourth flow, this time from CICSR2 to CICSL. It can go to any member of CICSL, but in this example VTAM routes it to CICSL2.

The predefined connection entry for CICSL in CICSR2 is not in use and so it is used now. CICSL2 looks for the predefined connection entry for CICSR. It is in use, and so an entry for CICSR2 is autoinstalled.

Affinities are created at SYSPLEXL and SYSPLEXR, associating CICSL2 with CICSR2. If you need to end these affinities, you may or may not need to do so explicitly.

Ending affinities

When a session is established with a member of a generic resource, VTAM creates an association called an affinity between the generic resource member and the partner LU, so that it knows where to route subsequent flows. In most cases, VTAM ends the affinity when all activity on the session has ceased. However, for some types of session, VTAM assumes that resynchronization data may be present, and therefore relies on CICS to end the affinity. The sessions affected are:

- APPC synclevel 2 sessions
- APPC sessions using limited resource support
- LU6.1 sessions.

In VTAM terms, the CICS generic resource member “owns” the affinity and is responsible for ending it. The affinity persists even after a connection has been deleted or CICS has performed an initial or cold start. *For a connection between two generic resources, both partners own an affinity, and each must be ended*. For APPC connections between CICS TS OS/390, Version 1.3 or later regions, the APPC connection quiesce protocol does this automatically—see “APPC connection quiesce processing” on page 292. For other connections, the affinities must be ended explicitly.

CICS provides commands that can be used to end affinities explicitly:

- You can use SET CONNECTION ENDAFFINITY when there is an installed connection definition.
- You can use PERFORM ENDAFFINITY after an autoinstalled connection has been deleted, as well as when it is still present. You must supply the NETNAME (and, if the connection has been deleted, the NETID) of the remote system. The NETNAME is the name by which the remote system is known to VTAM. (Note that, if the remote system is also a generic resource, the NETNAME is always the member name, even if the connection was defined using the generic resource name.)

These commands are valid only for LU6.1 and APPC connections. The connection, if present, must be out of service and its recovery status (as shown by the RECOVSTATUS option of the INQUIRE CONNECTION command) must be NORECOVDATA. Note that only those affinities that are owned by CICS can be ended by CICS.

CICS has no certain knowledge that an affinity exists for a given connection. To help you, message DFHZC0177 is issued whenever there is a possibility that an affinity has been created that you may have to end explicitly. This message gives the NETNAME and NETID to be used on the PERFORM ENDAFFINITY command.

Having received message DFHZC0177, to check whether an affinity that must be ended explicitly does indeed exist, you can use the SNA `D NET,GRAFFIN` command. This command produces messages IST1706 and IST1707, which should contain the information you need. Alternatively, the *MVS/ESA Version 5 Interactive Problem Control System (IPCS) Commands* manual, GC28-1491, tells you how to produce a dump of the VTAM ISTGENERIC data area. This contains SPTE records that show which affinities exist. For example, start the dump with:

```
DUMP COMM=(title)
```

Reply with:

```
r xx ,STRLIST=(STRNAME=ISTGENERIC,
                  ACC=NOLIMIT,(LNUM=ALL,ADJ=CAP,EDATA=SER))
```

Look at the dump with:

```
STRDATA DETAIL ALLSTRS ALLDATA
```

If a request to end an affinity is rejected by VTAM because no such affinity exists, message DFHZC0181 is issued. This may mean either that you supplied an incorrect NETNAME or NETID, or that you (or CICS) were wrong in supposing that an affinity existed.

When should you end affinities?

You need to end affinities if you reconfigure your sysplex. For example, you **must** end any relevant affinities before you do any of the following:

- Change the name of a generic resource.
- Change a generic resource name connection to a member-name connection.
- Change a parallel-session connection to a single-session connection.
- Remove systems from a generic resource. If you remove a system from a generic resource and do not end its affinities, VTAM treats it as though it were still a member of the generic resource.

Note: For connections between generic resources, you must end the affinities owned by both generic resources.

Writing a batch program to end affinities

If a generic resource member that owns affinities fails and cannot be recovered, the affinities must be ended. In a case like this, you cannot use the SET CONNECTION ENDAFFINITY or PERFORM ENDAFFINITY commands. Instead, you can use a batch program to clear the affinities owned by the failed member. This section demonstrates how to write such a batch program. The program must be written in assembler language.

Note: You can use the dump technique described in the *MVS/ESA Version 5 Interactive Problem Control System (IPCS) Commands* manual to discover what affinities the failed generic resource member owns.

Important:

You should use this technique only if it is impossible to restart the failed CICS system.

Program input

The following input parameters are needed:

- Member name (in the generic resource group) of the failed system
- Generic resource name of the failed system
- APPLID of the partner system
- NETID of the partner system.

Program output

The program uses the VTAM CHANGE OPTCD=ENDAFFIN macro to end the affinities. You will probably need to produce a report on the success or failure of this and the other VTAM macro calls that the program uses. Consult the *OS/390 eNetwork Communications Server: SNA Programming* manual for the meaning of RTNCD/FDB2 values.

Processing

1. Reserve storage for the following:
   - The ACB of the failed sysplex member:

     ```
     acb-name ACB AM=VTAM,
                  PARMS=(PERSIST=YES)
     ```

     Note that the above example assumes that you are using persistent sessions.
   - The RPL, which is required by the VTAM macros:

     ```
     rpl-name RPL   AM=VTAM,OPTCD=(SYN)
     ```
   - The NIB, which is required by the CHANGE OPTCD=ENDAFFIN macro:

     ```
     nib-name NIB
     ```
2. Issue a VTAM OPEN command for the ACB of the member which owns the affinity, passing the input APPLID for this member.
3. If any sessions persist, use the VTAM SENDCMD macro to terminate them. (If you are not using persistent sessions this will not be necessary.)
   a. Move the following command to an area in storage. In this example, *applid1* is the member name of the failed member and *applid2* is the APPLID of the partner system.

      ```
      'VARY NET,TERM,LU1=applid1,LU2=applid2,TYPE=FORCE,SCOPE=ALL'
      ```
   b. Issue the SENDCMD macro, as in the example below. In this example:
      - *rpl-name* is the name of an RPL.
      - *acb-name* is the ACB of the failed sysplex member.
      - *output-area* is the name an area in storage where the VARY command is held.
      - *command-length* is the length of the command.

      ```
      SENDCMD  RPL=rpl-name,
         ACB=acb-name,
         AREA=output-area,
         RECLEN=command-length,
         OPTCD=(SYN)
      ```
4. Use the VTAM RCVCMD macro to receive messages from VTAM. Note that RCVCMD must be issued three times after the SENDCMD to be sure that the VARY command worked correctly. In the following example:
   - *rpl-name* and *acb-name* are as described above.
   - *input-area* is the area of storage into which the message is to be received.
   - *receive_length* is the length of data to be received.

```
RCVCMD  RPL=rpl-name,
   ACB=acb-name,
   AREA=input-area,
   AREALEN=receive-length,
   OPTCD=(SYN,TRUNC)
```

5. Issue this command twice more to make sure of receiving all the output from VTAM.
6. Issue the VTAM CHANGE OPTCD=ENDAFFIN macro to end the affinity. Before issuing the macro the following fields must be initialized in the NIB:
   - NIBSYM is set to the APPLID of the partner system.
   - NIBGENN is set to the generic resource name of the failed system.
   - NIBNET is set to the NETID of the partner system.

   ```
   CHANGE  RPL=rpl-name,
           ACB=acb-name,
           NIB=nib-name,
           OPTCD=(SYN,ENDAFFIN)
   ```
7. Issue the VTAM CLOSE command for the ACB.

Programming notes:

1. The VTAM commands should be synchronous, to avoid the use of exits (OPTCD=SYN).
2. Care must be taken **not** to run the program for an APPLID of a running CICS. If you do, and you are using VTAM persistent sessions, a *predatory takeover* will occur—that is, your program will assume control of the sessions belonging to the APPLID.

JCL for submitting the ENDAFFINITY program

```
//JOBNAME   JOB 1,userid,
// NOTIFY=userid,CLASS=n,MSGLEVEL=(n,n),MSGCLASS=n,REGION=1024K
//*
//JOBLIB    DD DSN=loadlib-name,DISP=SHR
//*
//*****************************************************************
//* PARM='FAILED_APPLID,FAILED_GENERIC,PARTNER_NETID,PARTNER_APPLID'
//*****************************************************************
//*
//RUN    EXEC PGM=ENDAFFIN,PARM='parm1,parm2,parm3,parm4'
//*
//REPORT    DD SYSOUT=*
//SYSPRINT  DD SYSOUT=*
//
```

Figure 38. Example JCL for submitting the ENDAFFINITY program

Using ATI with generic resources

Automatic transaction initiation (ATI) is the process whereby a transaction is started by a request made internally within the CICS system, rather than by a terminal end-user entering a transaction name. This can happen when, for example, an application program issues an EXEC CICS START command, or the trigger level on a transient data queue is reached. Often the started transaction is associated with a terminal, which may or may not be owned by the region in which the transaction runs.

ATI is described in “Traditional routing of transactions started by ATI” on page 59. In particular, “Traditional routing of transactions started by ATI” on page 59 describes

how CICS invokes the “terminal not known” global user exits, XICTENF and XALTENF, to deal with the situation where the terminal is not defined to the AOR.

When an automatic transaction initiation (ATI) request is issued in an application-owning region (AOR) for a terminal that is logged on to a TOR, CICS uses the terminal definition in the AOR to determine the TOR to which the request should be shipped. If there is no definition of the terminal in the AOR, you may be able to use the “terminal-not-known” global user exits (XICTENF and XALTENF) to supply the name of the TOR.

However, if a user logs on to a generic resource (using a generic resource name), VTAM may connect his or her terminal to any of the regions in the generic resource. If the user then logs off and on again, VTAM may connect his terminal to the same region, or to a different one. In this situation, the terminal definition in the AOR may not reflect the correct location of the terminal; and your terminal-not-known exit program has no way of knowing the correct destination for the ATI request.

CICS solves this problem by using VTAM's knowledge of where the terminal is logged on, to ship the ATI request to the correct TOR:

1. First, the ATI request is shipped to the TOR specified in the remote terminal definition (or specified by the terminal-not-known exit)—we shall call this the “first-choice TOR”. If the terminal is logged on to the first-choice TOR, the ATI request completes as normal.
2. If the terminal cannot be located on the first-choice TOR, the TOR asks VTAM for the applid of the generic resource member where the terminal is logged on. If the terminal is not logged on to any applid within the generic resource group, the ATI request fails.

   If the terminal is located on the first-choice TOR but not logged on, the TOR asks VTAM for the applid of the generic resource member where the terminal is logged on. If the terminal is not logged on to any applid within the generic resource group, the ATI request is scheduled on the first-choice TOR. If the terminal is logged on to a different applid within the generic resource group, this information is passed to the AOR, and the ATI request is shipped to the correct TOR.
3. If the first-choice TOR is not available (and such an inquiry is possible) the AOR asks VTAM for the location of the terminal. The inquiry is possible when all of the following are true:
   - The VTAM in the AOR is version 4.2 or later (that is, it supports generic resources).
   - The AOR was started with the VTAM system initialization parameter set to 'YES'.
   - The VTAM generic resource name where the terminal may be logged on is known to the AOR. Such information is obtained from the skeleton TCTTE representing the remote terminal. If the first choice TOR name has been supplied by the user terminal-not-known exit, such an inquiry is not possible. Note that the inquiry will fail if the terminal is not logged on to the VTAM generic resource name found in the skeleton TCTTE.

   If the AOR is in one network and the TORs in another, the inquiry fails.

   If the inquiry is successful, the ATI request is shipped to the TOR where the terminal is logged on.

VTAM knows the terminal by its netname, not by its CICS terminal identifier (TERMID). If there is a terminal definition in the AOR at the time the START is

issued, CICS obtains the netname from that definition. If there is not, your terminal-not-known exit program should return:

- A netname that VTAM can use to locate the terminal
- The name of a connection to any member of the generic resource that is likely to be active.

Note:

1. If CICS has no netname for the terminal, the ATI request is shipped to the first-choice TOR, and the termid is used to locate the terminal. If the terminal cannot be found on the first-choice TOR, the ATI request fails.
2. Because CICS uses the terminal's netname to find its location in the generic resource group, the ATI request will still work if, on the second or subsequent logon, the termid changes (for instance, if the autoinstall user program does not implement a consistent mapping between netname and termid).
3. The ATI support described in this section applies only to terminals that use the generic name to log on to a generic resource. If a user logs on to a TOR using the member name, CICS does not attempt to discover from VTAM to which TOR the terminal is connected.
4. The ATI support described in this section does not apply to ATI to an APPC connection.
5. The TORs can use autoinstall or CEDA-defined terminal definitions.

   The AORs must **not** use CEDA-defined remote terminal definitions. If CEDA-defined terminals are used, the ATI request will always be shipped to the first-choice TOR and will not be re-routed to a different TOR within the same VTAM generic resource group, even though the terminal may be logged on to another TOR.

Example 1:

1. A user logs on using the generic resource name CICS, which is the name of a set of TORs (TOR1 through TOR6). She is connected to TOR1, because it is the most lightly loaded.
2. The user runs a transaction, which is routed to an AOR, AOR1. The terminal definition is shipped to AOR1.
3. The transaction issues an EXEC CICS START request, to start another transaction, after an interval, against the same terminal. The second transaction, like the first, is located on AOR1.
4. After the first transaction has completed, the user logs off; and logs on again later to collect the output from the second transaction. When logging on the second time, again using the generic resource name CICS, the user is connected to TOR2 because that is now the most lightly loaded.
5. The interval specified on the START request expires. However, the terminal is no longer defined to TOR1. The shipped terminal definition has not yet been deleted from AOR1 by the timeout delete mechanism.

- **Result:**

  Because the shipped definition of the user's terminal still exists on AOR1, AOR1 ships the ATI request to TOR1 (the TOR referenced in the definition). Because the terminal is not logged on to TOR1, TOR1 queries VTAM and returns the result to AOR1. AOR1 then ships the request to the correct TOR (TOR2).

Example 2:

1. A user logs on using the generic resource name CICS, which is the name of a set of TORs (TOR1 through TOR6). She is connected to TOR1, because it is the most lightly loaded.
2. The user runs a transaction, which is routed to an AOR, AOR1. The terminal definition is shipped to AOR1.
3. The transaction does some asynchronous processing—that is, it starts a second transaction, which happens to be on another AOR, AOR2. After it has finished processing, the second transaction is to reinvoke the original transaction to send a message to the user-terminal at TOR1.
4. The user logs off while the application is in process, and logs on again later to collect the message. When logging on the second time, again using the generic resource name CICS, the user is connected to TOR2 because that is now the most lightly loaded.
5. The second transaction completes its processing, and issues an EXEC CICS START command to reinvoke the original transaction, in conjunction with the original terminal. The START request is shipped to AOR1. However, the terminal is no longer defined to TOR1, and the shipped terminal definition has been deleted from AOR1 by the timeout delete mechanism.

- **Result:**

  Because the shipped terminal definition has been deleted from AOR1, CICS invokes the XICTENF and XALTENF exits. Your exit program should return:

  – The netname of the user's terminal
  – The name of a connection to any member of the generic resource that is likely to be currently active.

  CICS is then able to query VTAM, as described in Example 1, and ship the request to the correct TOR (TOR2).

Using the ISSUE PASS command

The EXEC CICS ISSUE PASS command can be used to disconnect a terminal from CICS and transfer it to the VTAM application specified on the LUNAME option. For example, to transfer a terminal from this CICS to another terminal-owning region, you could issue the command:

```
EXEC CICS ISSUE PASS
LUNAME(applid)
```

where `applid` is the applid of the TOR to which the terminal is to be transferred.

When your TORs are members of a generic resource group, you can transfer a terminal to any member of the group by specifying LUNAME as the generic resource name. For example:

```
EXEC CICS ISSUE PASS LUNAME(grname)
```

where `grname` is the generic resource name. VTAM transfers the terminal to the most lightly-loaded member of the generic resource. (If the system that issues the ISSUE PASS command is itself the most lightly-loaded member, VTAM transfers the terminal to the next most lightly-loaded member.)

Note that, if the system that issues an ISSUE PASS LUNAME(grname) command is the *only* CICS currently registered under the generic resource name (for example, the others have all been shut down), the ISSUE PASS command does **not** fail with an INVREQ. Instead, the terminal is logged off and message DFHZC3490 is written to the CSNE log. You can code your node error program to deal with this situation. For advice on coding a node error program, see Writing a node error program, in the *CICS Customization Guide*.

If you need to transfer a terminal to a specific TOR within the CICS generic resource group, you must specify LUNAME as the member name—that is, the CICS APPLID, as in the first example command.

Rules checklist

Here is a checklist of the rules that govern CICS use of the VTAM generic resources function:

- Generic resource names must be unique in the network.
- A CICS region cannot be both a member of a generic resource and an XRF partner.
- A CICS region that is a member of a generic resource can have only one generic resource name and only one applid.
- A generic resource name cannot be the same as a VTAM applid in the network.
- Within a generic resource, member names only must be used. There must be no definitions in any of the members of the generic resource for the generic resource name.
- Non-LU6 devices that require sequence number resynchronization cannot log on using the generic resource name. They must use the applid and therefore cannot take advantage of session balancing.
- APPC connections to a generic resource that are initiated by the partner (that is, on which the non-generic resource sends the first bind) can log on using a member name.
- For LU6.1 connections initiated by a generic resource member, the partner must know the member by its generic resource name.

  Therefore, you are strongly recommended not to try to access the same LU6.1 partner from more than one member of a generic resource.
- For APPC connections initiated by a generic resource member, where the partner is not itself a member of a CICS Transaction Server for z/OS generic resource, the partner must know the member TOR by its generic resource name.

  Therefore, you are strongly recommended not to try to access such partners from more than one member of a generic resource.
- A system cannot statically define both an APPC generic resource name connection and an APPC member name connection to the same generic resource. (Generic resource name connections and member name connections are described in “Establishing connections between CICS TS for z/OS generic resources” on page 126.)

  Furthermore, all members of a generic resource must choose the same method. That is (for statically-defined APPC connections to a partner generic resource), they must all use member name connections or all use generic resource name connections.

Dealing with special cases

This section describes some special cases that you may need to consider. **Note that much of the information applies only to links to back-level systems—where, for example, you are initiating a connection to a non-CICS TS for z/OS system. For connections between CICS TS for z/OS generic resources, much of the following information can be disregarded**.

Non-autoinstalled terminals and connections

Important:

> Because members of a generic resource should be functionally equivalent, it is not recommended that you should predefine terminals to specific members of a generic resource. Use autoinstall instead, and allow VTAM to balance the TORs' workload dynamically. However, there may be times—for example, while you are migrating an existing TOR into a generic resource—when it is necessary to use static definitions.

If an LU is predefined to a specific terminal-owning region, and the LU initiates the connection (that is, it sends the first bind request) using the TOR's generic resource name, the generic resource function must make the connection to the “correct” terminal-owning region—the one that has the definition. This requirement means that you must install the VTAM generic resource resolution exit program, ISTEXCGR, to enforce selection of the correct applid (for the terminal-owning region).

Note that this is not necessary if the connection is always initiated by the terminal-owning region (by means, for example, of a START request).

A sample ISTEXCGR exit program is supplied with VTAM 4.2. For details, see the *OS/390 eNetwork Communications Server: SNA Customization* manual.

Outbound LU6 connections

This section discusses outbound LU6 connections from TORs that are members of a generic resource group. By “outbound” we mean connections to systems outside the CICSplex.

Using a “hub”

For LU6 connections initiated by a generic resource member, where the partner is not itself a CICS Transaction Server for z/OS generic resource, the partner must know the member TOR by its generic resource name.

The requirement therefore applies when a generic resource member initiates any of the following kinds of connection:

- APPC connections to single systems
- APPC connections to members of a CICSplex that are not also generic resource members
- All LU6.1 connections.

Because (unless the partner is also a CICS TS for z/OS generic resource) an attempt by a generic resource member to connect to an LU6 partner will succeed only if the partner knows the TOR by its generic resource name, it follows that the

partner can accept a connection to only one member of the generic resource at a time. In a configuration in which more than one member of a generic resource must connect to a remote system, you can choose a region within the CICSplex to act as a **network hub**. This means that all generic resource members daisy-chain their requests for services from remote systems through the hub.

The network hub can be a member of the generic resource, in which case it is necessary to install a VTAM generic resource resolution exit program to direct any *incoming* binds from LU6 partners that know us by our generic resource name to the network hub region.

An alternative solution is to have a network hub that is **not** a member of the generic resource. This avoids the need for the VTAM generic resource resolution exit program, but requires that LU6 partners that may initiate connections to the CICSplex log on using the applid of the network hub region.

Figure 39 shows a network hub that is not a member of the generic resource.

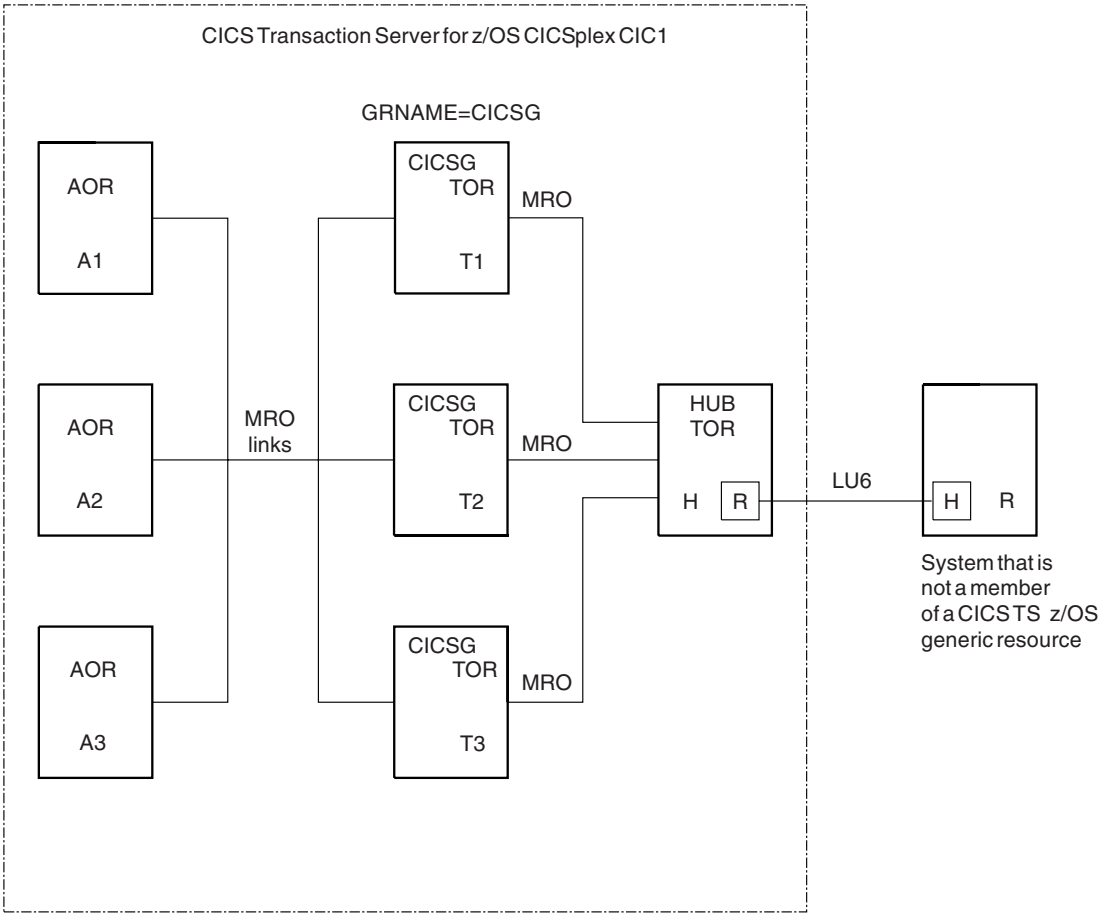


Figure 39. A network hub. Hubs are typically used for outbound LU6 requests from members of a generic resource group to a system that is not a member of a CICS Transaction Server for z/OS generic resource.

In Figure 39, the regions in CICSplex CIC1 are connected by MRO links. The terminal-owning regions T1, T2, and T3 are members of the generic resource group, CICSG, but the hub TOR, H, is not. H has an LU6.1 or APPC connection to the remote region, R. The TORs daisy-chain their requests to R through H.

Part 3. Defining intercommunication resources

This part of the manual tells you how to define the various resources that may be required in a CICS intercommunication environment. CICS resources are defined using resource definition online (RDO). This part tells you how to:

- Define links to remote systems. The links described are:
  - MRO links to other CICS regions
  - MRO links for use by the external CICS interface
  - Multi-session APPC links to other APPC systems (CICS or non-CICS)
  - Single-session APPC links to APPC terminals
  - LUTYPE6.1 links to IMS systems.
- Manage APPC links using the master terminal transaction (CEMT).
- Define remote resources to the local CICS system. The resources can be:
  - Remote files
  - Remote DL/I PSBs
  - Remote transient-data queues
  - Remote temporary-storage queues
  - Remote terminals
  - Remote APPC connections
  - Remote programs
  - Remote transactions.
- Define local resources for ISC and MRO. In general, these resources are those that are required for ISC and MRO and are obtained by including the relevant functional groups in the appropriate tables. However, you have the opportunity to modify some of the supplied definitions and to provide your own communication profiles.

Chapter 13. Defining links to remote systems

This chapter tells you how to define and manage communication links ("connections") to other CICS regions or to other, non-CICS systems.

The types of link described are:

- Links for multiregion operation (MRO)
- Links for use by the external CICS interface (EXCI)
- IPIC links to remote CICS TS for z/OS, Version 3.2 (or later) regions, for use with distributed program link
- ISC over SNA links to remote systems, using logical unit type 6.2 (APPC) protocols
- ISC over SNA links to remote IMS 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

MRO and ISC over SNA (APPC and LUTYPE 6.1) connections are defined differently from IPIC connections.

MRO and ISC over SNA connections

The definition of an MRO or ISC over SNA connection to a remote system consists of two 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 DEFINE CONNECTION command. Each session, or group of parallel sessions, is defined by the 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.

IPIC connections

IPIC connections ("IPCONN".)

To create an IPIC connection, you must install two, complementary, resources:

1. An IPCONN definition specifies the *outbound* attributes of the TCP/IP connection.
2. A TCPIPSERVICE definition specifies the *inbound* attributes of the connection. The TCPIPSERVICE to be used is specified by the TCPIPSERVICE option of the IPCONN definition.

For more information, see "Defining IP interconnectivity links" on page 151.

Naming the local CICS system

A CICS Transaction Server for z/OS system can be known by more than one name:

- Application identifier (applid)
- System identifier (sysidnt)
- VTAM generic resource name

All CICS systems have an applid and a sysidnt. A terminal-owning region that is a member of a VTAM generic resource group also has a VTAM generic resource name (VTAM generic resource names are described in Chapter 12, "Installation considerations for VTAM generic resources," on page 119).

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 over SNA, the applid is used on a VTAM APPL statement, to identify CICS to VTAM.

For IPIC, the APPLID option of an IPCONN definition identifies the applid of the remote system.

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

Within a z/OS sysplex, the applid of each CICS region must be unique. If your CICS regions are not part of a sysplex, if your network consists of more than one sysplex, or if your CICS regions communicate with systems outside the local sysplex, it is advisable to keep applids unique across the network, if this is possible. If your network does contain systems with identical applids, on IPIC connections you can specify the NETWORKID option; this unique value enables you to connect to two or more remote systems that have identical applids.

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 175.

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 16, “Defining remote resources,” on page 191.) 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, for an MRO or ISC over SNA connection, on the CONNECTION definition you would specify:

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

For an IPIC connection, on the IPCONN definition you would specify:

```
DEFINE IPCONN(sysidnt)  The local name for the remote system
       APPLID(applid)  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 "Defining links for use by the external CICS interface" on page 149. 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 "Defining links for use by the external CICS interface" on page 149.

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.

Defining an MRO link

The definition for an MRO link is shown in Figure 40 on page 147.

Note: For reasons of clarity and conciseness, inapplicable and inessential options have been omitted from Figure 40 on page 147, 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 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|XM)
  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 40. 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 24, “Intersystem session queue management,” on page 267.

For information about the ATTACHSEC and USEDFLTUSER security options see Specifying user security in link definitions, in the *CICS RACF Security Guide*.

On the SESSIONS definition, you must specify the number of SEND and RECEIVE sessions that are required (at least one of each). Initial requests can never be sent on a RECEIVE session. Bear this in mind when deciding how many SEND and RECEIVE sessions you need.

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.

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

Choosing the access method for MRO

You can specify ACCESSMETHOD(XM) to select MVS cross-memory services for an MRO link. Cross-memory services are used only if the other end of the link also specifies cross-memory. To select the CICS Type 3 SVC for interregion communication, use ACCESSMETHOD(IRC).

When you specify ACCESSMETHOD(XM) in a connection definition, a region containing this definition uses one of the 512 available MRO XM logons for the LPAR. A region can contain both ACCESSMETHOD(XM) and ACCESSMETHOD(IRC) connections, but if the region contains one or more XM connections then the region uses an MRO XM logon.

The use of MVS cross-memory services reduces the number of instructions necessary to transmit messages between regions. Also, less virtual storage is required in the MVS common service area. However, cross-memory services can be less attractive from the security point of view (see Security implications of choice of MRO access method , in the *CICS RACF Security Guide*).

Cross-memory services also require CICS address spaces to be nonswappable. For low-activity systems that would otherwise be eligible for address space swapping, you might prefer to accept the greater path length of the CICS interregion SVC rather than the greater real storage requirements of nonswappable address spaces.

Note: If you are using cross-system multiregion operation (XCF/MRO), CICS selects the XCF access method dynamically—overriding the CONNECTION definition, which can specify either XM or IRC.

Figure 41 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(XM)      cross-memory services
  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 41. 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 42 on page 149.

```
       CICSA                                CICSB

DFHSIT  TYPE=CSECT                       DFHSIT  TYPE=CSECT
       ,APPLID=CICSA       1           4        ,APPLID=CICSB

DEFINE CONNECTION(CICB)    2           8   DEFINE CONNECTION(CICA)
    GROUP(PRODSYS)         3           9       GROUP(TESTSYS)
    NETNAME(CICSB)         4           1       NETNAME(CICSA)
    ACCESSMETHOD(IRC)                          ACCESSMEHOD(IRC)
    QUEUELIMIT(500)                            QUEUELIMIT(NO)
    MAXQTIME(500)
    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 42. Defining compatible MRO nodes

In Figure 42, related options are shown by identical numbers.

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). The external CICS interface does not support the MRO cross-memory access method (XM). The cross-system coupling facility (XCF) *is* supported.
- Let SENDCOUNT and SENDPFX default to blanks.

Figure 43 on page 150 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 43. 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 44 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 44. 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 can install *new* MRO and EXCI connections dynamically, while CICS is fully operational—there is no need to close down interregion communication (IRC) to do so. Note that CICS commits the installation of connection definitions at the group level—if the install of any connection or terminal fails, CICS backs out the installation of all connections in the group. Therefore, when adding new connections to a CICS region with IRC open, ensure that the new connections are in a group of their own.

You cannot modify *existing* MRO (or EXCI) links while IRC is open. You should therefore ensure, when defining an MRO link, that you specify enough SEND and RECEIVE sessions to cater for the expected workload.

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

Defining IP interconnectivity links

This section describes how to define connections to remote CICS TS for z/OS, Version 3.2 (or later) regions, for use with distributed program link (DPL) using IP interconnectivity (IPIC)

Note: DPL programs can communicate over MRO, APPC, or IPIC connections. For statically-routed DPL, the target region is identified by the REMOTESYSTEM name on the PROGRAM definition. REMOTESYSTEM can specify the name of an IPCONN definition or a CONNECTION definition.

To create an IPIC connection, you must install two, complementary, resources:

1. An IPCONN definition specifies the *outbound* attributes of the TCP/IP connection.
2. A TCPIPSERVICE definition specifies the *inbound* attributes of the connection. The TCPIPSERVICE to be used is specified by the TCPIPSERVICE option of the IPCONN definition.

The most significant attributes of the IPCONN definition are:

APPLID(*applid*)
Specify the application identifier (applid) of the remote system, as defined on the **APPLID** parameter of its system initialization table (SIT).

There are some rules about duplicate APPLIDs:

- You cannot install two or more IPCONN definitions that specify the same APPLID *and* the same NETWORKID. (The combination of APPLID and NETWORKID can be used to ensure unique naming of systems across the network. See the description of the NETWORKID option, below.)
- You *can* install an IPCONN definition that specifies the same APPLID as the NETNAME of an installed MRO, APPC, or LUTYPE6.1 CONNECTION definition.
- If an installed IPCONN definition has the same name as an installed CONNECTION definition, the APPLID of the IPCONN definition must match the NETNAME of the CONNECTION definition. If they do not, the message that results depends on the situation:
  - DFHIS3009 if the error is detected during IPCONN autoinstall
  - DFHAM4913 if the error is detected during IPCONN install
  - DFHZC6312 if the error is detected during CONNECTION install or autoinstall

  The IPCONN definition takes precedence over the CONNECTION definition: that is, if an IPCONN and a CONNECTION have the same name, CICS uses the IPCONN.
- a CONNECTION and an IPCONN with the same NETNAME and APPLID do not have to have the same name.

This allows the possibility to use a distinct sysid for communication over TCP/IP rather than relying on the CICS default of routing all supported function via the IPCONN, if it exists.

The previous rules are validated at install time.

AUTOCONNECT({**NO**|**YES**})
Specify **YES** if you want sessions to be established when the IPCONN definition is installed, **NO** if not. If you specify **YES**, bear in mind that, for connectivity to be achieved when you install the IPCONN definition:

1. The TCPIPSERVICE definition named on the TCPIPSERVICE option of this IPCONN definition must also be installed in this region and must specify PROTOCOL(IPIC).
2. Corresponding IPCONN and TCPIPSERVICE definitions must be installed in the remote region. “Corresponding” means that:
   a. The HOST option of the IPCONN definition on the remote region must specify this region.
   b. The PORT option of the IPCONN definition on the remote region must specify the same port number as that specified on the PORTNUMBER option of the local TCPIPSERVICE definition named by this IPCONN.
   c. The TCPIPSERVICE definition on the remote region (named by the IPCONN definition on the remote region) must specify PROTOCOL(IPIC) and, on its PORTNUMBER option, the same port number as that specified by the PORT option of this IPCONN.

HOST(*hostname*)
Specify the host name or dotted-decimal IP address of the target system: for example, `abc.example.com`, or `9.20.183.7`.

INSERVICE({**NO**|**YES**})
Specify **YES** to make the connection available for use when it is installed, or **NO** to make it unavailable.

IPCONN(*name*)
The eight-character name of this IPCONN definition. Specify the 4-character “local name” (SYSID) by which CICS knows the remote system, padded with four trailing blanks.

NETWORKID(*name*)
Specify the network ID of the remote system. (The remote system’s network ID is either its VTAM NETID or, for VTAM=NO systems, the value of its **UOWNETQL** system initialization parameter.)

If NETWORKID is not specified, CICS assumes that the remote system is in the same network as the local system. In this instance, CICS uses the VTAM NETID, or the value of the UOWNETQL system initialization parameter, of this CICS (that is, the CICS on which this definition is installed).

Specify NETWORKID if you want to connect to a remote system that is in a different network, and so has a different VTAM NETID or UOWNETQL value. In this instance, it could be possible for two or more remote systems to have the same APPLID. (Although CICS APPLIDs must be unique within a sysplex, you may, for example, want to connect to a system outside the sysplex or in a different sysplex.) The combination of APPLID and NETWORKID ensures that the remote system is referred to by a unique name.

PORT(*port*)
Specify, in the range 1 through 65535, the decimal number of the port that the

remote region will listen on. The port number is combined with the HOST value to determine the destination for outbound requests on this IPCONN.

RECEIVECOUNT(*number*)
Specify, in the range 1-999, the number of receive sessions; that is, sessions that receive incoming requests. The actual number of receive sessions that are used depends also on the number of send sessions defined in the remote system. When the connection is established, these values are exchanged and the lower value is used.

SENDCOUNT(*number*)
Specify, in the range 1-999, the number of send sessions; that is, sessions that send outgoing requests. The actual number of send sessions that are used depends also on the number of receive sessions defined in the remote system. When the connection is established, these values are exchanged and the lower value is used.

TCPIPSERVICE(*name*)
Specify the name of the TCPIPSERVICE definition, with PROTOCOL(IPIC), that defines the attributes of the inbound processing for this IPCONN.

Figure 45 shows a typical IPCONN definition.

```
DEFINE
  IPCONN(IP01)                    local name for connection
  GROUP(groupname)                groupname of related definitions
  HOST(aor001.widgets.com)        host name of remote system
  PORT(32022)                     port number that remote region will
                                  listen on
  TCPIPSERVICE(IP01tcps)          name of a PROTOCOL(IPIC) TCPIPSERVICE that
                                  defines the attributes of inbound requests
  APPLID(AOR001)                  SIT APPLID of the remote system
  INSERVICE(YES)
  RECEIVECOUNT(20)                the number of receive sessions
  SENDCOUNT(20)                   the number of send sessions
```

Figure 45. Example IPCONN definition for an IPIC connection

For detailed explanations of all the possible attributes of IPCONN definitions, see the *CICS Resource Definition Guide*.

For DPL over IPIC links, the most significant attributes of the TCPIPSERVICE definition are:

IPADDRESS({**INADDR_ANY**|*ipaddress*})
Specify the dotted decimal IP address on which this TCPIPSERVICE will listen for incoming connections. It must be of the form nnn.nnn.nnn.nnn where nnn is 0 through 255. This is the IP address that must be specified on the HOST option of the IPCONN definition on the remote system.

PORTNUMBER(*port*)
Specify, in the range 1 through 65535, the decimal number of the port on which CICS is to listen for incoming client requests. This is the number that must be specified on the PORT option of the IPCONN definition on the remote system.

PROTOCOL({**ECI**|**HTTP**|**IIOP**|**IPIC**|**USER**})
The application level protocol used on the TCP/IP port. For IPIC links, specify IPIC.

SOCKETCLOSE({**NO**|*hhmmss*})
If, and for how long, CICS should wait before closing the socket, after issuing a receive for incoming data on that socket. For IPIC links, you must specify SOCKETCLOSE(NO).

TCPIPSERVICE(*name*)
The 8-character name of this service. This is the name that must be specified on the TCPIPSERVICE option of the corresponding IPCONN definition.

TRANSACTION(*transaction*)
The 4-character ID of the CICS transaction attached to process new requests received for this service. For IPIC, specify CISS (or another transaction that executes program DFHISCOP).

URM(*program*)
Specify the name of the IPCONN autoinstall user program, if required. If you do not specify this attribute CICS uses the CICS-supplied, default, IPCONN autoinstall user program, DFHISAIP.

Figure 46 shows a typical TCPIPSERVICE definition of an IPIC connection.

```
DEFINE
  TCPIPSERVICE(IP01tcps)          name of the TCPIPSERVICE to receive
                                  inbound requests (same as that
                                  specified on the IPCONN definition)
  GROUP(groupname)                groupname of related definitions
  IPADDRESS(9.20.185.7)           IP address we will listen at
  PORTNUMBER(1500)                port number we will listen on
  PROTOCOL(IPIC)                  must be IPIC for IP interconnectivity links
  SOCKETCLOSE(NO)                 must be NO for IP interconnectivity links
  TRANSACTION(CISS)               transaction to process new requests
  URM(DFHISAIP)                   IPCONN autoinstall user program
```

Figure 46. Example TCPIPSERVICE definition for an IPIC connection

The following figure shows the relationship between IPCONN and TCPIPSERVICE definitions.

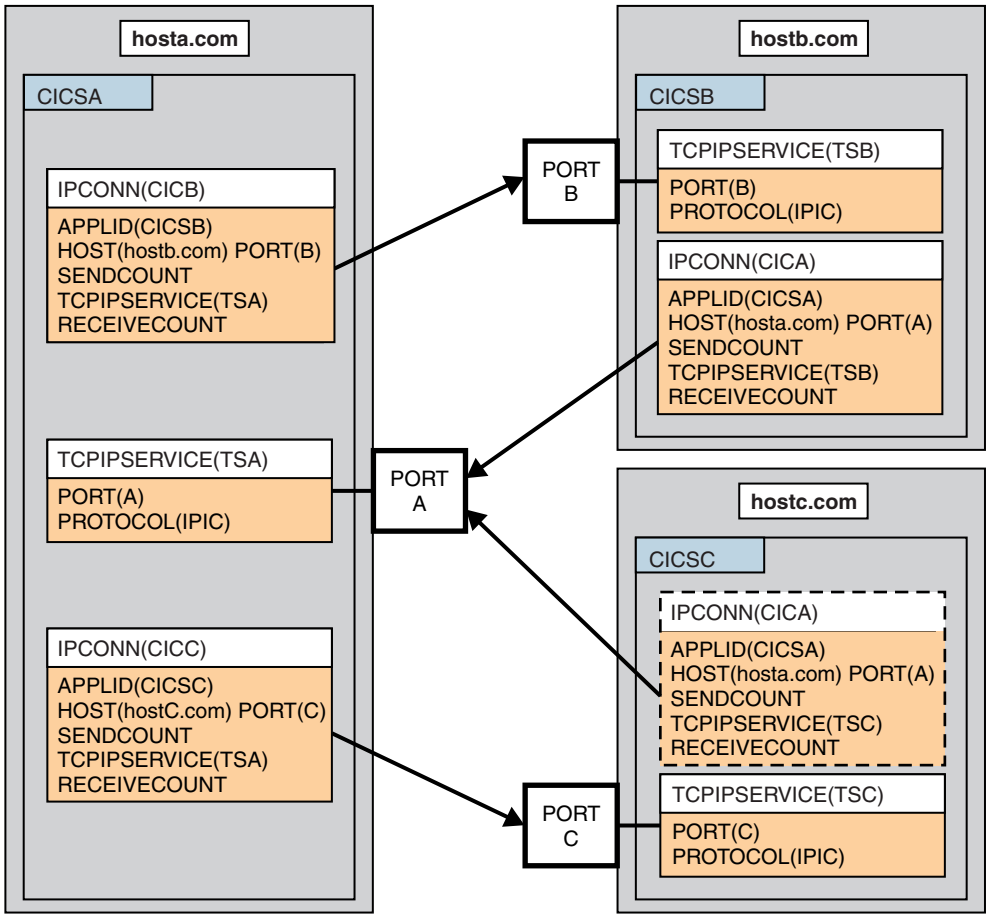


Figure 47. Related IPCONN and TCPIPSERVICE definitions

For detailed explanations of all the possible attributes of TCPIPSERVICE definitions, 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 159. 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 “ACF/VTAM LOGMODE table entries for CICS” on page 112.)

Defining the remote APPC system

The form of definition for an APPC system is shown in Figure 48.

```
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)
```

Figure 48. 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 145 (but see the box that follows).

Important:

> If you are defining an APPC link to a terminal-owning region that is a member of a VTAM generic resource group, NETNAME can specify either the TOR's *generic resource name*, or its applid. (See the note about VTAM generic resource names in “Generic and specific applids for XRF” on page 175.) For advice on coding NETNAME for connections to a generic resource, see Chapter 12, “Installation considerations for VTAM generic resources,” on page 119.

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 159.)

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 161.

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 24, “Intersystem session queue management,” on page 267.

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 162.

For information about security options, see Implementing LU6.2 security, in the *CICS RACF Security Guide*.

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 49.

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|UNCONDREL|NONE)
```

Figure 49. 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 names SNASVCMG or CPSVCMG.

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 161.

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 162.

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 50.

```
          CICSA                                  CICSB

DFHSIT  TYPE=CSECT                          DFHSIT  TYPE=CSECT
       ,APPLID=CICSA       1            3          ,APPLID=CICSB

DEFINE CONNECTION(CICB)    2           10   DEFINE CONNECTION(CICA)
    GROUP(groupname)                            GROUP(groupname)
    NETNAME(CICSB)         3            1       NETNAME(CICSA)
    ACCESSMETHOD(VTAM)                          ACCESSMEHOD(VTAM)
    PROTOCOL(APPC)                              PROTOCOL(APPC)
    SINGLESESS(N)          4            4       SINGLESESS(N)
    QUEUELIMIT(500)                             QUEUELIMIT(NO)
    MAXQTIME(500)                               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)          8            9       SENDSIZE(jjj)
    RECEIVESIZE(jjj)       9            8       RECEIVESIZE(kkk)
```

Figure 50. Defining compatible CICS APPC ISC nodes

In Figure 50, related options and operands are shown by identical numbers.

Note:

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 autoinstalling, 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 Personal Computers (PCs), all with the same characteristics, you would set up one template to autoinstall all of them. If 500 of your PCs 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 Autoinstalling APPC connections, in 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 48 on page 156 and Figure 49 on page 157. 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 Autoinstalling APPC connections, in 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 in which it is changed to 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 159), 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 8 primary contention-winning sessions.

Attention: 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 CEMT SET MODENAME, in 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 SESSIONS definition attributes, in the *CICS Resource Definition Guide*.

For further information about CICS support for persistent sessions, see Chapter 28, “Intercommunication and VTAM persistent sessions,” on page 307.

Defining logical unit type 6.1 links

Important:

You are advised to use MRO or APPC links for CICS-to-CICS communication.

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

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

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.
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-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.

The form of definition for individual LUTYPE6.1 sessions is shown in Figure 51 on page 165.

```
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 51. 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 111. The relationships between CICS and IMS definitions are summarized in Figure 52 on page 168.

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 175).

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 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 24, “Intersystem session queue management,” on page 267.

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 24, “Intersystem session queue management,” on page 267.

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 23, “CICS-to-IMS applications,” on page 245.

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 and RECEIVESIZE
Specify the maximum VTAM request unit (RU) sizes for these sessions.

- If CICS is the primary half-session, ensure that:
  1. The CICS SENDSIZE is less than or equal to the value specified on the RECANY parameter of the IMS COMM macro.
  2. The CICS RECEIVESIZE is greater than or equal to the IMS OUTBUF size.
- If IMS is the primary half-session, ensure that:
  1. The CICS SENDSIZE is greater than or equal to the IMS OUTBUF size.

2. The CICS RECEIVESIZE is less than or equal to the IMS RECANY size.

```
          CICS                                IMS
  DFHSIT TYPE=CSECT                     COMM      APPLID=SYSIMS

       ,SYSIDNT=CICL                  7             RECANY=nnn+22
       ,APPLID=SYSCICS     1                        EDTNAME=ISCEDT

                                      4   TYPE      UNITYPE=LUTYPE6

                                      1   TERMINAL  NAME=SYSCICS
DEFINE 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 SESSIONS(csdname)
    GROUP(groupname)
    CONNECTION(IMSR)       2
    SESSNAME(IMS2)                     8    SUBPOOL   NAME=CIC2
    NETNAMEQ(CIC2)         8                NAME      CICLT2 COMPT=2
    PROTOCOL(LU61)         4
    SENDCOUNT(1)
    SENDSIZE(nnn)          7           3    DFSHSBxx  USERVAR=SYSIMS
    RECEIVESIZE(mmm)       6
    IOAREALEN(nnn,16364)

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

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

Figure 52. Defining compatible CICS and IMS nodes

Figure 52 shows the relationship between the CICS and IMS definitions of an intersystem link. Related options and operands are shown by identical numbers.

Note: For an example of a VTAM logmode table entry for IMS, see Figure 33 on page 113.

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 53 on page 170). 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 53. Defining multiple links to an IMS node

Defining indirect links for transaction routing

In some older releases of CICS (no longer supported), indirect links between CICS regions were required for transaction routing across intermediate regions. In a network consisting solely of currently-available CICS systems, indirect links are only required if you are using non-VTAM terminals. Optionally, you can define them for use with VTAM terminals. 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.

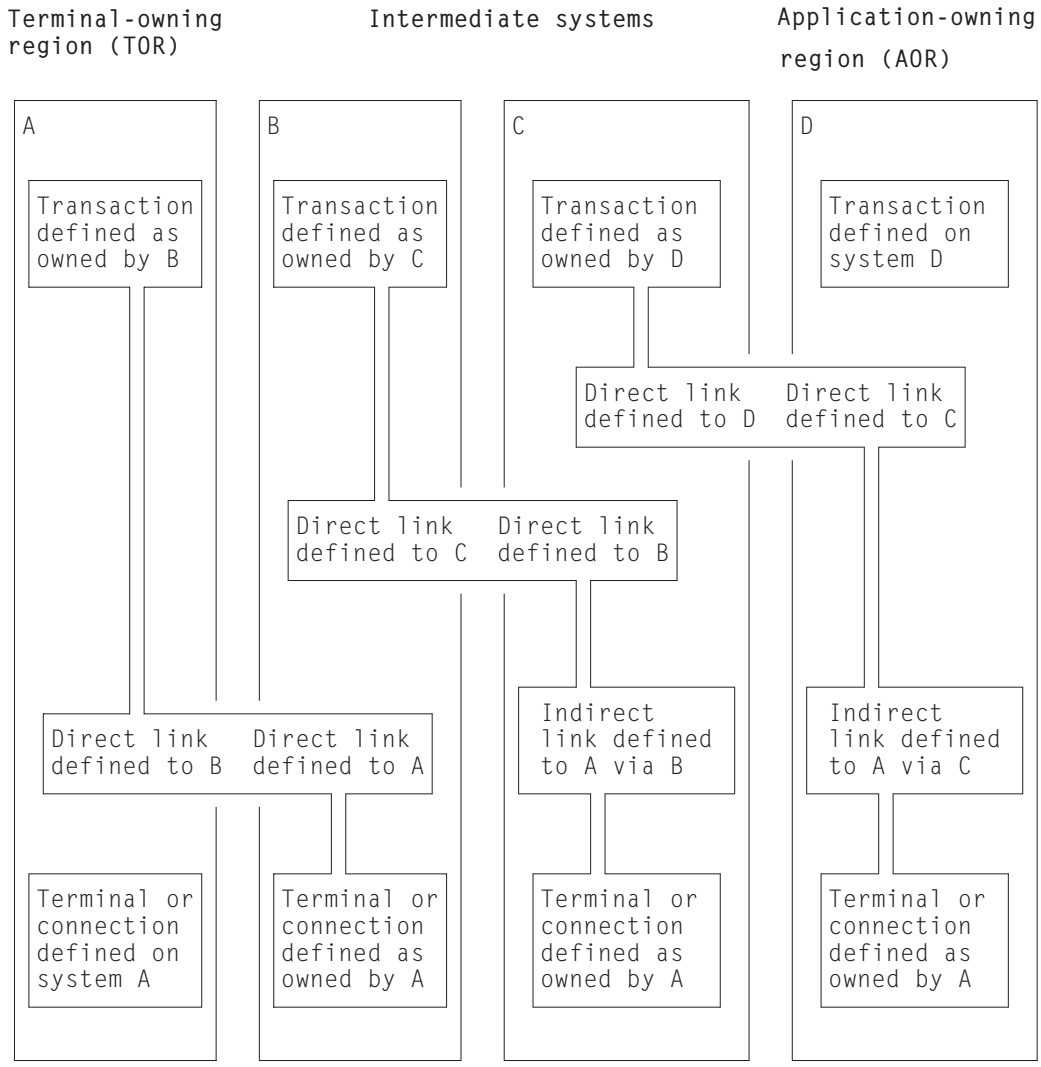


Figure 54. Indirect links for transaction routing

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 z/OS

As explained in Chapter 16, “Defining remote resources,” on page 191, 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 old releases of CICS (no longer supported), 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 54 on page 171, 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 z/OS, 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 z/OS, you may decide to define indirect links:

- To specify the preferred path to the TOR, if more than one exists, and you are using shippable terminals.
- 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 back-level system. If you introduce a new CICS Transaction Server for z/OS 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.

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 55 on page 174.

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.

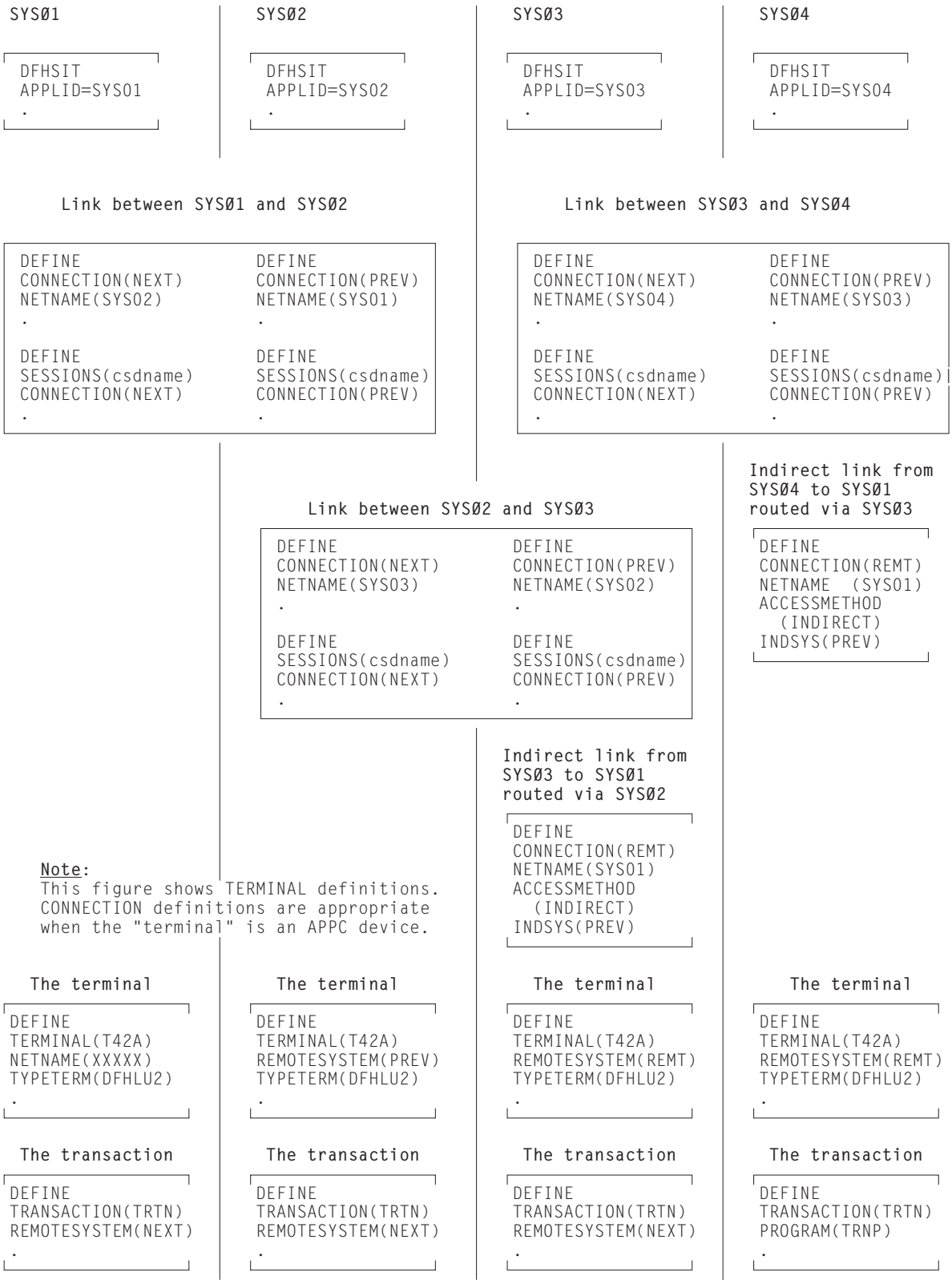


Figure 55. Defining indirect links for transaction routing. Because the remote terminal definitions in SYS04 and SYS03 do not specify the REMOTESYSNET option, indirect links are required.

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 uses hard-coded terminal definitions that do not specify REMOTESYSNET and the system 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 z/OS system are described in Chapter 16, “Defining remote resources,” on page 191.

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

Figure 55 on page 174 shows hard-coded remote terminal definitions that 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 16, “Defining remote resources,” on page 191.

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.

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/ESA 3.3 CICS XRF Guide*.

Important:

Do not confuse the term “generic applid” with “generic resource name”.

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

For XRF, a CICS system's **generic applid** is defined on the APPLID system initialization parameter and is the name by which CICS is known in the network. (That is, it is the name quoted by remote CICS systems, on the NETNAME option of CONNECTION definitions or the APPLID option of IPCONN definitions, to identify this CICS.)

A CICS system's **specific applid** is used to distinguish between the pair of XRF systems. It is the name quoted on a VTAM APPL statement, to identify this CICS to VTAM.

A CICS system's **generic resource name** is defined on the GRNAME system initialization parameter, and enables CICS to become a member of a VTAM generic resource group. See Chapter 12, “Installation considerations for VTAM generic resources,” on page 119.

Note, in particular, that:

- You cannot use both VTAM generic resources and XRF.
- If you use VTAM generic resources, you should specify only one name on the APPLID system initialization parameter.

Chapter 14. Managing APPC links

This chapter shows you how to use the master terminal transaction, CEMT, to manage APPC connections. It shows how the action of the CEMT commands is affected by the way the connections have been defined to CICS.

The commands are described under the headings:

- Acquiring the connection
- Controlling and monitoring sessions on the connection
- Releasing the connection.

The commands used to achieve these actions are:

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

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

The information is mainly about parallel-sessions connections between CICS systems.

General information about managing APPC links

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.

Note: 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 this 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 Chapter 26, “Recovery and restart in interconnected systems,” on page 277:

- 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 161. 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 6.

Table 6. 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.

Table 6. Effect of AUTOCONNECT on the SESSIONS definition (continued)

Setting	Effect
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 7, by issuing a CEMT INQUIRE MODENAME command.

Table 7. 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 executed to accomplish this are:

- CNOS negotiation with the partner system to define the changes that are to take place.
- Binding or unbinding of the appropriate sessions.

The algorithms used by CICS to negotiate with the partner the numbers of sessions to be made available are complex, and the numbers of sessions actually acquired may not match your expectation. The outcome can depend on the following:

- The history of preceding SET MODENAME commands
- The activity in the partner system
- Errors that have caused CICS to place sessions out of service.

Modegroups can normally be controlled with the few simple commands described in Table 8.

Table 8. SET MODENAME commands

Command	Effect
SET MODENAME ACQUIRED	Acquires all negotiated contention-winner sessions.

Table 8. SET MODENAME commands (continued)

Command	Effect
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.

The effects of limited resources

If an APPC connection traverses nonleased links (such as Dial, ISDN, X.25, X.21, or Token Ring links) to communicate to remote systems, the links can be defined within the network as limited resources. CICS recognizes this definition and automatically unbinds the sessions as soon as no transactions require them. If new transactions are invoked that require the connections, CICS binds the appropriate number of sessions. The connection status is shown by the CEMT INQUIRE CONNECTION command as follows:

Acquired
Some of the sessions in the connection are bound, and are probably in use. The LU services manager sessions in modegroup SNASVCMG may be unbound.

Available
The connection has been acquired, but there are no transactions that currently require the use of the connection. All the sessions have been unbound because they are defined in the network as limited resources.

The connection behaves in other ways exactly as for a connection over non-limited-resource links. The SET MODENAME and SET CONNECTION RELEASED commands operate normally.

Making the connection unavailable

The SET CONNECTION RELEASED command quiesces transactions using the connection and releases the connection. It cannot, on its own, prevent reacquisition of the connection from the partner system. To prevent your partner from reacquiring the connection, you must execute a sequence of commands. The choice of command sequence determines the status the connection adopts and how it responds to further commands from either partner.

If the number of available sessions for every modegroup of a connection is reduced to zero (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-SNAVCMG modegroups becomes zero.]

```
SET CONNECTION(....) RELEASED
       INQ CONNECTION(....)
```

[Repeat this command until the RELEASED status is displayed.]

```
SET CONNECTION(....) OUTSERVICE
```

Figure 56. 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 of APPC link management

Table 9 summarizes the effect of CEMT commands on the status of an APPC link.

Table 9. Effect of CEMT commands on an operational APPC link

Commands issued in sequence shown below								
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

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.

Chapter 15. TCP/IP management and control

TCP/IP management and control allows you to monitor work that enters or leaves CICS over Transmission Control Protocol/Internet Protocol (TCP/IP) connections. It provides, for TCP/IP networks, a subset of the management functions already provided for APPC networks; plus some additional functions that are not available for APPC or MRO networks.

Note: By "TCP/IP network" we mean systems that are interconnected by:

- IPIC connections (IPCONN) . Currently, these can be used only between CICS TS 3.2 regions, and between a CICS TS 3.2 region and a Java client.
- TCP/IP connections from clients that carry, for example, Web Interface, IIOP, or SOAP over HTTP requests inbound to CICS.

TCP/IP management and control enables you, for example, to:

- Use CICSPlex SM, or an equivalent tool, to:
  - Get a CICSplex-wide view of the TCP/IP network.
  - Examine, in real time:
    - The TCP/IP network resources that a particular CICS region is using
    - The work passing in and out of a particular CICS region over the TCP/IP network
    - The CICS resources and tasks associated with a distributed transaction that flows across the CICSplex over the TCP/IP network
    - The CICS region in which a distributed transaction originated
- Save the data collected by CICS so that it can be examined off line, at some point after the tasks and resources that it relates to are no longer available.

Reasons why you might want to use TCP/IP management and control include:

- To diagnose connectivity problems
- To investigate other problems, such as transaction delays
- To track work across the CICSplex
- To capture system data over time, for use in capacity planning
- To monitor the CICSplex

Some useful SPI commands

You can use the following system programming interface (SPI) commands to retrieve information about IPIC connections:

EXEC CICS EXTRACT STATISTICS
Specify a RESTYPE of "IPCONN" to retrieve resource statistics for IPIC. (Global statistics are not available.)

EXEC CICS INQUIRE ASSOCIATION
In a TCP/IP network, this command returns information about a task (for example, how the task was started, and the IP address of the TCP/IP client that requested it to start). The task is specified by a task number, which typically has been returned, as one of a list of numbers, by the EXEC CICS INQUIRE ASSOCIATION LIST command.

EXEC CICS INQUIRE ASSOCIATION LIST
This command returns a list of tasks, in the local region, that have matching

user correlation data in their associated data control blocks (ADCBs). Typically, the user correlation data has been added, at the point of origin of a distributed transaction, by a CICS XAPADMGR global user exit program. See “The XAPADMGR global user exit.”

EXEC CICS INQUIRE TASK
The IPALTFACILITIES option returns the address of a list of IDs, each of which identifies an IPCONN session that the task has used to communicate with another system. The LISTSIZE option returns the number of items in the list.

EXEC CICS PERFORM STATISTICS
Specify a statistics type of “IPCONN” to record resource statistics for IPIC connections. (Global statistics are not available.)

CICS ApplData

CICS TS 3.2 generates 40 bytes of application-specific information for each of the TCP sockets that it owns. CICS uses the SIOCSAPPLDATA ioctl to associate this information with the z/OS Communications Server TCP/IP socket. This new ioctl is provided by new function supplied with APARs PK32534 on z/OS V1R7 and PK40411 on z/OS V1R8. You can use this information to correlate TCP/IP connections with the CICS regions and transactions using them. This can be useful for problem determination, capacity planning, and accounting applications.

In CICS, you can get the ApplData information using the CECI INQUIRE ASSOCIATION transaction, CICSPlex SM displays and SMF records. In TCP/IP, the ApplData information is available on the Netstat ALL/-A, ALLConn/-a and COnn/-c reports and can be searched with the APPLD/-G filter. See *IP System Administrator's Commands* for additional information on using ApplData with Netstat. The ApplData information is available in the SMF 119 TCP Connection Termination record. See *IP Configuration Reference* for additional information. The ApplData information is available through the Network Management Interface. See *IP Programmer's Guide and Reference* for more information.

The XAPADMGR global user exit

Use the XAPADMGR exit for distributed transactions. It allows you to add user information to a task's Associated Data Origin Descriptor, at the point of origin of the distributed transaction. This information could later be used as, for example, search keys for processing carried out through CICSPlex SM. For further information about the XAPADMGR exit, see the *CICS Customization Guide*.

CICS provides a sample global user exit program, DFH$APAD, for use at the XAPADMGR exit point. The exit program is invoked, if enabled, when non-system tasks for which no input Origin Descriptor Record is provided are attached.

DFH$APAD performs the following processing:

- Provides addressability to the associated data provided as input to the exit.
- Chooses a field from this data and places it in the output buffer.
- Adds a field to the user correlation data in the output buffer.

Using CICSPlex SM to analyze TCP/IP traffic

As noted under “The XAPADMGR global user exit” on page 188 above, user correlation information added to a task's associated data origin descriptor, at the point of origin of the distributed transaction, can be used as search keys for later processing carried out through CICSPlex SM.

A search key (or “filter string”) can contain the following “wildcard” characters:

? matches exactly one arbitrary character

***** matches zero or more arbitrary characters

A filter string with no wildcards must be an exact match to the entire correlator. Therefore, a filter string that is a substring of the correlator must contain at least one wildcard character to match any user correlator string. For example, to find a substring that could be anywhere in the data add both a leading and a trailing '*' to your filter string.

The CICSPlex SM TASKASSC resource table provides information about the tasks that make up a distributed transaction. You can filter the records using a substring of the user correlation data added (by a CICS XAPADMGR global user exit program) to the user data section of the task's associated data origin descriptor.

For more information, see the *CICSPlex System Manager Resource Tables Reference* and the *CICSPlex System Manager Operations Views Reference* .

Using CICS monitoring to analyze TCP/IP traffic

Fields 360 through 371 in the performance class monitoring records in group DFHCICS are TCP/IP-related. See the *CICS Performance Guide* .

Chapter 16. Defining remote resources

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

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. But see““A note on daisy-chaining.”

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.

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.

Note: The following types of request cannot be daisy-chained:

- Dynamically-routed DPL requests—see “Daisy-chaining of DPL requests” on page 91
- Dynamically-routed transactions started by non-terminal-related START commands
- Dynamically-routed transactions that are associated with CICS business transaction services activities.

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 57.

```
     CICSA                                      CICSB
 (Local System)                             (Remote System)

DFHSIT TYPE=                                  DFHSIT TYPE=
       ,APPLID=CICSA       1
                                           3          ,APPLID=

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
       REMOTENAME(FILEB)   5
```

Figure 57. Local and remote resource names

Figure 57 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 57 on page 192, 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.

Defining remote resources for 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 is defined 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
DEFINE
   FILE(name)
   GROUP(.....)
   DESCRIPTION(......)
 Remote Attributes
   REMOTESYSTEM(name)
   REMOTENAME(name)
   RECORDSIZE(record-size)
   KEYLENGTH(key-length)
```

Figure 58. 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 26.)

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 the C language, 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 Sharing the CSD in non-RLS mode, in the *CICS System Definition Guide*.) If the local and remote systems share a CSD, you need define each 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

To enable the local CICS system to access remote DL/I databases, you must define the remote PSBs in a PDIR. The form of macro used for this purpose is:

```
DFHDLPSB TYPE=ENTRY
         ,PSB=psbname
         ,SYSIDNT=name
         ,MXSSASZ=value
         [,RMTNAME=name]
```

Figure 59. Macro for defining remote DL/I PSBs

This entry refers to a PSB that is known to IMS/ESA DM on the system identified by the SYSIDNT option.

The SYSIDNT and MXSSASZ operands are mandatory, because the PDIR contains only remote entries.

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 transient data queue must be defined as a remote destination.

A remote definition provides CICS with sufficient information to enable it to ship transient data requests to the specified remote system. Remote definitions are created as shown in Figure 60.

```
Definition using CEDA
DEFINE
   TDQUEUE(name)
   GROUP(groupname)
   DESCRIPTION(text)
 Remote Attributes
   REMOTESYSTEM(sysidnt)
   REMOTENAME(name)
   REMOTELENGTH(length)
```

Figure 60. Sample definitions for remote transient data queues

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 Temporary storage EXEC interface program exits, in 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 61.

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

Figure 61. Macro for defining remote temporary storage queues

Defining remote resources for 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.

Defining remote server programs

A remote server program can be defined using the CEDA transaction. Figure 62 shows the program attributes that you need to specify. How you specify the attributes depends on whether DPL requests for the program are to be routed to the remote region *statically* or *dynamically*.

```
DEFINE
   PROGRAM(name)
   GROUP(.....)
   DESCRIPTION(......)
 Remote Attributes
   REMOTESYSTEM(name)
   REMOTENAME(name)
   TRANSID(name)
   DYNAMIC(NO|YES)
```

Figure 62. Defining a remote program (DPL)

The name of the remote system

To route DPL requests for the program statically:

- Allow the value of the DYNAMIC option to default to NO.
- On the REMOTESYSTEM option, specify the name of the server region to which LINK requests for this program are to be shipped. The name must be the name of an installed CONNECTION definition or an installed IPCONN definition.

An EXEC CICS LINK command that names the program is shipped to the server region named on the REMOTESYSTEM option.

To route DPL requests for the program dynamically:

- Specify DYNAMIC(YES).
- Do not specify the REMOTESYSTEM option; or use REMOTESYSTEM to specify a default server region.

An EXEC CICS LINK command that names the program causes the dynamic routing program to be invoked. The routing program can select the server region to which the request is shipped.

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.

For dynamic requests that are routed using the CICSPlex System Manager (CICSPlex SM), the TRANSID option takes on a special significance, because CICSPlex SM's routing logic is transaction-based. CICSPlex SM routes each DPL request according to the rules specified for its associated transaction.

Note: The CICSPlex SM system programmer can use the EYU9WRAM user-replaceable module to change the transaction ID associated with a DPL request.

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

When definitions of remote server programs aren't required

There are some circumstances in which you may not need to install a static definition of a remote server program:

- The server program is to be autoinstalled.

  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 Autoinstalling programs, map sets, and partition sets, in the *CICS Resource Definition Guide*. For programming information about writing program-autoinstall user programs, see Writing a program to control autoinstall of APPC connections, in the *CICS Customization Guide*.)

  When the autoinstall user program is invoked, it can install:

  A local definition of the server program
  CICS runs the server program on the local region.

  A definition that specifies REMOTESYSTEM(remote_region) and DYNAMIC(NO)
  CICS ships the LINK request to the remote region.

  A definition that specifies DYNAMIC(YES)
  CICS invokes the dynamic routing program to route the LINK request.

  Note: The DYNAMIC attribute takes precedence over the REMOTESYSTEM attribute. Thus, a definition that specifies both REMOTESYSTEM(remote_region) and DYNAMIC(YES) defines the program as dynamic, rather than as residing on a particular remote region. (In this case, the REMOTESYSTEM attribute names the default server region passed to the dynamic routing program.)

  No definition of the server program
  CICS invokes the dynamic routing program to route the LINK request.

Note: This assumes that the autoinstall control program *chooses* not to install a definition. If no definition is installed because autoinstall fails, the dynamic routing program is not invoked.

- The client program names the target region explicitly, by specifying the SYSID option on the EXEC CICS LINK command.

  Note:

  1. If there is no installed definition of the program named on the LINK command, the dynamic routing program is invoked but cannot route the request, which is shipped to the remote region named on the SYSID option.
  2. If the SYSID option names the local CICS region, the dynamic routing program *is* able to route the request.

- DPL calls for the server program are to be routed dynamically.

  If there is no installed definition of the program named on the LINK command, the dynamic routing program is invoked and (provided that the SYSID option is not specified) can route the request.

  Note: Although in some cases a remote definition of the server program may not be necessary, in others a definition will be required—to set the program's REMOTENAME or TRANSID attribute, for example. In these cases, you should install a definition that specifies DYNAMIC(YES).

Defining remote resources for asynchronous processing

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

Note, however, that an application can use the CICS RETRIEVE command to obtain the name of a remote temporary storage queue which it subsequently names in a function shipping request.

Defining remote transactions

A remote transaction for CICS asynchronous processing is a transaction that is owned by another system and is invoked from the local CICS system only by 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 63 on page 199.

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

Figure 63. 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 5, “Asynchronous processing,” on page 37.

Restriction on the REMOTENAME option

Some asynchronous-processing requests are for processes that involve transaction routing. One example is a 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.

Defining remote resources for transaction routing

CICS transactions can be routed to remote regions either statically or dynamically. A transaction that is to be routed may be started in a variety of ways. For example:

- From a user-terminal.
- By a terminal-related ATI request (for example, a terminal-related EXEC CICS START command).
- By a non-terminal-related ATI request (for example, by a non-terminal-related EXEC CICS START command).
- If the transaction is associated with a CICS business transaction services (BTS) activity, by a BTS RUN ASYNCHRONOUS command. (BTS is described in What are CICS business transaction services?, in the *CICS Business Transaction Services*.)

The resources you need to define are:

- If the request to start the transaction is associated with a terminal, the terminal—see “Defining terminals for transaction routing”
- In every case, the transaction—see “Defining transactions for transaction routing” on page 209.

Defining terminals for transaction routing

The information in this section applies only to terminal-related transaction routing—that is, to the routing of:

- Transactions started from user-terminals
- Transactions started by terminal-related ATI requests.

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 7770 or 2260 terminals
- Pooled 3600 or 3650 pipeline logical units
- MVS 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 202).
   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 “Defining indirect links for transaction routing” on page 170).

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 202.

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 207.)
- 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, normally you do not need to specify REMOTESYSNET. However, if the direct link is an APPC connection to a

TOR that is a member of a VTAM generic resource group, you may need to specify REMOTESYSNET. REMOTESYSNET is needed in this case if the NETNAME specified on the CONNECTION definition is the generic resource name of the TOR (not the applid).

Only a few of the various terminal properties need be specified for a remote terminal definition. They are:

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

Figure 64. 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 65. 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 65. 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 Sharing the CSD in non-RLS mode, in 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 “Defining indirect links for transaction routing” on page 170).

When a shipped definition is to be installed on an intermediate or application-owning region, the autoinstall user program is invoked in that region. If the name of the shipped definition clashes with that of a remote terminal or connection already installed on the region, CICS assigns an *alias* to the shipped definition, and passes the alias to the autoinstall user program. (Terminal aliases are described on page “Terminal aliases” on page 208.) CICS-generated aliases for

shipped terminals and connections are recognizable by their first character, which is always '{'. Their remaining three characters can have the values 'AAA' through '999'. Your autoinstall user program can accept a CICS-generated alias, override it, or reject the install. Note that it can also specify an alias for a shipped definition when there is *no* clash with an installed remote definition.

You need to consider assigning aliases to shipped definitions if, for example, you have two or more terminal-owning regions that use similar sets of terminal identifiers for transaction routing to the same AOR. For information about writing an autoinstall user program to control the installation of shipped terminals, see Writing a program to control autoinstall of shipped terminals, in the *CICS Customization Guide*.

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. You do this because, for example, specifying a remote terminal for an intrapartition transient data queue (see “Defining intrapartition transient data queues” on page 222) 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.

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 61 for more information.

If you function ship START 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 66 for more details.

A better way of handling terminal-related START requests is to use the enhanced routing methods described in “Routing transactions invoked by START commands” on page 68. If the START request is issued in the TOR, it is *not* function-shipped to the AOR: thus the “terminal-not-known” cannot occur; nor do you need to use FSSTAFF to prevent the transaction being started against the “wrong” terminal. Instead, the START executes directly in the TOR, and the transaction is routed as if it had been initiated from a terminal. If you are using shippable terminals, a terminal definition is shipped to the AOR if required.

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 66. Defining a shippable terminal (transaction routing)

This method can be used for any VTAM terminal. It is particularly appropriate if you use autoinstall in the TOR.

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 25, “Efficient deletion of shipped terminal definitions,” on page 271.

Defining remote non-VTAM terminals

Non-VTAM terminals must be defined using resource definition macros: you cannot use RDO.

Note: CICS Transaction Server for z/OS, Version 3 Release 2 does not directly support the Telecommunication Access Method (TCAM). However, TCAM *is* supported indirectly: that is, terminals connected by TCAM/DCB (not TCAM/ACB) to a pre-CICS TS for z/OS TOR can use transaction routing to gain access to a CICS Transaction Server for z/OS, Version 3 Release 2 AOR. Thus, you can define remote (but not local) TCAM terminals to a CICS Transaction Server for z/OS, Version 3 Release 2 system.

CICS Transaction Server for z/OS, Version 3 Release 2 does not support the Basic Telecommunication Access Method (BTAM) at all, even indirectly. Thus, you cannot define remote BTAM 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.

Using resource definition macros, you can define remote non-VTAM 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

Both methods allow the same terminal definitions to be used to generate the required entries in both the local and the remote system.

Definition using DFHTCT TYPE=REMOTE:

The format of the DFHTCT TYPE=REMOTE macro 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]
    TCAM SNA Only
    [,BMSFEAT=([FMHPARM][,NOROUTE][,NOROUTEALL]
        [,OBFMT][,OBOPID])]
    [,HF={NO|YES}]
    [,LDC={listname|(aa[=nnn],bb[=nnn],cc[=nnn],...)
    [,SESTYPE=session-type]
    [,VF={NO|YES}]
```

Figure 67. 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 “Defining indirect links for transaction routing” on page 170).

Sharing terminal definitions:

TCAM terminals:

> This section applies to all supported types of non-VTAM terminals except TCAM. CICS Transaction Server for z/OS, Version 3 Release 2 does not support local TCAM terminals and therefore no local definitions are built.

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 68 on page 206.

```
  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=REMOTE,                DFHTCT TYPE=REMOTE,
       SYSIDNT=CICR                       SYSIDNT=CICR
       TRMIDNT=aaaa,                      TRMIDNT=aaaa,
       TRMTYPE=3277,                      TRMTYPE=3277,
       TRMMODL=2,                         TRMMODL=2,
       ALTSCRN=(43,80)                    ALTSCRN=(43,80)
       .                                   .
       .                                   .
DFHTCT TYPE=FINAL                  DFHTCT TYPE=FINAL
```

Figure 68. Typical use of DFHTCT TYPE=REMOTE macro

In Figure 68, 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 TCAM terminals, no local terminal definitions are built.

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 remote 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 “Defining indirect links for transaction routing” on page 170).

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 69.

```
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
```

Figure 69. Typical use of DFHTCT TYPE=REGION macro

In Figure 69, the same copy book of terminal definitions is used in both the terminal-owning region and the application-owning region.

In the terminal-owning region, local terminal entries are built.

Note: For TCAM terminals, no local terminal definitions are built.

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.

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 70. 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. (Nor can you have two remote terminal definitions (for terminals on different remote regions) 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:

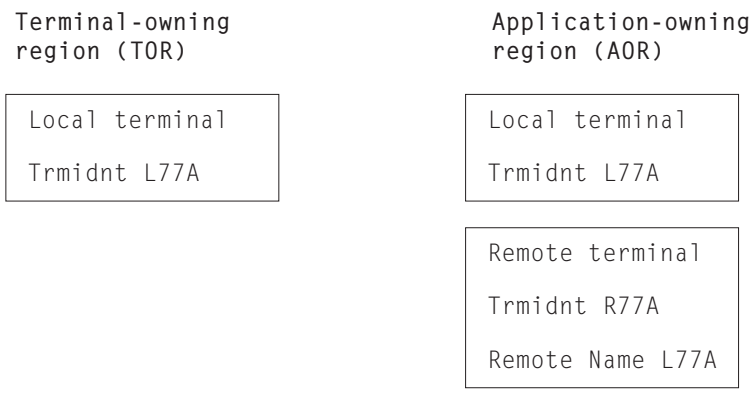


Figure 71. Local and remote names for remote terminals

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. It applies to all forms of transaction routing.

The general form of the CEDA DEFINE command for a transaction is shown in Figure 72 on page 210.

```
  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)
    ISOLATE(YES|NO)
REMOTE ATTRIBUTES
    DYNAMIC(NO|YES)
    REMOTESYSTEM(name)
    REMOTENAME(local-name|remote-name)
    TRPROF(DFHCICSS|name)
    LOCALQ(NO|YES)
    ROUTABLE(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 72. 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. [8] 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 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.

8. We ignore here the special case of an EXEC CICS START command that uses the SYSID option to name the remote region on which the transaction is to run. A remote region named explicitly on a START command takes precedence over one named on the transaction definition.

In this case, the transaction is routed to the system named in the REMOTESYSTEM option. This is known as **static** transaction routing. [9]

3. The remote attributes specify DYNAMIC(YES).

   In this case, the decision about where to execute the transaction is taken by your dynamic or distributed routing program. See “Two routing programs” on page 53.

   Note: Exceptions to this rule are transactions initiated by EXEC CICS START commands that are ineligible for enhanced routing. For example, if one of these transactions is defined as DYNAMIC(YES), your dynamic routing program is invoked but cannot route the transaction. See “Routing transactions invoked by START commands” on page 68.

The name in the TRANSACTION option is the name by which the transaction is invoked in the local 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.

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 217.

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. See Transaction security, in the *CICS RACF Security Guide* for information on security.
- For transaction routing on mapped APPC connections or MRO sessions, 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

9. The REMOTESYSTEM option must name a **direct** link to another system (not an indirect link nor a remote APPC connection).

are no free sessions to a remote system, ALLOCATE requests should be queued or rejected, are described in Chapter 24, “Intersystem session queue management,” on page 267.)

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 requesting region: the REMOTESYSTEM option must specify the name of the target region (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 region: the REMOTESYSTEM option must be blank, or specify the name of the target region.

If the transaction may be initiated by an EXEC CICS START command, check whether you can use the enhanced routing method described in “Routing transactions invoked by START commands” on page 68. If enhanced routing is possible, define the transaction as ROUTABLE(YES) in the region in which the START will be issued.

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 requesting region and the target region (and possibly between intermediate systems too, if “daisy chaining” is involved). The REMOTESYSTEM option specifies the name of the target region.

If the transaction may be initiated by an EXEC CICS START command, check whether you can use the enhanced routing method described in “Routing transactions invoked by START commands” on page 68. If enhanced routing is possible, specify the single definition as ROUTABLE(YES).

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 requesting region), a remote transaction definition is created. If they are the same (as in the target region), 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 Sharing the CSD in non-RLS mode, in the *CICS System Definition Guide*.

Dynamic transaction routing

There are three methods of defining transactions that are to be dynamically routed.

Note: Using dual-purpose definitions (on which the REMOTESYSTEM option specifies the default target region) is a fourth 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 routing program to be invoked unnecessarily in the target region, after the transaction has been routed.

Using separate local and remote definitions: This method is as described under "Static transaction routing" on page 212. It is the recommended method for transactions that may be initiated by terminal-related EXEC CICS START commands.

For dynamic routing of a transaction initiated by a START command, you must define the transaction as ROUTABLE(YES) in the region in which the START command is issued.

Using identical definitions: This is the recommended method for transactions that:

- Are associated with CICS business transaction services (BTS) activities
- Are associated with method requests for enterprise beans or CORBA stateless objects (the request processor transactions specified on the REQUESTMODEL definitions)
- May be initiated by non-terminal-related START commands.

These types of transactions are routed using the distributed routing model, which is a peer-to-peer system—each region can be both a requesting/routing region and a target region. Therefore, the transactions should be defined identically in each participating region. The regions may or may not be able to share a CSD—see Sharing the CSD in non-RLS mode, in the *CICS System Definition Guide*.

On each TRANSACTION definition:

- Specify DYNAMIC(YES).
- Do not specify a value for the REMOTESYSTEM option.
- If the transaction may be initiated by a non-terminal-related START command, specify ROUTABLE(YES).

Note that the "identical definitions" method differs from the "dual-purpose definitions" method in several ways:

- It is used for dynamic, not static, routing.
- The TRANSACTION definitions do not specify the REMOTESYSTEM option.
- The participating regions are not required to share a CSD.

Using a single transaction definition in the TOR: This is the recommended method for terminal-initiated transactions. 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 for all your terminal-initiated transactions:

- 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 terminal-initiated transaction you should define as dynamic is the dynamic transaction routing definition specified on the DTRTRAN parameter.
- The only terminal-initiated transactions you should define as remote are those 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 73.

```
  DEFINE
    TRANSACTION(CRTX)
    GROUP(DFHISC)
    PROGRAM(########)
    TWASIZE(00000)
    PROFILE(DFHCICST)
    STATUS(ENABLED)
    TASKDATALOC(ANY)
    TASKDATAKEY(CICS)
REMOTE ATTRIBUTES
    DYNAMIC(YES)
    REMOTESYSTEM()
    REMOTENAME()
    TRPROF(DFHCICSS)
    ROUTABLE(NO)
RECOVERY
    DTIMOUT(NO)
    INDOUBT(BACKOUT)
    RESTART(NO)
    SPURGE(YES)
    TPURGE(YES)
```

Figure 73. 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.

ROUTABLE(NO)
This attribute relates to the enhanced routing of transactions initiated by EXEC CICS START commands.

Specifying ROUTABLE(YES) means that, if the transaction is the subject of an eligible START command, it will be routed using the enhanced routing method described in “Routing transactions invoked by START commands” on page 68. You are recommended to:

- Specify ROUTABLE(NO) on the common transaction definition
- Install individual definitions of transactions that may be initiated by START commands.

By reserving the common definition for use with transactions that are started from user-terminals, you prevent transactions that are initiated by terminal-related START commands from being dynamically routed “by accident”.

Defining remote resources for DTP

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 191.)

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 74 shows the general form of the CEDA DEFINE PARTNER command. The PARTNER resource has been designed specifically to support **Systems**

```
DEFINE
  PARTNER(sym_dest_name)
  [GROUP(groupname)]
  [NETWORK(name)]
  NETNAME(name)
  [PROFILE(name)]
  {TPNAME(name)|XTPNAME(value)}
```

Figure 74. Defining a remote partner

Application Architecture (SAA) conventions. For more guidance about this, see PARTNER resource definitions, in 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 17. 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 a session to another system, either explicitly by means of an ALLOCATE command or implicitly because it uses, for example, function shipping, a communication profile is associated with the communication between the transaction and the session. The communication profile specifies the following information:

- 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 ALLOCATE command.

The options of the CEDA DEFINE PROFILE command that are relevant to intersystem sessions are shown in Figure 75 on page 218. For further information about the CEDA DEFINE PROFILE command, see PROFILE definition attributes, in the *CICS Resource Definition Guide*.

A profile is always required for a session acquired by an ALLOCATE command; either a profile that you have defined and which is named explicitly on the command, or the default profile DFHCICSA. If CICS cannot find the profile, the CBIDERR condition is raised in the application program.

The only option shown in Figure 75 on page 218 that applies to MRO sessions is INBFMH. And, for MRO sessions that are acquired by an 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 75. 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 CSPG - page retrieval, in 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 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 or DPL request.

Note that, if you use DPL, you may need to increase the value specified for RTIMEOUT—see “Modifying the default profiles.”

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.

You can modify DFHCICSF, used for function shipping and DPL requests. One reason for doing so might be to increase the value of the RTIMEOUT option. For example, the default value may be adequate for single function shipping requests, but inadequate for a DPL call to a back-end program that retrieves a succession of records from a data base.

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 76 on page 221.

```
  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        DFHZLS1     LU services model
  06F20000    CLS2        DFHLUP      LU services model
     -        CLS3        DFHLUP      LU services model
```

Figure 76. 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 31.)

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 CICS-supplied resource definitions, groups, and lists, in the *CICS Resource Definition Guide*.

Some of the contents of groups DFHISC and DFHSTAND are summarized in Figure 77 on page 222.

```
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     DFHZLS1   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 77. Some definitions required for ISC and MRO

Defining intrapartition transient data queues

An intrapartition transient data queue can be defined as shown:

```
DEFINE
  TDQUEUE(name)
  GROUP(groupname)
  DESCRIPTION(text)
  TYPE(Intra)
  Intrapartition Attributes
  ATIFACILITY(terminal)
  RECOVSTATUS(logical)
  FACILITYID (terminal)
  RECOVSTATUS(name)
  TRANSID ()
  TRIGGERLEVEL(value)
  USERID(userid)
  Indoubt Attributes:
  WAIT(yes)
  WAITACTION(reject)
        ...
```

Figure 78. Defining an intrapartition transient data queue

For further information about defining transient data queues, see Defining TDQUEUE resources, in the *CICS Resource Definition Guide*. This section is concerned with the CICS intercommunication aspects of queues that:

- Cause automatic transaction initiation
- Specify an associated principal facility (such as a terminal or another 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 name in the queue definition 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 transient data queue definition. A principal facility 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 a destination of terminal, and give a terminal identifier. If you omit the terminal identifier, the name of the terminal defaults to the name of the queue.

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 7, “CICS transaction routing,” on page 55), and the normal transaction routing rules apply.

For any remote terminal other than an APPC terminal, specify a destination of terminal and a terminal identifier.

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 definition for the transient data queue. 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 95 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 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 7, “CICS transaction routing,” on page 55), and the normal transaction routing rules apply.

You can name a remote connection in the definition for the transient data queue.

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.

It is advisable to define the user transaction to execute in the local CICS region, by specifying DYNAMIC(NO) and no REMOTE attributes. Routing the mirror transaction to another CICS region can impact performance and make problem determination more difficult.

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 Autoinstalling programs, map sets, and partition sets, in the *CICS Resource Definition Guide*.)

Part 4. Application programming in an intersystem environment

This part of the manual describes the application programming aspects of CICS intercommunication. It gives you an overview of CICS application programming in an intersystem environment, and then discusses programming for each of the following CICS intercommunication facilities:

- Function shipping
- Distributed program link (DPL)
- Asynchronous processing
- Transaction routing
- CICS-to-IMS applications

Chapter 18. 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
- 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 or session that is associated with your transaction when the transaction is initiated. 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, a pair of transactions converse with one another. The *front-end transaction* is initiated first, acquires a session to the remote system, and causes the *back-end transaction* to be initiated.

  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.

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, the system is not defined to CICS, it is not available, or the session fails).

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 HANDLE ABEND 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 possibility 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*.

Chapter 19. 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 21, “Application programming for asynchronous processing,” on page 239.

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 25.

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 BDAM 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 OS/390 system, or DL/I DOS/VS databases associated with a remote CICS/VSE system. (See Chapter 1, “Introduction to CICS intercommunication,” on page 3 for a list of systems with which CICS Transaction Server for z/OS, Version 3 Release 2 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.

Only a subset of DL/I requests can be function shipped to a remote CICS system. For guidance about restrictions, see the *CICS IMS Database Control Guide*.

Temporary storage

Function shipping allows you to send data to or receive data from temporary-storage queues located on remote systems. Definitions of remote temporary-storage queues can be made by the system programmer. You can, however, use the SYSID option on the WRITEQ TS, READQ TS, and DELETEQ TS commands to specify the system on which the request is to be executed.

For MRO sessions, the MAIN and AUXILIARY options of the WRITEQ TS command can be used to select the required type of storage.

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 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 if it is not specified in the transient data resource definition that has been installed.

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 Intersystem communication program exits XISCONA and XISLCLQ, in 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 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 (ADCD).

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 20. 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 z/OS 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 z/OS systems; (they could be, for example, CICS Transaction Server for Windows or CICS 6000 systems). See Chapter 1, "Introduction to CICS intercommunication," on page 3 for a list of systems with which CICS Transaction Server for z/OS can communicate.

DPL programs can be written in PL/I, C, COBOL, or assembler language.

As Chapter 8, "CICS distributed program link," on page 85 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 by the program resource definition or the dynamic routing program. Optionally, you can use the SYSID option on the LINK command to select the system on which the command is to be executed.

When your client program issues a 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 a 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 8, "CICS distributed program link," on page 85.

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.

Failure of the server program

If the server program fails, the ABEND condition and an abend code are returned to the client program. The client program therefore also terminates abnormally, unless it has issued the HANDLE ABEND command before issuing the LINK command.

The server program

Permitted commands

The EXEC CICS commands that a DPL server program can issue are limited to a subset of the CICS API. For details of the restricted DPL subset, see API restrictions for distributed program link, in the *CICS Application Programming Reference*.

Syncpoints

If the server program was started by a 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 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 SYNCPOINT requests.
- 'DS' if the program was started by a LINK request **with** the SYNCONRETURN option, and can therefore issue 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 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 “Remote system not available” on page 233.

The default action for the SYSIDERR condition is to terminate the task abnormally.

Server's work backed out

If the client program issues the 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 a 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 a 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 a 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 LINK command fails if, for example, MROLRM=YES is specified in the CICS server region (SERV). This is because the mirror used for the READQ TS command is still around. For the above sequence of commands to work, the client program must issue a SYNCPOINT after the 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 Mixing DPL and function shipping to the same CICS system, in the *CICS Application Programming Guide*.

These errors are indicated by the INVREQ and PGMIDERR conditions.

On 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. A RESP2 value of 17 indicates that a TRANSID value of spaces (blanks) was supplied by the dynamic routing program.

On the PGMIDERR condition, an accompanying RESP2 value of 25 indicates that the dynamic routing program rejected the link request.

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.

Multiple updates to a recoverable resource by the same distributed UOW

In a non-DPL environment, it is possible for multiple programs within one unit of work (UOW) to update the same recoverable resource. For instance, `program1` might update `Record1` in a recoverable file, then link to `program2`, which could update the same record, `Record1`, in the same file. This is not necessarily good programming practice but it does work, because CICS considers the owner of the resource to be the task, not the program.

However, in a DPL environment, where the programs involved are running in different CICS regions, it is not possible for multiple programs to update the same recoverable resource within the same UOW. Using the same example as above, `program1` updates `Record1` in a recoverable file, then links to `program2`, which runs under a mirror task in another region. If `program2` function-ships a file control request to update `Record1` in the same file, the request hangs. It hangs because the mirror task processing `program2`'s file control request cannot get the record lock for `Record1`. The lock is owned by the task under which `program1` is running. Even though the file control mirror task and the task under which `program1` is running are part of the same distributed UOW, CICS does not allow the update. This is because CICS uses the task, not the distributed UOW, as the basis for locking recoverable resources.

Chapter 21. Application programming for asynchronous processing

This chapter discusses the application programming requirements for CICS-to-CICS asynchronous processing. The general information given for CICS transactions that use the START or RETRIEVE commands is also applicable to CICS-to-IMS communication.

A description of the concepts of asynchronous processing is given in Chapter 5, “Asynchronous processing,” on page 37. 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 START, in the *CICS Application Programming Reference*.

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 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 233).

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 START commands” on page 42).

Retrieving data associated with a remotely-issued start request

The RETRIEVE command is used to retrieve data that has been stored for a task as a result of a remotely-issued start request. This is the only available method for accessing such data.

As far as your transaction is concerned, there is no distinction between data stored by a remote start request and data stored by a local start request, and the normal considerations for use of the RETRIEVE command apply.

Chapter 22. 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 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 be written in C or assembler language.

Basic mapping support

Any BMS maps or partition sets that your program uses must reside in the same CICS system as the program.

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 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.

The *map* name is supported in the same way as the *mapset* name. However, some old CICS products (no longer supported) have no knowledge of map names being passed in ATTACH and DETACH sequences. When sending an ATTACH sequence, CICS Transaction Server for z/OS 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.

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. EIBAID, which contains the attention identifier, is always set to zeros at the start of a session.

Using the EXEC CICS ASSIGN command in the AOR

You may find that two of the options of the EXEC CICS ASSIGN command return unexpected values.

PRINSYSID
This option returns the sysid of the principal facility to the transaction. 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.

USERID
For a routed transaction, CICS takes the userid from one of several sources, depending on how you specified your security requirements. See Transaction routing security with LU6.2, in the *CICS RACF Security Guide*.

As Table 10 on page 243 shows, CICS returns the following values:

- If the connection is defined with the ATTACHSEC(LOCAL) option, and SEC=YES or MIGRATE 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 RACF 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 or MIGRATE 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 10. Values returned by the USERID option of EXEC CICS ASSIGN, for routed transactions

<table>
<tr><th rowspan="3">TOR's DFHSIT SEC=</th><th colspan="3">ATTACHSEC value in CONNECTION definition</th></tr>
<tr><th rowspan="2">IDENTIFY
VERIFY
PERSISTENT
MIXIDPE</th><th colspan="2">LOCAL</th></tr>
<tr><th>AOR's DFHSIT SEC=YES or MIGRATE</th><th>AOR's DFHSIT SEC=NO</th></tr>
<tr><td>YES
or
MIGRATE</td><td>Userid sent at attach</td><td rowspan="2"><u>ISC</u>
1. USERID of session
2. SECURITYNAME of connection

<u>MRO</u> RACF userid of TOR</td><td rowspan="2">DFLTUSER of AOR</td></tr>
<tr><td>NO</td><td>Userid sent at attach
(DFLTUSER of TOR)</td></tr>
</table>

Chapter 23. 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 79 on page 246.

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.

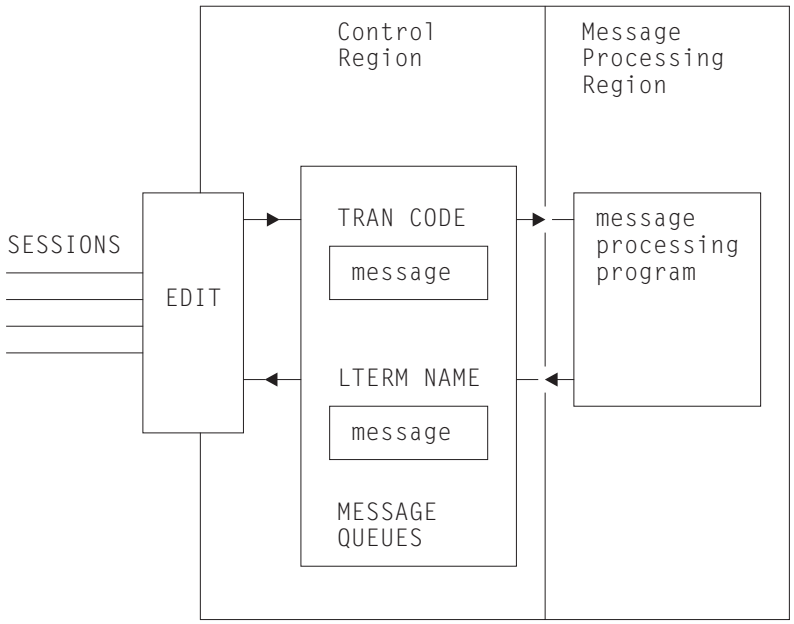
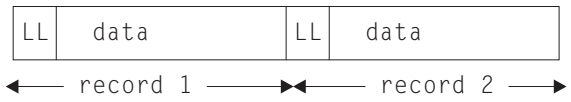


Figure 79. Basic IMS message queuing

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.



In CICS, the I/O area contains a complete message, which can contain one or more records. The blocking of records for output, and the deblocking on input, must be done by your CICS application program.

Chain of RUs

In this format, which is the most common CICS format, a message is transmitted as multiple SNA RUs, as shown here.

In CICS, the I/O area contains a complete message.

Forms of intersystem communication with IMS

There are three forms of CICS-to-IMS communication that must be considered:

1. Asynchronous processing using CICS START and RETRIEVE commands

2. Asynchronous processing using CICS SEND LAST and RECEIVE commands
3. Distributed transaction processing (that is, synchronous processing) using CICS SEND and RECEIVE commands.

The basic differences between these forms of communication are described in Chapter 5, “Asynchronous processing,” on page 37 and Chapter 9, “Distributed transaction processing,” on page 95.

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.

CICS-to-IMS applications—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 CICS START and RETRIEVE “interval control” commands, see , in the *CICS Application Programming Reference*. 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 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 80.

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.

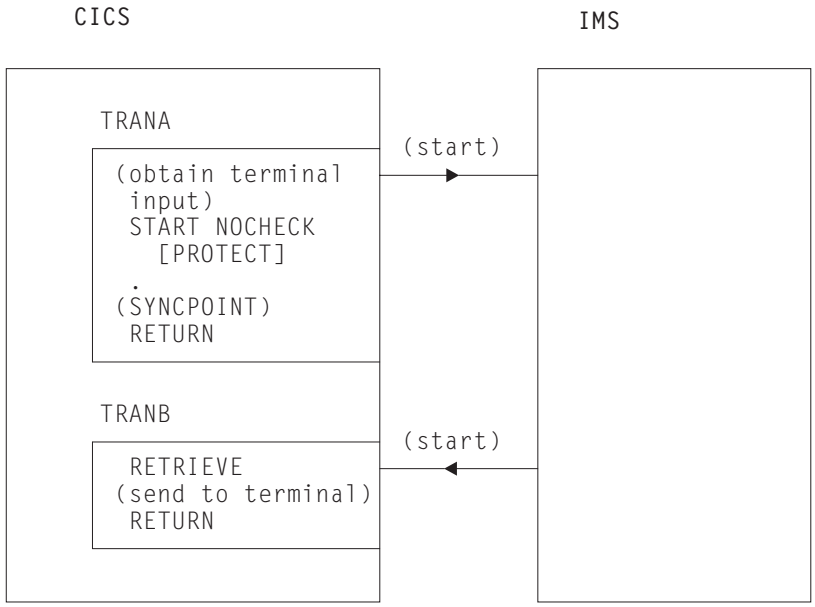


Figure 80. START and RETRIEVE asynchronous processing–CICS front end

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 42), 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 81.

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.

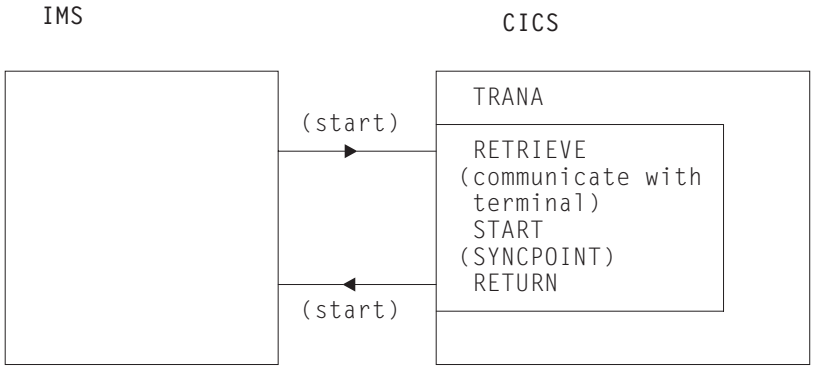


Figure 81. RETRIEVE and START asynchronous processing – IMS front end

The START command

This section shows the format of the START command that is used to schedule remote IMS transactions. Note that no interval control is possible (although it is not an error to specify INTERVAL(0)) and that the NOCHECK 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 13, “Defining links to remote systems,” on page 143).

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 82.

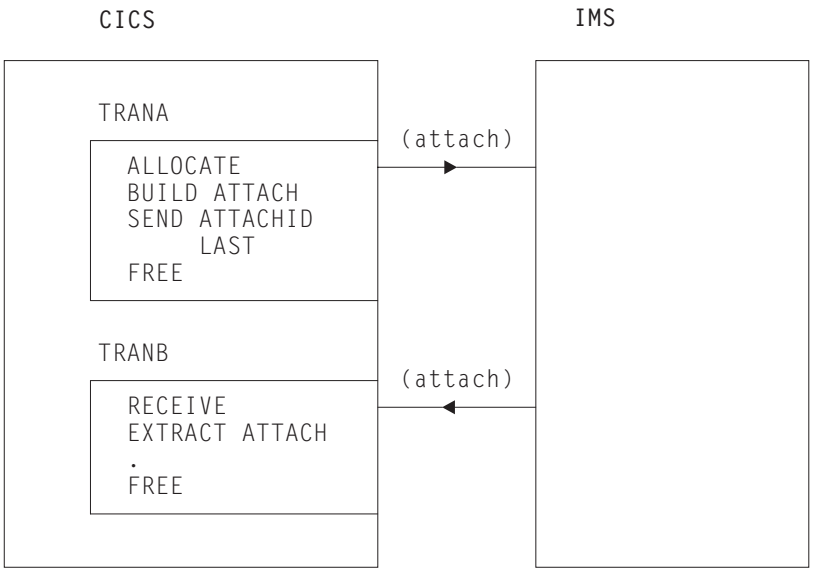


Figure 82. SEND and RECEIVE asynchronous processing – CICS front end

CICS-to-IMS applications—DTP

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 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 issued a HANDLE (which is still active) for the SYSBUSY condition. The possible combinations are shown below:

- Active HANDLE for SYSBUSY condition
  - Control is returned immediately to the label specified in the HANDLE command, whether or not you have specified NOQUEUE.

- No active 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|BASICEDTvname)]
  [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 SEND (LUTYPE6.1), RECEIVE (LUTYPE6.1), and CONVERSE (LUTYPE6.1), in the *CICS Application Programming Reference*.

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 EXEC interface block, in the *CICS Application Programming Reference*. 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 261.

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 261.

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 217.

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.

Table 11. State 1—session not allocated

STATE 1 — CICS-TO-IMS CONVERSATIONS — SESSION NOT ALLOCATED		
Commands you can issue	**What to test**	**New state**
ALLOCATE [NOQUEUE] *	SYSIDERR	1
Ditto	SYSBUSY *	1
Ditto	Otherwise (obtain session name from EIBRSRCE)	2

If you want your program to wait until a session is available, omit the NOQUEUE option of the ALLOCATE command and do not code a HANDLE command for the SYSBUSY condition.

If you want control to be returned to your program if a session is not immediately available, either specify NOQUEUE on the ALLOCATE command and test EIBRCODE for SYSBUSY (X'D3'), or code a HANDLE CONDITION SYSBUSY command.

Table 12. State 2—send state

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

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 254.

Table 13. State 3—receive pending after SEND INVITE

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

Table 14. State 4—receive state

STATE 4 — CICS-TO-IMS CONVERSATIONS — RECEIVE STATE		
Commands you can issue	**What to test**	**New state**
RECEIVE [NOTRUNCATE] *	EIBCOMPL *	—
Ditto	EIBSYNC	5
Ditto	EIBFREE	7
Ditto	EIBRECV	4
Ditto	Otherwise	2

If NOTRUNCATE is specified, a zero value in EIBCOMPL indicates that the data passed to the application by CICS is incomplete (because, for example, the data area specified in the RECEIVE command is too small). CICS 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.

Table 15. State 5—receiver take syncpoint

STATE 5 — CICS-TO-IMS CONVERSATIONS — RECEIVER TAKE SYNCPOINT		
Commands you can issue	**What to test**	**New state**
SYNCPOINT	EIBFREE (saved value)	7
Ditto	EIBRECV (saved value)	4
Ditto	Otherwise	2

Table 16. State 6—free pending after SEND LAST

STATE 6 — CICS-TO-IMS CONVERSATIONS — FREE PENDING AFTER SEND LAST		
Commands you can issue	**What to test**	**New state**
SYNCPOINT	—	7
FREE	—	1

Table 17. State 7—free session

STATE 7 — CICS-TO-IMS CONVERSATIONS — FREE SESSION		
Commands you can issue	**What to test**	**New state**
FREE	—	1

Part 5. Performance in an intersystem environment

This part of the manual gives advice on improving aspects of CICS performance in a multi-system environment. It describes how to:

- Control the length of intersystem queues
- Delete redundant shipped terminal definitions from application-owning regions and intermediate systems

Note: For information about CICS performance in general, refer to the *CICS Performance Guide*.

Chapter 24. 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 START commands” on page 42.

Overview of session queue management

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 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.

Managing allocate queues

The following sections describe three methods for managing allocate queues.

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.

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.

Note: When CICS purges an allocate request because of the QUEUELIMIT and MAXQTIME settings being exceeded, the SYSIDERR condition is returned to the application program.

For information about the QUEUELIMIT and MAXQTIME options of the CEDA DEFINE CONNECTION command, see CONNECTION definition attributes, in 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 ALLOCATE (APPC), in the *CICS Application Programming Reference*.

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. It extends the function provided by the XISCONA global user exit, which is invoked only for function shipping and DPL requests (including function shipped EXEC CICS START requests used for asynchronous processing). XZIQUE is invoked for transaction routing, asynchronous processing, and distributed transaction processing requests, as well as for function shipping and DPL. 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, both could be invoked for function shipping and DPL 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.

If you already have an XISCONA global user exit program, you could possibly modify it for use at the XZIQUE exit point.

When the XZIQUE exit is invoked

The XZIQUE global user exit is invoked, if it is enabled, at the following times:

- Whenever CICS tries to acquire a session with a remote system and there is no free session available. It is invoked whether or not you have specified the QUEUELIMIT option on the CONNECTION definition, and whether or not the limit has been exceeded. It is not invoked if the allocate request specifies NOQUEUE or NOSUSPEND.

  Requests for sessions can arise in a number of ways, such as explicit EXEC CICS ALLOCATE commands issued by DTP programs, or by transaction routing or function shipping requests.
- Whenever an allocate request succeeds in finding a free session, after the queue on the connection has been purged by a previous invocation of the exit program. In this case, your exit program can indicate that CICS is to continue processing normally, resuming queuing when necessary.

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 VTAM working-set module exits, in the *CICS Customization Guide*.

Chapter 25. Efficient deletion of shipped terminal definitions

This chapter describes how CICS deletes redundant shipped terminal definitions. It contains the following topics:

Overview of how shipped terminals are deleted

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). For brevity, we shall refer to AORs and intermediate systems collectively as "back-end systems. This is particularly necessary in the last case above, to prevent a possible mismatch between termids in the TOR and the back-end systems.

CICS 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 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.

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 Its purpose is to ensure that shipped definitions remain installed only while they are in use.

Note: Shipped definitions are not deleted if there is an automatic initiate descriptor (AID) associated with the terminal.

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 z/OS system to which terminals are shipped, you need only specify two system initialization parameters:

DSHIPIDL={020000|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={120000|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.

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 Specifying CICS system initialization parameters, in 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 CEMT INQUIRE DELETSHIPPED, CEMT PERFORM DELETSHIPPED, and CEMT SET DELETSHIPPED, in the *CICS Supplied Transactions* manual. For programming information about their EXEC CICS equivalents, see INQUIRE DELETSHIPPED, PERFORM DELETSHIPPED, and SET DELETSHIPPED, in the *CICS System Programming Reference* manual.

Tuning the performance of timeout delete

A careful choice of DSHIPINT and DSHIPIDL settings results in a minimal number of mass deletions of shipped definitions, and a scheduling of those that do take place for times when your system is lightly loaded. Conversely, a poor choice of settings could result in unnecessary mass delete operations. Here are some suggestions for coding DSHIPINT and DSHIPIDL:

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.

Attention: 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*.

Part 6. Recovery and restart in an intersystem environment

This part of the manual tells you what CICS can do if things go wrong in an intercommunication environment, and what you can do to help. The topics covered include:

- Individual session failure
- System failure and restart
- Those aspects of the CICS extended recovery facility (XRF) that affect intercommunication
- Those aspects of CICS support for VTAM persistent sessions that affect intercommunication

Chapter 26. Recovery and restart in interconnected systems

This chapter describes those aspects of CICS recovery and restart that apply particularly in the intercommunication environment. It assumes that you are familiar with the concepts of units of work (UOWs), 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 log (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 unit of work can include actions that are to be taken by two or more connected systems. Such a unit of work is known as a *distributed* unit of work, because the resources to be accessed are distributed across more than one system. A distributed unit of work is made up of two or more *local* units of work, each of which represents the work to be done on one of the participating systems. In a distributed unit of work, the participating systems must agree to commit the changes they have made; this, 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.

The rest of this chapter contains the following topics:

Terminology

Important:

1. For brevity, in this chapter, the terms *CICS Transaction Server for z/OS* and *CICS TS for z/OS* are used to mean all the following CICS products:
   - CICS Transaction Server for z/OS, Version 3 Release 2
   - CICS Transaction Server for z/OS, Version 3 Release 1
   - CICS Transaction Server for z/OS, Version 2 Release 3
   - CICS Transaction Server for z/OS, Version 2 Release 2
2. In this chapter, the terms "unit of work" and "UOW" mean a *local* unit of work—that is, *that part of a distributed unit of work that relates to resources on the local system*.

The system programming and CEMT commands and the CICS messages described later in the chapter return information about local UOWs.

Where a *distributed* unit of work is meant, the term is used explicitly.

This chapter introduces a number of new terms, such as *in-doubt*, *initiator*, *agent*, *coordinator*, *subordinate*, *shunted*, and *resynchronization*. These terms are explained as they occur in the text, with examples. You may also find it useful to refer to the CICS glossary.

The rest of the chapter contains the following sections:

- "Syncpoint exchanges" gives examples of CICS syncpoint flows, and explains the terms used to describe them.
- "Recovery functions and interfaces" on page 281 describes the ways in which CICS can recover from a communication failure, and the commands you can use to control CICS recovery actions. Note that this and the next two sections apply only to MRO, IPIC, and ISC over SNA (APPC) parallel-session connections to other CICS Transaction Server for z/OS (CICS TS for z/OS) systems.
- "Initial and cold starts" on page 285 describes the effect of initial and cold starts on inter-connected systems, and how to decide when a cold start is possible.
- "Managing connection definitions" on page 288 describes how safely to modify or discard MRO, IPIC, and ISC over SNA (APPC) parallel-session connections to other CICS TS for z/OS systems.
- "Connections that do not fully support shunting" on page 290 describes exceptions that apply to connections other than MRO, IPIC, or ISC over SNA (APPC) parallel-session links to other CICS TS for z/OS systems.
- "Problem determination" on page 292 describes the messages that CICS may issue during a communication failure and recovery, and contains examples of how to resolve in-doubt and resynchronization failures.

Syncpoint exchanges

Consider the following example:

Syncpoint example:

An order-entry transaction is designed so that, when an order for an 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.***
3. ***A synchronization point is taken to indicate the end of the current UOW.***

In a single CICS system, the syncpoint causes steps 1 and 2 both 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 83 on page 280.

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.

Syncpoint flows

The ways in which syncpoint requests and responses are exchanged on intersystem conversations are defined in the APPC and LUTYPE6.1 architectures. CICS MRO and IPIC use the APPC recovery protocols. Although the formats of syncpoint flows for APPC and LUTYPE6.1 are different, the concepts of syncpoint exchanges are similar.

In CICS, the flows involved in syncpoint exchanges 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 help 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*.

Figures Figure 83 on page 280 through Figure 85 on page 281 show some examples of syncpoint flows. In the figures, the numbers in brackets, for example, (1), show the sequence of the actions in each flow.

A CICS task may contain one or more UOWs. A local UOW that initiates syncpoint activity—by, for example, issuing an EXEC CICS SYNCPOINT or an EXEC CICS RETURN command—is called an **initiator**. A local UOW that receives syncpoint requests from an initiator is called an **agent**. The simplest case is shown in Figure 83 on page 280. There is a single conversation between an initiator and an agent. 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 unit of work is complete. However, the agent retains recovery information about the UOW until its partner tells it (by means of a “forget” flow) that the information can be discarded.

Between the commit flow and the committed flow, the initiator is in-doubt, but the agent is not. The local UOW that is not in-doubt is called the **coordinator**, because it coordinates the commitment of resources on both systems. The local UOW that is in-doubt is called the **subordinate**, because it must obey the decision to commit or

back out taken by its coordinator.

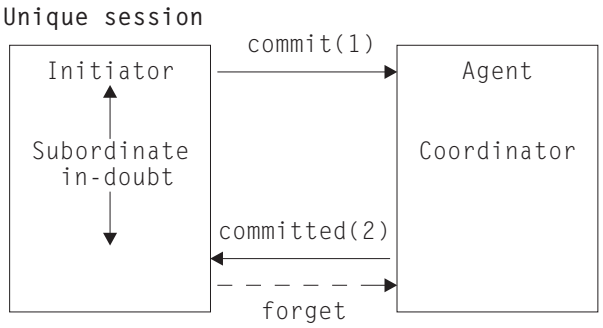


Figure 83. Syncpointing flows—unique session. In this distributed UOW, there is one coordinator and one subordinate. The coordinator is not in-doubt.

Figure 84 shows a more complex example. Here, the agent UOW (Agent1) has a conversation with a third local UOW (Agent2). Agent1 initiates syncpoint activity on this latter conversation before it responds to the initiator. Agent2 commits first, then Agent1, and finally the initiator. Note that, in Figure 84, Agent1 is both the coordinator of the initiator and a subordinate of Agent2.

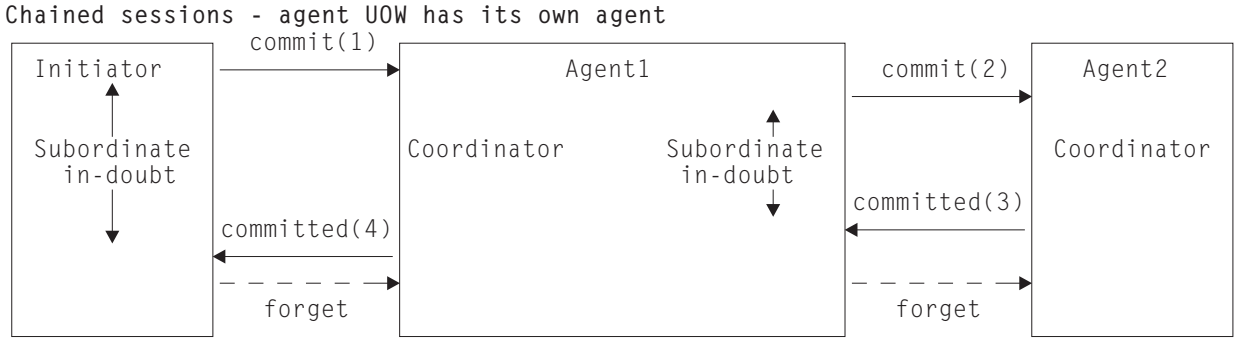


Figure 84. Syncpointing flows—chained sessions. In this distributed UOW, Agent1 is both the coordinator of the initiator, and a subordinate of Agent2.

Figure 85 on page 281 shows a more general case, in which the initiator UOW has more than one (directly-connected) agent. It must inform each of its agents that a syncpoint is being taken. It does this by sending a “prepare to commit” request to all of its agents except the last. The **last agent** is the agent that is not told to prepare to commit.

Note: CICS chooses the last agent dynamically, at the time the syncpoint is issued. CICS external interfaces do not provide a means of identifying the last agent.

Each agent that receives a “prepare” request responds with a “commit” request. When all such “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.

Note that, in Figure 85 on page 281, the Initiator is both the coordinator of Agent1 and a subordinate of Agent2. Agent2 is the last agent.

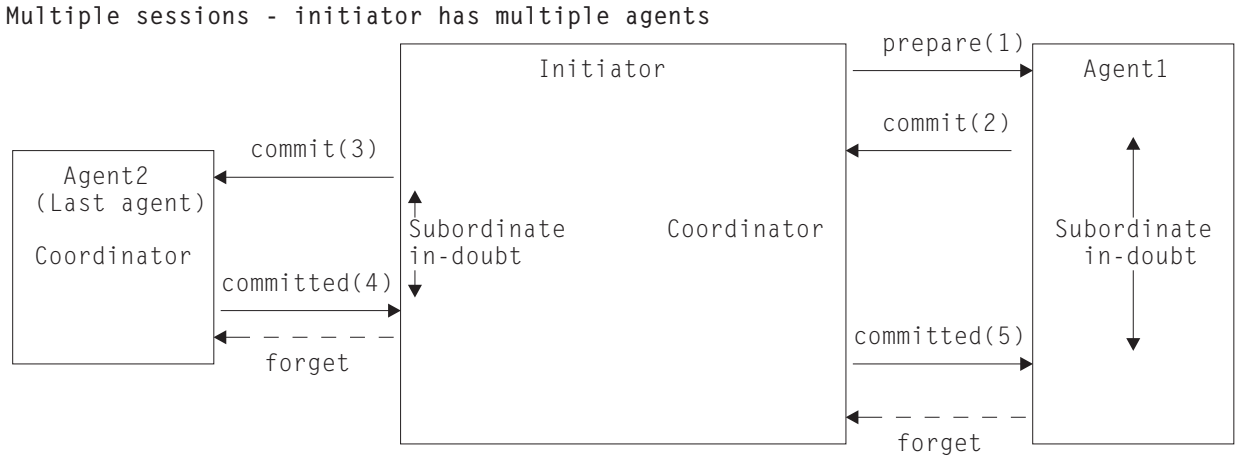


Figure 85. Syncpointing flows—multiple sessions. In this distributed UOW, the Initiator is both the coordinator of Agent1, and a subordinate of Agent2. Agent2 is the last agent, and is therefore not told to prepare to commit.

Recovery functions and interfaces

This section describes the functions and interfaces provided by CICS for recovery after a communication failure, or a CICS system failure.

Important:

Not all CICS releases provide the same level of support; this section describes MRO, IPIC, and ISC over SNA (APPC) parallel-session connections to other CICS Transaction Server for z/OS systems. Much of it applies also to other types of connection, but with some restrictions. For information about the restrictions for connections to non-CICS Transaction Server for z/OS systems, and for LU6.1 and APPC single-session connections, see “Connections that do not fully support shunting” on page 290.

This section also assumes that each CICS system is restarted correctly (that is, that AUTO is coded on the START system initialization parameter). If an initial start is performed there are implications for connected systems; these are described in “Initial and cold starts” on page 285.

Recovery functions

If CICS is left in-doubt about a unit of work due to a communication failure, it can do one of two things (how you can influence which of the two actions CICS takes is described in “The in-doubt attributes of the transaction definition” on page 282):

- Suspend commitment of updated resources until the systems are next in communication. The unit of work is **shunted**. When communication is restored, the decision to commit or back out is obtained from the coordinator system; the unit of work is **unshunted**, and the updates are committed or backed out on the local system in a process called **resynchronization**.
- Take a unilateral decision to commit or back out local resources. In this case, the decision may be inconsistent with other systems; at the restoration of

communications the decisions are compared and CICS warns of any inconsistency between neighboring systems (see “Messages that report CICS recovery actions” on page 292).

There is a trade-off between the two functions: the suspension of in-doubt UOWs causes updated data to be locked against subsequent access; this disadvantage has to be weighed against the possibility of corruption of the consistency of data, which could result from taking unilateral decisions. When unilateral decisions are taken, there may be application-dependent processes, such as reconciliation jobs, that can restore consistency, but there is no general method that can be provided by CICS.

Recovery interfaces

This section summarizes the resource definition options, system programming commands, and CICS-supplied transactions that you can use to control and investigate units of work that fail during the in-doubt period. For definitive information about defining resources, system programming commands, and CEMT transactions, see CICS Transaction Server for z/OS Resource Definition Guide, in the *CICS Resource Definition Guide*, CICS Transaction Server for z/OS System Programming Reference, in the *CICS System Programming Reference* manual, and CICS Transaction Server for z/OS Supplied Transactions, in the *CICS Supplied Transactions* manual, respectively.

The in-doubt attributes of the transaction definition

You can control the action that CICS takes after a communication failure during the in-doubt period by specifying in-doubt attributes when you define the transaction, using the WAIT, WAITTIME, and ACTION options of the TRANSACTION definition. These options are honored when communication is lost with the coordinator and the UOW is in the in-doubt period.

WAIT({YES|NO})
Specifies whether or not a unit of work is to wait, pending recovery from a failure that occurred after it had entered the in-doubt period, before taking the action specified by ACTION.

YES
The UOW is to wait, pending recovery from the failure, to resolve its in-doubt state and determine whether recoverable resources are to be backed out or committed. In other words, it is to be shunted.

NO The UOW is not to wait. CICS takes immediately whatever action is specified on the ACTION attribute.

Note: The setting of the WAIT option can be overridden by other system settings—see the description of DEFINE TRANSACTION in TRANSACTION definition attributes, in the *CICS Resource Definition Guide*.

WAITTIME({00,00,00|dd,hh,mm})
Specifies, if WAIT=YES, how long the transaction is to wait, before taking the action specified by ACTION.

You can use WAIT and WAITTIME to allow an opportunity for normal recovery and resynchronization to take place, while ensuring that a unit of work releases locks within a reasonable time.

ACTION({BACKOUT|COMMIT})
Specifies the action to be taken when communication with the coordinator of the unit of work is lost, and the UOW has entered the in-doubt period.

BACKOUT
All changes made to recoverable resources are backed out, and the resources are returned to the state they were in before the start of the UOW.

COMMIT
All changes made to recoverable resources are committed and the UOW is marked as completed.

The action is dependent on the WAIT attribute. If WAIT specifies YES, ACTION has no effect unless the interval specified on the WAITTIME option expires before recovery from the failure.

Whether you specify BACKOUT or COMMIT is likely to depend on the kinds of changes that the transaction makes to resources in the remote system—see "Specifying in-doubt attributes—an example."

Specifying in-doubt attributes—an example: As an illustration of specifying the in-doubt attributes of a transaction, 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 by updating the stock file, and sends a record to a remote transient data queue to initiate the dispatch of the part.

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 update 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. 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.

To deal with a communications failure—for example, a failure of the remote system—during the in-doubt period, specify on the local transaction definition, WAIT(YES), ACTION(BACKOUT), and a WAITTIME long enough to allow the remote system to be recycled. This enables resynchronization to take place automatically, if communication is restored within the specified time limit. During the WAITTIME period, until resynchronization takes place, the local UOW is shunted, and a lock is held on the stock-file record.

If communication is not restored within the time limit, changes made to the stock file on the local system are backed out. The addition to the TD queue on the remote system may or may not have been committed; this must be investigated after communication is restored.

INQUIRE commands

The CEMT and EXEC CICS interfaces provide a set of inquiry commands that you can use to investigate the execution of distributed units of work, and diagnose problems.

Note: In the following list of commands, **INQUIRE CONNECTION** applies to MRO and ISC over SNA (APPC) connections. **INQUIRE IPCONN** applies to IPIC connections.

In summary, the commands are:

INQUIRE {CONNECTION | IPCONN} RECOVSTATUS
Use it to find out whether any resynchronization work is outstanding between the local system and the connected system. The returned CVDA values are:

NORECOVDATA
Neither side has recovery information outstanding.

NOTAPPLIC
This is not an IPIC, APPC parallel-session, nor a CICS-to-CICS MRO connection, and does not support two-phase commit protocols.

NRS CICS does not have recovery outstanding for the connection, but the partner may have.

RECOVDATA
There are in-doubt units of work associated with the connection, or there are outstanding resyncs awaiting FORGET on the connection. Resynchronization takes place when the connection next becomes active, or when the UOW is unshunted.

INQUIRE {CONNECTION | IPCONN} PENDSTATUS
Use it to discover whether there are any UOWs for which resynchronization is impossible because of an initial start by the connected system.

INQUIRE CONNECTION XLNSTATUS (APPC parallel-sessions only)
Use it to discover whether the link is currently able to support syncpoint (synclevel 2) work. See “The exchange lognames process” on page 287 for more information.

Note: XLNSTATUS is not applicable to IPCONNs.

INQUIRE UOW
Use it to discover why a unit of work is waiting or shunted. If the reason is a connection failure (the WAITCAUSE option returns a CVDA value of CONNECTION), the SYSID and LINK options return the sysid and netname of the remote system that caused the UOW to wait or be shunted.

Note that INQUIRE UOW returns information about a *local* UOW—that is, for a distributed UOW it returns information only about the work required on the local system. You can assemble information about a distributed UOW by matching the network-wide identifier returned in the NETUOWID field against the identifiers of local UOWs on other systems. For an example of how to do this, see “Resolving a resynchronization failure” on page 300.

INQUIRE UOWLINK
This command allows you to inquire about the resynchronization needs of individual UOWs. Use it to discover information about connections involved in a distributed UOW.

For a local UOW, INQUIRE UOWLINK returns a list of tokens (*UOW-links*) representing connections to the systems that are involved in the distributed UOW. For each UOW-link, INQUIRE UOWLINK returns:

- The CONNECTION name
- The resynchronization status of the connection

- Whether the connection is to a coordinator or a subordinate system.

For examples of the use of these commands to diagnose problems with distributed units of work, see “Problem determination examples” on page 295.

SET {CONNECTION | IPCONN} command

In exceptional cases, you may need to override the in-doubt action normally controlled by the transaction definition. For example, a connected system may take longer than expected to restart. If the connected system is the coordinator of any UOWs, you can use the EXEC CICS or CEMT SET {CONNECTION | IPCONN} UOWACTION(FORCE|COMMIT|BACKOUT) command to force the UOWs to take a local, unilateral decision to commit or back out.

Note: **SET CONNECTION** applies to MRO and ISC over SNA (APPC) connections. **SET IPCONN** applies to IPIC (IP) connections.

The following commands are described in “The exchange lognames process” on page 287 and “Managing connection definitions” on page 288:

- SET {CONNECTION | IPCONN} PENDSTATUS
- SET {CONNECTION | IPCONN} RECOVSTATUS.

Initial and cold starts

This section describes functions to manage the exceptional conditions that can occur in a transaction-processing network when one system performs an initial or cold start.

Important:

- Except where otherwise stated, this section describes the effect of initial and cold starts on CICS Transaction Server for z/OS systems that are connected by MRO, IPIC, or ISC over SNA (APPC) parallel-session links. For information about the effects when other connections are used, see “Connections that do not fully support shunting” on page 290.
- In the rest of this chapter, the term “cold start” means a cold start in the CICS TS for z/OS meaning of the phrase (explained below). Where an “initial start” is intended, the term is used explicitly.

CICS Transaction Server for z/OS systems can be started without full recovery in two ways:

Initial start

An *initial start* may be performed in either of the following circumstances:

- 'INITIAL' is specified on the START system initialization parameter.
- 'AUTO' is specified on the START system initialization parameter, and the recovery manager utility program, DFHRMUTL, has been used to set the AUTOINIT autostart override in the global catalog.

On an initial start, all information about both local and remote resources is erased, and all resource definitions are reinstalled from the CSD or from CICS tables.

An initial start should be performed only in exceptional circumstances. Examples of times when an initial start is appropriate are:

- When bringing up a new CICS system for the first time

- After a serious software failure, when the global catalog or system log has been corrupted.

Cold start

A *cold start* may be performed in either of the following circumstances:

- 'COLD' is specified on the START system initialization parameter.
- 'AUTO' is specified on the START system initialization parameter, and the DFHRMUTL utility has been used to set the AUTOCOLD autostart override in the global catalog.

In CICS TS for z/OS, a cold start means that log information about *local* resources is erased, and resource definitions are reinstalled from the CSD or from CICS tables. However, resynchronization information relating to remote systems or to RMI-connected resource managers is preserved. The CICS log is scanned during startup, and information regarding unit of work obligations to remote systems, or to non-CICS resource managers (such as DB2®) connected through the RMI, is preserved. (That is, any decisions about the outcome of local UOWs, needed to allow remote systems or RMI resource managers to resynchronize their resources, are preserved.)

For guidance information about the different ways in which CICS can be started, see the *CICS Recovery and Restart Guide*.

Deciding when a cold start is possible

At a cold start, information relating to intersystem recovery is read from the system log. Connected systems act as if the local system restarted normally, and resynchronize any outstanding work. Note that updates to *local* resources that were not fully committed or backed out during the previous run of CICS are not recovered at a cold start, *even if the updates were part of a distributed unit of work*.

A cold start will not damage the integrity of data if **all** the following conditions are true:

1. *Either*
   - The local system has no local recoverable resources (a TOR, for example), *or*
   - The previous run of CICS was successfully quiesced (shutdown was normal rather than immediate) and no units of work were shunted.

     Note: On a normal shutdown, CICS issues messages to help you decide whether it can be cold started safely. If there are no shunted UOWs, CICS issues message DFHRM0204. If there *are* shunted UOWs, it issues message DFHRM0203—you should not perform a cold start.
2. Attached resource managers that use the RMI are subsequently reconnected to allow resynchronization.
3. Connections to remote systems required for resynchronization are subsequently acquired.

   The cold-started system may or may not contain the same connection definitions that were in use at the previous shutdown. If autoinstalled connections are missing, the remote system may cause them to be recreated, in which case resynchronization takes place. If this does not happen—or if CEDA- or GRPLIST- installed definitions are missing—some action must be taken. See “Managing connection definitions” on page 288.

   If you have defined the cold-started system to be part of a VTAM generic resource group, its connections can be correctly reestablished, provided the

affinity relationship maintained by VTAM is still valid. However, the loss of autoinstalled definitions may make it difficult to end VTAM affinities, if this is required. See “APPC connections to and from VTAM generic resources” on page 289.

The DFHRMUTL utility returns information about the type of the last CICS shutdown which is of use in determining whether a cold restart is possible or not. For further details, see the *CICS Operations and Utilities Guide*.

The exchange lognames process

The protocols that control the communication of syncpointing commit and backout decisions depend on information in the system log. Each time CICS systems connect they exchange tokens called **lognames**. Lognames are verified during resynchronization; an *exchange lognames failure* means that the recovery protocol has been corrupted. A failure can take two forms:

1. A **cold/warm log mismatch**. A cold/warm log mismatch is caused by the loss of log data at one partner when the other has resynchronization work outstanding.

   Note: The term “cold start” is used in the *SNA Peer Protocols* manual, and by other products that communicate with CICS TS for z/OS to describe the cause of a loss of log data.

   “Cold start” is also used in CICS TS for z/OS messages and interfaces to describe the action of a partner system that results in a loss of log data for CICS TS for z/OS.

   However, in CICS TS for z/OS, a loss of log data for connected systems is caused by an *initial* start (not by a cold start), or by a SET CONNECTION NORECOVDATA command.
2. A **lognames mismatch**. A lognames mismatch is caused by a corruption of logname data. This can occur due to:
   a. A system logic error
   b. An operational error—for example, a failure to perform an initial start when migrating from a back-level CICS release to CICS Transaction Server for z/OS.

The exchange lognames process is defined by the APPC architecture. For a full
description of the concepts and physical flows, see the *SNA Peer Protocols manual*.
MRO and IPIC use a similar protocol to APPC, with the important difference that
after the erasure of log information at a partner, they allow new work to begin
whatever the condition of existing work. On APPC synclevel 2 sessions, no further
work is possible until action has been taken to delete any outstanding
resynchronization work.

After a partner system has been reconnected, you can use the INQUIRE CONNECTION PENDSTATUS command to check whether there is any outstanding resynchronization work that has been invalidated by the erasure of log information at the partner. A status of 'PENDING' indicates that there is. To check whether APPC connections are able to execute new synclevel 2 work, use the INQUIRE CONNECTION XLNSTATUS command. A status of 'XNOTDONE' indicates that the exchange lognames process has not completed successfully, probably because of a loss of log data.

When CICS detects that a partner system has lost log data, the possible actions it can take are:

1. None. If there is no resynchronization work outstanding on the local system, the loss of log data has no effect.
2. Keep outstanding resynchronization work (which may include UOWs which were in-doubt when communication was lost) for investigation.
3. Delete outstanding resynchronization work; any in-doubt UOWs are committed or backed out according to the ACTION option of the associated transaction definition, and any decisions remembered for the partner are forgotten.

When there is outstanding resynchronization work, you can control (for IPIC, MRO and APPC connections) which of actions 2 or 3 CICS takes:

- **Automatically**, using the XLNACTION option of the connection definition. To delete resynchronization work as soon as the loss of log data by the partner is detected, use XLNACTION(FORCE).
- **Manually**, using the SET UOW and SET CONNECTION PENDSTATUS(NOTPENDING) commands.

Considerations for APPC connections

The exchange lognames process affects only level 2 synchronization conversations. If it fails, synclevel 2 conversations are not allowed on the link until the failure is resolved by operator action. However, synclevel 0 and synclevel 1 traffic on the link is unaffected by the failure, and continues as normal.

Managing connection definitions

Important:

This section describes how to manage definitions of MRO, IPIC, and APPC parallel-session connections between CICS Transaction Server for z/OS systems. For considerations that apply to other types of connection, see “Connections that do not fully support shunting” on page 290.

Recovery information for a remote system is largely independent of the connection definition for the system. This allows you to manage (for example, modify) connection definitions independently of any recovery information that may be outstanding. However, in some cases the connection definition holds important information, which means that it must be kept, unmodified, until any recovery between the systems is complete.

MRO connections to CICS TS for z/OS systems

For connections to other CICS Transaction Server for z/OS systems, the connection definition contains no recovery information. You can modify connections without regard to recovery, provided that the netname of the connection remains the same.

If a connection definition is lost at a cold start, use the CEMT INQUIRE UOWLINK RESYNCSTATUS(UNCONNECTED) command to discover whether CICS retains any recovery information for the previously-connected system. This command will tell you whether CICS contains any tokens (**UOW-links**) associating UOWs with the lost connection definition. If there are UOW-links present, you can either:

- Reinstall a suitable connection definition based on the UOW-link attributes and reestablish the connection.

- If you are certain that the associated UOW information is of no use, use the SET UOWLINK(xxxxxxx) ACTION(DELETE) command to delete the UOW-link. (You may need to use the SET UOW command to force an in-doubt UOW to commit or back out before the UOW-links can be deleted.)

You can use the same commands if a connection has been discarded.

Note: Before discarding a connection, you should use the INQUIRE CONNECTION RECOVSTATUS command to check whether there is any recovery information outstanding. If there is, you should discard the connection *only if there is no possibility of achieving a successful resynchronization with the partner*. In this exceptional circumstance, you can use the SET CONNECTION UOWACTION command to force in-doubt units of work before discarding the connection.

IPIC connections to CICS TS for z/OS systems

IPIC connections contain no recovery information and can be managed in the same way as MRO connections to CICS TS for z/OS systems.

APPC parallel-session connections to CICS TS for z/OS systems

APPC parallel-session connections in a CICS Transaction Server for z/OS system that is not registered as a member of a VTAM generic resource contain no recovery information and can be managed in the same way as MRO connections to CICS TS for z/OS systems.

APPC connections to and from VTAM generic resources

If CICS is a member of a VTAM generic resource group, the local VTAM may have an affinity which directs any new binds from a partner to this same local system. You must not end the affinity held by VTAM if there is any possibility that resynchronization with the partner may be needed; if you do, binds (and subsequent resynchronization messages) may be directed to a different member of the generic resource. In most cases, it is safest to allow the APPC connection quiesce protocol to end the affinities automatically—see “APPC connection quiesce processing” on page 292.

CICS prevents the execution of the SET CONNECTION ENDAFFINITY command if a logname has been received from the partner system, because this is the condition under which the partner may begin recoverable work and start resynchronization. The discarding of a connection is also prevented, because its loss means that the logname is no longer visible. If you intend ending affinities, you should do it *before* shutting down CICS prior to a cold start, because a cold start restores a logname without the associated connection. Ending affinities without removing the logname can cause exchange logname failures later.

For further information about affinities and how to end them, see “Ending affinities” on page 131.

Managing connection definitions

For members of a generic resource, the connection definition is the only way (using the INQUIRE and SET CONNECTION RECOVSTATUS commands) of safely managing lognames and affinities. Connections can be discarded only if their recovery status (RECOVSTATUS) is NORECOVDATA. You can use the SET CONNECTION RECOVSTATUS command to set a connection's recovery status to NORECOVDATA if neither the local system nor the partner has any in-doubt units of work dependent on the other. A simple and safe test is that neither system's

connection to the other should have a status of RECOVSTATUS(RECOVDATA). If this test succeeds, you can issue SET CONNECTION NORECOVDATA on both, and SET CONNECTION ENDAFFINITY on the generic resource members.

Connections that do not fully support shunting

The information in previous sections assumes that you are using MRO, IPIC, or APPC parallel-session connections to other CICS Transaction Server for z/OS systems—that is, that your network consists solely of current systems that fully support shunting. Much of the preceding information applies equally to other types of connection. This section describes exceptions that apply, for example, to connections to back-level systems.

LU6.1 connections

This section describes the ways in which LU6.1 connections differ from APPC parallel-session connections and MRO connections to CICS TS for z/OS systems.

Recovery functions and interfaces

Some recovery functions are not available to LU6.1 connections:

- Shunting is not always supported.
- Some recovery-related commands and options are not supported.
- Resynchronization takes place on a session-by-session basis.

Restriction on shunting support

There is no LU6.1 protocol by which one system can notify another system that a unit of work has been shunted. The only time when a UOW that includes an LU6.1 session can be shunted is when all the following are true:

- There is only one LU6.1 session in the local UOW.
- The LU6.1 session is the coordinator.
- The LU6.1 session has failed during the in-doubt period.
- The LU6.1 session is to the last agent.

Under these conditions, the UOW can be shunted, because there is no need for the LU6.1 partner to be notified of the shunt.

Under other conditions, a UOW that fails in the in-doubt period, and that involves an LU6.1 session, takes a unilateral decision. If WAIT(YES) is specified on the transaction definition, it has no effect—WAIT(NO) is forced.

Unsupported commands

The following commands are not supported on LU6.1 connections:

- INQUIRE CONNECTION PENDSTATUS
- INQUIRE CONNECTION RECOVSTATUS
- INQUIRE CONNECTION XLNSTATUS.

Lack of SYNCPOINT ROLLBACK support

There is no LU6.1 protocol by which one system can notify another that a UOW has been backed out, without terminating the conversation. An attempt to issue an EXEC CICS SYNCPOINT ROLLBACK command in a UOW that includes an LU6.1 session results in an ASP8 abend. This abend cannot be handled by the application program.

Any resources in the UOW are backed out, but the transaction is not able to continue.

Session-by-session resynchronization
Unlike APPC parallel-session connections and CICS TS for z/OS-CICS TS for z/OS MRO connections, LU6.1 sessions are resynchronized one by one, as they are bound. Therefore, any UOW that requires resynchronization is not resynchronized until the session that failed is reconnected.

Initial and cold starts

The LU6.1 connection definition contains sequence numbers used for recovery. If you perform an initial or cold start of CICS when there are LU6.1 connections on which recovery is outstanding, the sequence numbers are lost, and it becomes impossible for the partner systems to resynchronize their outstanding units of work.

Lognames are not used. Therefore, the XLNACTION option of the CEDA DEFINE CONNECTION command is meaningless for LU6.1 connections.

Managing connection definitions

Recovery information for a remote system is *not* stored independently from the connection definition for the system—the LU6.1 connection definition contains sequence numbers used for recovery. Therefore you should not modify or discard connections for which recovery information may be outstanding.

APPC connections to non-CICS TS for z/OS systems

Some non-CICS Transaction Server for z/OS systems that can be connected to by APPC links do not support shunting, and always take unilateral action if a session failure occurs during the in-doubt period. Inevitably, communication with a system that does not support shunting involves a risk of damage to data integrity through the taking of unilateral decisions. It is not possible for CICS to distinguish systems that do not support shunting from others that *do* support shunting. Therefore, it cannot preferentially select such a system to be the coordinator of a unit of work.

Note the following:

- When unshunting takes place, there may be some delay before the unshunting is communicated to the non-CICS TS for z/OS system.
- Sessions may be unbound by CICS or its partner system as a normal part of the shunting and resynchronization process.

APPC single-session connections

Normal syncpoint protocols cannot be used across a connection that is defined as SINGLESESS(YES). If function shipping is used (inbound or outbound), CICS communicates the outcome of a unit of work as described in Syncpointing (LU 6.2), in the *CICS Family: Communicating from CICS on zSeries* manual. However, resynchronization cannot be performed in the case of session failure.

CICS issues a message to inform you of the shunting—but not the unshunting— of a unit of work.

If the connection to which a function-shipped request is made is defined as remote (that is, it is owned by a remote region), the connection to the remote region must be defined as a parallel-session link, if recovery protocols with the resource-owning system are to be enabled.

APPC connection quiesce processing

When an APPC parallel-session connection with a CICS Transaction Server for z/OS region is shut down normally, CICS exchanges information with its partner to discover if there is any possibility that resynchronization is required when the connection is restarted. This exchange is known as the connection quiesce protocol (CQP).

CICS determines that resynchronization is not required if all the following conditions are true:

- The connection is being shut down.
- There are no user sessions active (the CQP uses the SNASVCMG sessions). If the SNASVCMG sessions become inactive before the user sessions, the CQP will not take place.
- The CICS recovery manager domain has no record of outstanding syncpoint work or resynchronization work for the connection.

Once the CQP has completed, CICS ensures that no recoverable work can be initiated for the connection until a fresh logname exchange has taken place.

If the CQP determines that resynchronization is not required, CICS:

- Sets the connection's recovery state to NORECOVDATA
- If CICS is a member of a generic resource group, ends any affinity held by VTAM and issues a message to say that the affinity has been ended.

If there is any failure of the CQP, CICS presumes that there is a possibility of resynchronization being necessary. You may use the procedures described in this chapter to determine if this is truly the case, and perform the necessary actions manually. Alternatively, you can reacquire the connection and release it again, to force CICS to re-attempt the CQP.

Problem determination

This section describes:

- Messages that report CICS recovery actions
- Examples of how to resolve in-doubt and resynchronization failures, which demonstrate how to use some of the commands previously discussed.

Messages that report CICS recovery actions

In the event of a communications failure, the connected systems may resolve their local parts of a distributed unit of work in ways that are inconsistent with each other. To warn of this possibility, when a CICS region loses communication with a partner, for each session on which the UOW is in the in-doubt period, it issues a DFHRMxxxx message. The message may appear at the time of a session failure, a failure of the partner, or during emergency restart.

When the connection has been reestablished, on each affected session the UOW is unshunted, its state is determined, and another message is issued. For LUTYPE6.1 conversations, these messages may appear only on the initiator side.

All messages contain the following information, which enables them to be correlated:

- The time and date of the original failure

- The transaction identifier and task number
- The netname of the remote system
- The operator identifier
- The operator terminal identifier
- The network-wide unit of work identifier
- The local unit of work identifier.

The messages associated with intersystem session failure and recovery are shown in three figures. Table 18 and Table 19 on page 294 show the messages that can be produced when contact is lost with the coordinator of the UOW: Table 18 shows the messages produced when WAIT(YES) is specified on the transaction definition and shunting is possible; Table 19 on page 294 shows the messages produced when WAIT(NO) is specified, or when shunting is not possible. Table 20 on page 295 shows the messages produced when contact is lost with a subordinate in the UOW. Full details are in the *CICS Messages and Codes* manual.

Table 18. Session failure messages. The failed session is to the coordinator of the UOW, WAIT(YES) is specified on the transaction definition and shunting is possible.

In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply (shown in columns 2 and 4).

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	Session failure	DFHRM0106	Intersystem session failure. Resource changes will not be committed or backed out until session recovery.
Stage 1	System failure/restart	—	—
Stage 2	Session recovery successful.	DFHRM0108	Intersystem session recovery. Suspended resource changes now being committed.
Stage 2	Session recovery successful.	DFHRM0109	Intersystem session recovery. Suspended resource changes now being backed out.
Stage 2	Wait time exceeded or SET UOW ACTION issued.	DFHRM0104 DFHRM0105	See next table.
Stage 2	SET CONNECTION NOTPENDING ¢ or XLNACTION (FORCE) ¢ or NORECOVDATA $ issued.	DFHRM0125 DFHRM0126	Local resources committed or backed out.
Stage 2	Session recovery after a cold start of local resources.	DFHRM0209	UOW backed out.
Stage 2	Session recovery after a cold start of local resources.	DFHRM0208	UOW committed.

Table 18. Session failure messages (continued). The failed session is to the coordinator of the UOW, WAIT(YES) is specified on the transaction definition and shunting is possible.

In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply (shown in columns 2 and 4).

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 2	Session recovery error — for example, partner cold-started. *	DFHRM0112 DFHRM0113 DFHRM0115 DFHRM0116 DFHRM0118 DFHRM0119 DFHRM0121 DFHRM0122	Intersystem recovery error. Local resource changes are committed or backed out.
Key: • * LU6.1 only • ¢ MRO, IPIC ??????????????????????, and APPC only • $ APPC and IPIC ??????????????????????? only			

Table 19. Session failure messages. The failed session is to the coordinator of the UOW. WAIT(NO) is specified on the transaction definition or shunting is not possible.

In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply (shown in columns 2 and 4).

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	Session failure	DFHRM0104 DFHRM0105	Intersystem session failure. Resource changes are being committed or backed out and may be out of sync with partner.
Stage 1	System failure/restart	—	—
Stage 2	Session recovery successful.	DFHRM0110	Intersystem session recovery. Resource updates found to be synchronized.
Stage 2	Session recovery successful.	DFHRM0111	Intersystem session recovery. Resource updates found to be out of sync.
Stage 2	SET CONNECTION NOTPENDING ¢ or XLNACTION (FORCE) ¢ or NORECOVDATA $ issued.	DFHRM0127	SET NOTPENDING issued.
Stage 2	Session recovery error — for example, partner cold-started. *	DFHRM0112 DFHRM0113 DFHRM0115 DFHRM0116 DFHRM0118 DFHRM0119 DFHRM0121 DFHRM0122	Local resource changes committed or backed out.
Key: • * LU6.1 only • ¢ MRO, IPIC ??????????????????????, and APPC only • $ APPC and IPIC ??????????????????????? only			

Table 20. Session failure messages. The failed session is to a subordinate in the UOW.

In each stage (1 and 2), the messages that CICS issues depend on the circumstances that apply (shown in columns 2 and 4).

Sequence of messages	Circumstances	Messages issued	Meaning of messages
Stage 1	UOW shunted due to failure of session to coordinator ¢	—	—
Stage 1	Session failure	DFHRM0107	Intersystem session failure. Notification of decision may not reach the remote system.
Stage 1	System failure/restart	—	—
Stage 2	Session recovery successful.	DFHRM0135 DFHRM0148 +	Intersystem session recovery. Resource updates found to be synchronized.
Stage 2	Session recovery successful.	DFHRM0110	Intersystem session recovery. Resource updates found to be synchronized, after a unilateral decision on the remote system.
Stage 2	Session recovery successful.	DFHRM0111 DFHRM0124 +	Intersystem session recovery. Resource updates found to be out of sync, after a unilateral decision on the remote system.
Stage 2	SET CONNECTION NOTPENDING ¢ or XLNACTION (FORCE) ¢ or NORECOVDATA $ issued.	DFHRM0127	SET NOTPENDING issued.
Stage 2	Session recovery error — for example, partner cold-started. *	DFHRM0114 DFHRM0117 DFHRM0120 DFHRM0123	Intersystem session recovery error. Resource changes may be out of sync.

Key:
- * LU6.1 only
- ¢ MRO, IPIC ??????????????????????, and APPC only
- $ APPC and IPIC ??????????????????????? only
- + DFHRM0124 and DFHRM0148 may occur without a preceding session failure message (DFHRM0107) or shunt.

Problem determination examples

This section contains examples of how to resolve in-doubt and resynchronization failures.

Resolving an in-doubt failure

This section is an example of how to resolve a unit of work that fails during the in-doubt period. It uses the following commands:

- CEMT INQUIRE TASK
- CEMT INQUIRE UOWENQ
- CEMT INQUIRE UOW
- CEMT INQUIRE UOWLINK

- CEMT INQUIRE CONNECTION

A user reports that their task has hung on region IYM51. A CEMT INQUIRE TASK command shows the following:

```
INQUIRE TASK
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Tas(0000061) Tra(RTD1) Fac(S254) Sus Ter Pri( 001 )
    Sta(TO) Use(CICSUSER) Uow(AB1DF09A54115600) Hty(ENQUEUE ) Hva(TDNQ    )
 Tas(0000064) Tra(CEMT) Fac(S255) Run Ter Pri( 255 )
    Sta(TO) Use(CICSUSER) Uow(AB1DF16E3B78B403)
```

The hanging task is 61, tranid RTD1. It is waiting on an enqueue for a transient data resource. A CEMT INQUIRE UOWENQ command shows:

```
INQUIRE UOWENQ
STATUS:  RESULTS
 Uow(AB1DF0804B0F5801) Tra(RFS4) Tas(0000060) Ret Tsq Own
    Res(RMLTSQ                   ) Rle(008) Enq(00000000)
 Uow(AB1DF0804B0F5801) Tra(RFS4) Tas(0000060) Ret Dat Own
    Res(DCXISCG.IYLX1.RMLFILE    ) Rle(021) Enq(00000000)
 Uow(AB1DF0804B0F5801) Tra(RFS4) Tas(0000060) Act Tdq Own
    Res(QILR                     ) Rle(004) Enq(00000000)
 Uow(AB1DF0804B0F5801) Tra(RFS4) Tas(0000060) Act Tdq Own
    Res(QILR                     ) Rle(004) Enq(00000000)
 Uow(AB1DF09A54115600) Tra(RTD1) Tas(0000061) Act Tdq Wai
    Res(QILR                     ) Rle(004) Enq(00000000)
```

In this instance, task 61 is the only waiter, and task 60 is the only owner, simplifying the task of identifying the enqueue owner. Task 60 owns one enqueue of type TSQUEUE, one of type DATASET, and two of type TDQ. These enqueues are owned on resources RMLTSQ, DCXISCG.IYLX1.RMLFILE and QILR respectively.

The CEMT INQUIRE TASK screen shows that task 60 has ended. You can use the CEMT INQUIRE UOW command to return information about the status of units of work that are associated with tasks which have ended, as well as with tasks that are still active.

```
INQUIRE UOW
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(AB1DD0FE5F219205) Inf Act Tra(CSSY) Tas(0000005)
    Age(00002569)                Use(CICSUSER)
 Uow(AB1DD0FE5FEF9C00) Inf Act Tra(CSSY) Tas(0000006)
    Age(00002569)                Use(CICSUSER)
 Uow(AB1DD0FE7FB82600) Inf Act Tra(CSTP) Tas(0000008)
    Age(00002569)                Use(CICSUSER)
 Uow(AB1DD98323E1C005) Inf Act Tra(CSNC) Tas(0000018)
    Age(00000282)                Use(CICSUSER)
 Uow(AB1DF0804B0F5801) Ind Shu Tra(RFS4) Tas(0000060)
    Age(00002699) Ter(S255) Netn(IGCS255 ) Use(CICSUSER) Con Lin(IYM52   )
 Uow(AB1DF09A54115600) Inf Act Tra(RTD1) Tas(0000061)
    Age(00002673) Ter(S254) Netn(IGCS254 ) Use(CICSUSER)
 Uow(AB1DF0B309126800) Inf Act Tra(CSNE) Tas(0000021)
    Age(00002647)                Use(CICSUSER)
 Uow(AB1DF16E3B78B403) Inf Act Tra(CEMT) Tas(0000064)
    Age(00002451) Ter(S255) Netn(IGCS255 ) Use(CICSUSER)
```

The CEMT INQUIRE UOW command can be filtered so that a UOW for a particular task is displayed. For example, CEMT INQUIRE UOW TASK(60) shows:

```
INQUIRE UOW TASK(60)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(AB1DF0804B0F5801) Ind Shu Tra(RFS4) Tas(0000060)
    Age(00002699) Ter(S255) Netn(IGCS255 ) Use(CICSUSER) Con Lin(IYM52   )
```

Note: The CEMT INQUIRE UOW command can also be filtered using a wildcard as a UOW filter. For example, CEMT INQUIRE UOW(*5801) would return information about UOW AB1DF0804B0F5801 only.

In order to see more information for a particular UOW, position the cursor alongside the UOW and press ENTER:

```
INQUIRE UOW
RESULT - OVERTYPE TO MODIFY
  Uow(AB1DF0804B0F5801)
  Uowstate( Indoubt )
  Waitstate(Shunted)
  Transid(RFS4)
  Taskid(0000060)
  Age(00002801)
  Termid(S255)
  Netname(IGCS255)
  Userid(CICSUSER)
  Waitcause(Connection)
  Link(IYM52)
  Sysid(ISC2)
  Netuowid(..GBIBMIYA.IGCS255 .0......)
```

The UOW in question is AB1DF0804B0F5801. The Uowstate is **Shunted**, which means that syncpoint processing has been deferred and locks are retained until resource integrity can be ensured. In this case, the UOW is shunted **Indoubt**, which means that task 60 failed during syncpoint processing while in the in-doubt window.

The reason for the UOW being shunted is given by Waitcause—in this case, it is **Connection**. The UOW has been shunted due to a failure of connection **ISC2**. The associated Link (or netname) for the connection is **IYM52**.

A CEMT INQUIRE UOWLINK command shows information about connections involved in distributed UOWs:

```
INQUIRE UOWLINK
STATUS:  RESULTS
 Uowl(02EC0011) Uow(AB1DF0804B0F5801) Con Lin(IYM52   )
    Coo Appc Una Sys(ISC2)               Net(..GBIBMIYA.IGCS255 .0......)
```

To see more information for the Link, position the cursor alongside the UOW and press ENTER:

```
INQUIRE UOWLINK
RESULT
  Uowlink(02EC0011)
  Uow(AB1DF0804B0F5801)
  Type(Connection)
  Link(IYM52)
  Action(             )
  Role(Coordinator)
  Protocol(Appc)
  Resyncstatus(Unavailable)
  Sysid(ISC2)
  Rmiqfy()
  Netuowid(..GBIBMIYA.IGCS255 .0......)
```

In this example, we can see that the connection ISC2 to system IYM52 is the syncpoint **Coordinator** for this UOW. The Resyncstatus is **Unavailable**, which means that the connection is not currently acquired.

A CEMT INQUIRE CONNECTION command confirms our findings:

```
I INQUIRE CONNECTION
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Con(ISC2) Net(IYM52   )     Ins Rel Vta Appc         Rec
 Con(ISC4) Net(IYM54   )     Ins Acq Vta Appc     Xok Unk
 Con(ISC5) Net(IYM55   )     Ins Acq Vta Appc     Xok Unk
```

To see more information for connection ISC2, position the cursor alongside the connection and press ENTER:

```
INQUIRE CONNECTION
RESULT
  Connection(ISC2)
  Netname(IYM52)
  Pendstatus( Notpending )
  Servstatus( Inservice )
  Connstatus( Released )
  Accessmethod(Vtam)
  Protocol(Appc)
  Purgetype(              )
  Xlnstatus()
  Recovstatus( Recovdata )
  Uowaction(              )
  Grname()
  Membername()
  Affinity(              )
  Remotesystem()
  Rname()
  Rnetname()
```

This shows that the connection ISC2 is **Released** with Recovstatus **Recovdata**, indicating that resynchronization is outstanding for this connection.

At this stage, if it is possible to acquire the connection to system IYM52, resynchronization will take place automatically, UOW AB1DF0804B0F5801 will be unshunted and its enqueues will be released, allowing task 61 to complete. However, if it is not possible to acquire the connection, you may decide to unshunt the UOW and override normal resynchronization. To decide whether to commit or backout the UOW, you need to inquire on the associated UOW on system IYM52. A CEMT INQUIRE UOW command on system IYM52 shows:

```
INQUIRE UOW
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(AB1DD01221BA6E01) Inf Act Tra(CSSY) Tas(0000005)
    Age(00003191)                          Use(CICSUSER)
 Uow(AB1DD0122276C201) Inf Act Tra(CSSY) Tas(0000006)
    Age(00003191)                          Use(CICSUSER)
 Uow(AB1DD01248A7B005) Inf Act Tra(CSTP) Tas(0000008)
    Age(00003191)                          Use(CICSUSER)
 Uow(AB1DD9057B8DD800) Inf Act Tra(CSNC) Tas(0000018)
    Age(00000789)                          Use(CICSUSER)
 Uow(AB1DF0805E76B400) Com Wai Tra(CSM3) Tas(0000079)
    Age(00003003) Ter(-AC3) Netn(IYM51   ) Use(CICSUSER) Wai
 Uow(AB1DF0B2FDD36400) Inf Act Tra(CSNE) Tas(0000019)
    Age(00003024)                          Use(CICSUSER)
 Uow(AB1DF15502238000) Inf Act Tra(CEMT) Tas(0000086)
    Age(00002853) Ter(S25C) Netn(IGCS25C ) Use(CICSUSER)
```

For transactions started at a terminal, the CEMT INQUIRE UOW command can be filtered using **Netuowid**, so that only UOWs associated with transactions executed from a particular terminal are displayed. In this case, task 60 on system IYM51 was executed at terminal S255. The Netuowid of UOW AB1DF0804B0F5801 on system IYM51 contains the luname of terminal S255.

Because Netuowids are identical for all UOWs which are connected within a single distributed unit of work, the Netuowid is a useful way of tying these UOWs together. In this example, the command CEMT INQUIRE UOW NETUOWID(*S255*) filters the CEMT INQUIRE UOW command as follows:

```
INQUIRE UOW NETUOWID(*S255*)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(AB1DF0805E76B400) Com Wai Tra(CSM3) Tas(0000079)
    Age(00003003) Ter(-AC3) Netn(IYM51   ) Use(CICSUSER) Wai
```

To see more information for UOW AB1DF0805E76B400, position the cursor alongside the UOW and press ENTER:

```
INQUIRE UOW
RESULT - OVERTYPE TO MODIFY
  Uow(AB1DF0805E76B400)
  Uowstate( Commit )
  Waitstate(Waiting)
  Transid(CSM3)
  Taskid(0000079)
  Age(00003003)
  Termid(-AC3)
  Netname(IYM51  )
  Userid(CICSUSER)
  Waitcause(Waitforget)
  Link(     )
  Sysid(    )
  Netuowid(..GBIBMIYA.IGCS255 .0......)
```

We can see that UOW AB1DF0805E76B400 is associated with a mirror task used in function shipping. The Uowstate **Commit** means that the UOW has been committed and the Waitstate **Waiting** means that it is waiting because the decision has not been communicated to IYM51. This allows us safely to commit the shunted UOW on system IYM51, in the knowledge that resource updates will be synchronous with those on IYM52 for this distributed unit of work. You can use the CEMT SET UOW command to commit the shunted UOW. Once the shunted UOW is committed, its enqueues are released and task 61 is allowed to continue.

Another possible scenario could be that IYM52 is not available. If it is not practical to wait for IYM52 to become available and you are prepared to accept the risk to data integrity, you can use the CEMT SET CONNECTION command to commit, backout, or force all UOWs that have failed in-doubt due to the failure of connection ISC2.

In this example, transaction RTD1 was suspended on an ENQUEUE for a transient data queue. An active lock for the queue was owned by UOW AB1DF0804B0F5801, which had failed in-doubt. To avoid tasks being suspended in this way, you could define the transient data queue with the WAITACTION option set to **REJECT** (the default WAITACTION). If you do this, an in-doubt failure of a task updating the queue results in a retained lock being held by the shunted UOW. Requests for the retained lock are then rejected with the LOCKED condition.

For detailed information about CEMT commands, see CEMT master terminal, in the *CICS Supplied Transactions* manual.

Resolving a resynchronization failure

This section is an example of how to resolve a resynchronization failure. It uses the following commands:

- CEMT INQUIRE CONNECTION
- CEMT INQUIRE UOWLINK
- CEMT INQUIRE UOW
- CEMT INQUIRE UOWENQ
- SET CONNECTION NOTPENDING.

A user has reported that their transaction on system IYLX1 (which involves function shipping requests to system IYLX4) is failing with a 'SYSIDERR'. A CEMT INQUIRE CONNECTION command on system IYLX1 shows the following:

```
INQUIRE CONNECTION
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Con(ISC2) Net(IYLX2   )     Ins Rel Vta Appc         Unk
 Con(ISC4) Net(IYLX4   ) Pen Ins Acq Vta Appc     Xno Unk
 Con(ISC5) Net(IYLX5   )     Ins Acq Vta Appc     Xok Unk
```

Figure 86. CEMT INQUIRE CONNECTION—connections owned by system IYLX1

The connection to system IYLX4 is an APPC connection called ISC4. To see more information about this connection, put the cursor on the ISC4 line and press ENTER—see Figure 87 on page 301.

```
INQUIRE CONNECTION
RESULT - OVERTYPE TO MODIFY
  Connection(ISC4)
  Netname(IYLX4)
  Pendstatus( Pending )
  Servstatus( Inservice )
  Connstatus( Acquired )
  Accessmethod(Vtam)
  Protocol(Appc)
  Purgetype(              )
  Xlnstatus(Xnotdone)
  Recovstatus( Nrs )
  Uowaction(              )
  Grname()
  Membername()
  Affinity(              )
  Remotesystem()
  Rname()
  Rnetname()
```

Figure 87. CEMT INQUIRE CONNECTION—details of connection ISC4

Although the Connstatus of connection ISC4 is **Acquired**, the Xlnstatus is **Xnotdone**. The exchange lognames (XLN) flow for this connection has not completed successfully. (When CICS systems connect they exchange lognames. These lognames are verified before resynchronization is attempted, and an exchange lognames failure means that resynchronization is not possible.) For function shipping, a failure for the connection causes a SYSIDERR. Synchronization level 2 conversations are not allowed on this connection until lognames are successfully exchanged. (This restriction does not apply to MRO connections.)

The reason for the exchange lognames failure is reported in the CSMT log. A failure on a CICS Transaction Server for z/OS system can be caused by:

- An initial start (START=INITIAL) of the CICS TS for z/OS system, or of a partner.

  Note: A cold start (START=COLD) of a CICS TS for z/OS system preserves resynchronization information (including the logname) and does not, therefore, cause an exchange lognames failure.
- Use of the CEMT SET CONNECTION NORECOVDATA command.
- A system logic or operational error.

The Pendstatus for connection ISC4 is **Pending**, which means that there is resynchronization work outstanding for the connection; this work cannot be completed because of the exchange lognames failure.

At this stage, if we were not concerned about loss of synchronization, we could force all in-doubt UOWs to commit or back out by issuing the SET CONNECTION NOTPENDING command. However, there are commands that allow us to investigate the outstanding resynchronization work that exists before we clear the pending condition.

You can use a CEMT INQUIRE UOWLINK command to display information about UOWs that require resynchronization with system IYLX4:

```
INQUIRE UOWLINK LINK(IYLX4)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uowl(016C0005) Uow(ABD40B40C1334401) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX150 M. A....)
 Uowl(01680005) Uow(ABD40B40C67C8201) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX151 M. F@b..)
 Uowl(016D0005) Uow(ABD40B40DA5A8803) Con Lin(IYLX4   )
    Coo Appc Col Sys(ISC4)            Net(..GBIBMIYA.IYLX156 M. .!h..)
```

Figure 88. CEMT INQUIRE UOWLINK—UOWs that require resynchronization with system IYLX4

To see more information for each UOW-link, press enter alongside it. For example, the expanded information for UOW-link 016C0005 shows the following:

```
I UOWLINK LINK(IYLX4)
RESULT - OVERTYPE TO MODIFY
  Uowlink(016C0005)
  Uow(ABD40B40C1334401)
  Type(Connection)
  Link(IYLX4)
  Action(              )
  Role(Coordinator)
  Protocol(Appc)
  Resyncstatus(Coldstart)
  Sysid(ISC4)
  Rmiqfy()
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
```

Figure 89. CEMT INQUIRE UOWLINK—detailed information for UOW-link 016C0005

The Resyncstatus of **Coldstart** confirms that system IYLX4 has been started with a new logname. The Role for this UOW-link is shown as **Coordinator**, which means that IYLX4 is the syncpoint coordinator.

You could now use a CEMT INQUIRE UOW LINK(IYLX4) command to show all UOWs that are in-doubt and which have system IYLX4 as the coordinator system:

```
INQUIRE UOW LINK(IYLX4)
STATUS:  RESULTS - OVERTYPE TO MODIFY
 Uow(ABD40B40C1334401) Ind Shu Tra(RFS1) Tas(0000674)
    Age(00003560) Ter(X150) Netn(IYLX150 ) Use(CICSUSER) Con Lin(IYLX4   )
 Uow(ABD40B40C67C8201) Ind Shu Tra(RFS1) Tas(0000675)
    Age(00003465) Ter(X151) Netn(IYLX151 ) Use(CICSUSER) Con Lin(IYLX4   )
 Uow(ABD40B40DA5A8803) Ind Shu Tra(RFS1) Tas(0000676)
    Age(00003462) Ter(X156) Netn(IYLX156 ) Use(CICSUSER) Con Lin(IYLX4   )
```

Figure 90. CEMT INQUIRE UOW LINK(IYLX4)—all UOWs that have IYLX4 as the coordinator

To see more information for each in-doubt UOW, press enter on its line. For example, the expanded information for UOW ABD40B40C1334401 shows the following:

```
INQUIRE UOW LINK(IYLX4)
RESULT - OVERTYPE TO MODIFY
  Uow(ABD40B40C1334401)
  Uowstate( Indoubt )
  Waitstate(Shunted)
  Transid(RFS1)
  Taskid(0000674)
  Age(00003906)
  Termid(X150)
  Netname(IYLX150)
  Userid(CICSUSER)
  Waitcause(Connection)
  Link(IYLX4)
  Sysid(ISC4)
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
```

Figure 91. CEMT INQUIRE UOW LINK(IYLX4)—detailed information for UOW ABD40B40C1334401

This UOW cannot be resynchronized by system IYLX4—its status is shown as **Indoubt**, because IYLX4 does not know whether the associated UOW that ran on IYLX4 committed or backed out.

You can use the CEMT INQUIRE UOWENQ command to display the resources that have been locked by all shunted UOWs (those that own retained locks):

```
INQUIRE UOWENQ OWN RETAINED
STATUS:  RESULTS
 Uow(ABD40B40C1334401) Tra(RFS1) Tas(0000674) Ret Tsq Own
    Res(RFS1X150                 ) Rle(008) Enq(00000008)
 Uow(ABD40B40C67C8201) Tra(RFS1) Tas(0000675) Ret Tsq Own
    Res(RFS1X151                 ) Rle(008) Enq(00000008)
 Uow(ABD40B40DA5A8803) Tra(RFS1) Tas(0000676) Ret Tsq Own
    Res(RFS1X156                 ) Rle(008) Enq(00000008)
```

Figure 92. CEMT INQUIRE UOWENQ—resources locked by all shunted UOWs

You can filter the INQUIRE UOWENQ command so that only enqueues that are owned by a particular UOW are displayed. For example, to filter for enqueues owned by UOW ABD40B40C1334401:

```
INQUIRE UOWENQ OWN UOW(*4401)
STATUS:  RESULTS
 Uow(ABD40B40C1334401) Tra(RFS1) Tas(0000674) Ret Tsq Own
    Res(RFS1X150                 ) Rle(008) Enq(00000008)
```

Figure 93. CEMT INQUIRE UOWENQ—resources locked by UOW ABD40B40C1334401

To see more information for this UOWENQ, press enter alongside it:

```
INQUIRE UOWENQ OWN UOW(*4401)
RESULT
  Uowenq
  Uow(ABD40B40C1334401)
  Transid(RFS1)
  Taskid(0000674)
  State(Retained)
  Type(Tsq)
  Relation(Owner)
  Resource(RFS1X150)
  Rlen(008)
  Enqfails(00000008)
  Netuowid(..GBIBMIYA.IYLX150 M. A....)
  Qualifier()
  Qlen(000)
```

Figure 94. CEMT INQUIRE UOWENQ—detailed information for UOWENQ ABD40B40C1334401

With knowledge of the application, it may now be possible to decide whether updates to the locked resources should be committed or backed out. In the case of UOW ABD40B40C1334401, the locked resource is the temporary storage queue RFS1X150. This resource has an ENQFAILS value of 8, which is the number of tasks that have received the **LOCKED** response due to this enqueue being held in retained state.

You can use the SET UOW command to commit, back out, or force the uncommitted updates made by the shunted UOWs. Next, you must use the SET CONNECTION(ISC4) NOTPENDING command to clear the pending condition and allow synchronization level 2 conversations (including the function shipping requests which were previously failing with SYSIDERR).

You can use the XLNACTION option of the CONNECTION definition to control the effect of an exchange lognames failure. In this example, the XLNACTION for the connection ISC4 is **KEEP**. This meant that:

- The shunted UOWs on system IYLX1 were kept following the cold/warm log mismatch with IYLX4.
- The APPC connection between IYLX1 and IYLX4 could not be used for function shipping requests until the pending condition was resolved.

An XLNACTION of **FORCE** for connection ISC4 would have caused the SET CONNECTION NOTPENDING command to have been issued automatically when the cold/warm log mismatch occurred. This would have forced the shunted UOWs to commit or back out, according to the ACTION option of the associated transaction definition. The connection ISC4 would then not have been placed into **Pending** status. However, setting XLNACTION to FORCE allows no investigation of shunted UOWs following an exchange lognames failure, and therefore represents a greater risk to data integrity than setting XLNACTION to KEEP.

Chapter 27. Intercommunication and XRF

For further information about the extended recovery facility (XRF) of CICS Transaction Server for z/OS, see the *CICS/ESA 3.3 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 13, “Defining links to remote systems,” on page 143.

MRO and ISC sessions are not XRF-capable because they cannot have backup sessions to the alternate CICS system.

You can use the AUTOCONNECT option in your link 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 Specifying CICS system initialization parameters, in the *CICS System Definition Guide*.

The rest of this chapter contains the following topics:

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 CONNECTION 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 SESSIONS definition specifies YES.
2. The sessions are being tracked, and are bound when the takeover occurs. The status of LUTYPE6.1 sessions is tracked unless RECOVOPTION(NONE) is specified in the SESSIONS definition.

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 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 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 28. 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 13, "Defining links to remote systems," on page 143 and CONNECTION definition attributes, in the *CICS Resource Definition Guide*. For details of the PSDINT system initialization parameter used to specify persistent session support, see PSDINT, in the *CICS System Definition Guide*.

The rest of this chapter contains the following topics:

- "Comparison of persistent session support and XRF"
- "Interconnected CICS environment, recovery and restart"
- "Effect on application programs" on page 309

Comparison of persistent session support and XRF

XRF allows an alternate, partially initialized CICS system to take over control from an active CICS system which has 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, MVS, or CEC 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 “Traditional routing of transactions started by ATI” on page 59.)
- *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

Appendix A. Intercommunication 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. The rules apply to:

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 7770 and 2260 terminals.
  - Pipeline logical units with pooling.
  - MVS system consoles. (Messages entered through a console can be directed to any CICS system via the MODIFY 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.
- For a transaction invoked by a terminal-related EXEC CICS START command to be eligible for *enhanced* routing, *all* of the following conditions must be met:
  - The START command is a member of the subset of eligible START commands—that is, it meets all the following conditions:
    - The START command specifies the TERMID option, which names the principal facility of the task that issues the command. That is, the transaction to be started must be terminal-related, and associated with the principal facility of the starting task.
    - The principal facility of the task that issues the START command is *not* a surrogate Client virtual terminal.
    - The SYSID option of the START command does not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)
  - The requesting region, the TOR, and the target region are all CICS Transaction Server for OS/390, Version 1 Release 3 or later.
  - The requesting region and the TOR (if they are different) are connected by either of the following:
    - An MRO link
    - An APPC parallel-session link.
  - The TOR and the target region are connected by either of the following:
    - An MRO link.
    - An APPC single- or parallel-session link. If an APPC link is used, at least one of the following must be true:
      1. Terminal-initiated transaction routing has previously taken place over the link.
      2. CICSPlex SM is being used for routing.
  - The transaction definition in the requesting region specifies ROUTABLE(YES).
  - If the transaction is to be routed dynamically, the transaction definition in the TOR specifies DYNAMIC(YES).
- For a non-terminal-related START request to be eligible for *enhanced* routing, *all* of the following conditions must be met:
  - The requesting region is CICS Transaction Server for OS/390, Version 1 Release 3 or later.

    Note: In order for the distributed routing program to be invoked on the *target* region, as well as on the requesting region, the target region too must be CICS Transaction Server for OS/390, Version 1 Release 3 or later.
  - The requesting region and the target region are connected by either of the following:
    - An MRO link.
    - An APPC single- or parallel-session link. If an APPC link is used, and the distributed routing program is to be invoked on the target region, at least one of the following must be true:
      1. Terminal-initiated transaction routing has previously taken place over the link.
      2. CICSPlex SM is being used for routing.

  - The transaction definition in the requesting region specifies ROUTABLE(YES).
  - If the request is to be routed dynamically:
    - The transaction definition in the requesting region specifies DYNAMIC(YES).
    - The SYSID option of the START command does not specify the name of a remote region. (That is, the remote region on which the transaction is to be started must not be specified explicitly.)
- The following types of dynamic transaction routing requests cannot be daisy-chained. The routing of:
  - Non-terminal-related START requests
  - CICS business transaction services processes and activities.

Dynamic routing of DPL requests

- For a distributed program link request to be eligible for dynamic routing, the remote program must either:
  - Be defined to the local system as DYNAMIC, *or*
  - Not be defined to the local system.
- Daisy-chaining of dynamically-routed DPL requests is not supported—see "Daisy-chaining of DPL requests" on page 91.

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.
- There are restrictions on the dynamic routing of transactions initiated by EXEC CICS START commands—see the list of conditions in "Transaction routing" on page 313.

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.

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 LPA-resident; otherwise jobs and console commands may abend on completion.
- Interregion communication requires subsystem interface (SSI) support.
- 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 320.

  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 APPC architecture" on page 328.

  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*.

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 21. CICS support of APPC options sets

Set #	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	Receive immediate **Note:** CICS programs support receive_immediate requests provided these requests are coded using the common programming Interface for communications.	Yes
108	Sync point 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
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

Table 21. CICS support of APPC options sets (continued)

Set #	Set name	Supported
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
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 CEMT master terminal, 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 INQUIRE CONNECTION, 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 155. 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 327 for details of the corresponding return-code mapping.

Note: Wherever CEMT is shown, the equivalent form of EXEC CICS command can be used.

Table 22. CHANGE_SESSION_LIMIT

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(vnle)	Not supported.
RESPONSIBLE(source)	Yes.
RESPONSIBLE(target)	Not supported. CICS does not support receipt of RESP(TARGET).
RETURN_CODE	Supported.

Table 23. INITIALIZE_SESSION_LIMIT

INITIALIZE_SESSION_LIMIT	DEFINE SESSIONS (CICS resource definition)
LU_NAME(vble)	CONNECTION( )
MODE_NAME(vble)	MODENAME( )
LU_MODE_SESSION_LIMIT(vble)	MAXIMUM(value1,)

Table 23. INITIALIZE_SESSION_LIMIT (continued)

INITIALIZE_SESSION_LIMIT	**DEFINE SESSIONS (CICS resource definition)**
MIN_CONWINNERS_SOURCE(vble)	MAXIMUM(,value2)
MIN_CONWINNERS_TARGET(vnle)	Not supported.
RETURN_CODE	Supported.

Table 24. PROCESS_SESSION_LIMIT

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.

Table 25. RESET_SESSION_LIMIT

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.

Table 26. ACTIVATE_SESSION

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.

Table 27. DEACTIVATE_CONVERSATION_GROUP

DEACTIVATE_CONVERSATION_GROUP	**Not supported.**

Table 28. DEACTIVATE_SESSION

DEACTIVATE_SESSION	**Not supported.**

Table 29. DEFINE_LOCAL_LU

DEFINE_LOCAL_LU	DEFINE SESSIONS + DFHSIT macro (CICS resource definitions)
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 (ESM).
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 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.

Table 30. DEFINE_MODE

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.

Table 30. DEFINE_MODE (continued)

DEFINE_MODE	EXEC CICS CONNECT PROCESS + MODEENT macro (ACF/VTAM systems definition) + DEFINE SESSIONS (CICS resource definition)
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.

Table 31. DEFINE_REMOTE_LU

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 RACF APPCLU profile.
SECURITY_ACCEPTANCE(NONE)	ATTACHSEC(LOCAL)
SECURITY_ACCEPTANCE (CONVERSATION)	ATTACHSEC(VERIFY)
SECURITY_ACCEPTANCE (ALREADY_VERIFIED)	ATTACHSEC(IDENTIFY) or ATTACHSEC(PERSISTENT).

Table 32. DEFINE_TP

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.

Table 32. DEFINE_TP (continued)

DEFINE_TP	DEFINE TRANSACTION (CICS resource definition)
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.

Table 33. DELETE

DELETE	EXEC CICS DISCARD
LOCAL_LU_NAME(vble)	Not supported.
REMOTE_LU_NAME	Not supported.
MODE_NAME	Not supported.
TP_NAME	DISCARD TRANSACTION( )
RETURN_CODE	Supported.

Table 34. DISPLAY_LOCAL_LU

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)	MAXIMUM on INQ MODENAME.
LU_SESSION_COUNT(vble)	ACTIVE on INQ MODENAME
SECURITY(vble)	Not available.
MAP_NAMES(vble)	Not supported.
REMOTE_LU_NAMES(vble)	INQ CONNECTION(*)
TP_NAMES(vble)	INQ TRANSACTION(*)
BIND_RSP_QUEUE_CAPABILITY(vble)	Not supported.
RETURN_CODE	Supported.

Table 35. DISPLAY_REMOTE_LU

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)	CONNECTION name.
UNINTERPRETED_LU_NAME(vble)	NETNAME on INQ CONNECTION.
INITIATE_TYPE(vble)	Not supported.
PARALLEL_SESSION_SUPPORT(vble)	SINGLESESS(YIN) on CEDA VIEW.
CNOS_SUPPORT(vble)	Always YES.
SECURITY_ACCEPTANCE_LOCAL_LU (vble)	Not available.
SECURITY_ACCEPTANCE_REMOTE_LU (vble)	Not available.
MODE_NAMES(vble)	CEDA VIEW SESSIONS with locally-known LU name.
RETURN_CODE	Supported.

Table 36. DISPLAY_MODE

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.

Table 37. DISPLAY_TP

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 Table 38. 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 Table 38. Refer to the *CICS Messages and Codes* manual for a full description of the message. In certain cases, DFHZC4901 is also issued to give further information.

Table 38. Messages triggered by CLS1

APPC RETURN CODE	CICS MESSAGE
OK	DFHZC4900 result = SUCCESSFUL
ACTIVATION_FAILURE_RETRY	DFHZC4900 result = VALUES AMENDED + DFHZC4901 MAX = 0

Table 38. Messages triggered by CLS1 (continued)

APPC RETURN CODE	CICS MESSAGE
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_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

CICS deviations from 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.

Bibliography

The CICS Transaction Server for z/OS library

The published information for CICS Transaction Server for z/OS is delivered in the following forms:

The CICS Transaction Server for z/OS Information Center
The CICS Transaction Server for z/OS Information Center is the primary source of user information for CICS Transaction Server. The Information Center contains:

- Information for CICS Transaction Server in HTML format.
- Licensed and unlicensed CICS Transaction Server books provided as Adobe Portable Document Format (PDF) files. You can use these files to print hardcopy of the books. For more information, see "PDF-only books."
- Information for related products in HTML format and PDF files.

One copy of the CICS Information Center, on a CD-ROM, is provided automatically with the product. Further copies can be ordered, at no additional charge, by specifying the Information Center feature number, 7014.

Licensed documentation is available only to licensees of the product. A version of the Information Center that contains only unlicensed information is available through the publications ordering system, order number SK3T-6945.

Entitlement hardcopy books
The following essential publications, in hardcopy form, are provided automatically with the product. For more information, see "The entitlement set."

The entitlement set

The entitlement set comprises the following hardcopy books, which are provided automatically when you order CICS Transaction Server for z/OS, Version 3 Release 2:

Memo to Licensees, GI10-2559
CICS Transaction Server for z/OS Program Directory, GI13-0515
CICS Transaction Server for z/OS Release Guide, GC34-6811
CICS Transaction Server for z/OS Installation Guide, GC34-6812
CICS Transaction Server for z/OS Licensed Program Specification, GC34-6608

You can order further copies of the following books in the entitlement set, using the order number quoted above:

CICS Transaction Server for z/OS Release Guide
CICS Transaction Server for z/OS Installation Guide
CICS Transaction Server for z/OS Licensed Program Specification

PDF-only books

The following books are available in the CICS Information Center as Adobe Portable Document Format (PDF) files:

CICS books for CICS Transaction Server for z/OS

General

CICS Transaction Server for z/OS Program Directory, GI13-0515
CICS Transaction Server for z/OS Release Guide, GC34-6811
CICS Transaction Server for z/OS Migration from CICS TS Version 3.1, GC34-6858

CICS Transaction Server for z/OS Migration from CICS TS Version 1.3, GC34-6855
CICS Transaction Server for z/OS Migration from CICS TS Version 2.2, GC34-6856
CICS Transaction Server for z/OS Installation Guide, GC34-6812

Administration

CICS System Definition Guide, SC34-6813
CICS Customization Guide, SC34-6814
CICS Resource Definition Guide, SC34-6815
CICS Operations and Utilities Guide, SC34-6816
CICS Supplied Transactions, SC34-6817

Programming

CICS Application Programming Guide, SC34-6818
CICS Application Programming Reference, SC34-6819
CICS System Programming Reference, SC34-6820
CICS Front End Programming Interface User's Guide, SC34-6821
CICS C++ OO Class Libraries, SC34-6822
CICS Distributed Transaction Programming Guide, SC34-6823
CICS Business Transaction Services, SC34-6824
Java Applications in CICS, SC34-6825
JCICS Class Reference, SC34-6001

Diagnosis

CICS Problem Determination Guide, SC34-6826
CICS Messages and Codes, GC34-6827
CICS Diagnosis Reference, GC34-6862
CICS Data Areas, GC34-6863-00
CICS Trace Entries, SC34-6828
CICS Supplementary Data Areas, GC34-6864-00

Communication

CICS Intercommunication Guide, SC34-6829
CICS External Interfaces Guide, SC34-6830
CICS Internet Guide, SC34-6831

Special topics

CICS Recovery and Restart Guide, SC34-6832
CICS Performance Guide, SC34-6833
CICS IMS Database Control Guide, SC34-6834
CICS RACF Security Guide, SC34-6835
CICS Shared Data Tables Guide, SC34-6836
CICS DB2 Guide, SC34-6837
CICS Debugging Tools Interfaces Reference, GC34-6865

CICSPlex SM books for CICS Transaction Server for z/OS

General

CICSPlex SM Concepts and Planning, SC34-6839
CICSPlex SM User Interface Guide, SC34-6840
CICSPlex SM Web User Interface Guide, SC34-6841

Administration and Management

CICSPlex SM Administration, SC34-6842
CICSPlex SM Operations Views Reference, SC34-6843
CICSPlex SM Monitor Views Reference, SC34-6844
CICSPlex SM Managing Workloads, SC34-6845
CICSPlex SM Managing Resource Usage, SC34-6846
CICSPlex SM Managing Business Applications, SC34-6847

Programming

CICSPlex SM Application Programming Guide, SC34-6848
CICSPlex SM Application Programming Reference, SC34-6849

Diagnosis

CICSPlex SM Resource Tables Reference, SC34-6850
CICSPlex SM Messages and Codes, GC34-6851
CICSPlex SM Problem Determination, GC34-6852

CICS family books

Communication

CICS Family: Interproduct Communication, SC34-6853
CICS Family: Communicating from CICS on zSeries, SC34-6854

Licensed publications

The following licensed publications are not included in the unlicensed version of the Information Center:

CICS Diagnosis Reference, GC34-6862
CICS Data Areas, GC34-6863-00
CICS Supplementary Data Areas, GC34-6864-00
CICS Debugging Tools Interfaces Reference, GC34-6865

Other CICS books

The following publications contain further information about CICS, but are not provided as part of CICS Transaction Server for z/OS, Version 3 Release 2.

Designing and Programming CICS Applications	SR23-9692
CICS Application Migration Aid Guide	SC33-0768
CICS Family: API Structure	SC33-1007
CICS Family: Client/Server Programming	SC33-1435
CICS Transaction Gateway for z/OS Administration	SC34-5528
CICS Family: General Information	GC33-0155
CICS 4.1 Sample Applications Guide	SC33-1173
CICS/ESA 3.3 XRF Guide	SC33-0661

Books from related libraries

IMS

- *CICS/VS to IMS/VS Intersystem Communication Primer*, SH19-6247 through SH19-6254
- *IMS/ESA Data Communication Administration Guide*, SC26-3060
- *IMS/ESA Installation Volume 1: Installation and Verification*, GC26-8736
- *IMS/ESA Installation Volume 2: System Definition and Tailoring*, GC26-8737
- *IMS/ESA Operations Guide*, SC26-8741
- *IMS Programming Guide for Remote SNA Systems*, SC26-4186

MVS/ESA

- *OS/390 MVS Setting Up a Sysplex*, GC28-1779
- *OS/390 Parallel Sysplex Application Migration*, GC28-1863

Network program products

- *Network Program Products General Information*, GC30-3350

Systems Application Architecture (SAA)

- *SAA Common Programming Interface Communications Reference*, SC26-4399

Systems Network Architecture (SNA)

- *Concepts and Products*, GC30-3072
- *Format and Protocol Reference Manual: Architecture Logic*, SC30-3112
- *Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2*, SC30-3269
- *Format and Protocol Reference Manual: Distribution Services*, SC30-3098
- *Reference: Peer Protocols*, SC31-6808-1
- *Sessions Between Logical Units*, GC20-1868
- *SNA Formats*, GA27-3136
- *Technical Overview*, GC30-3073
- *Transaction Programmer's Reference Manual for LU Type 6.2*, GC30-3084

VTAM

- *OS/390 eNetwork Communications Server: SNA Customization*, LY43-0110
- *OS/390 eNetwork Communications Server: SNA Data Areas Volume 1*, LY43-0111
- *OS/390 eNetwork Communications Server: SNA Data Areas Volume 2*, LY43-0112
- *OS/390 eNetwork Communications Server: SNA Diagnosis*, LY43-0079
- *OS/390 eNetwork Communications Server: SNA Planning and Migration Guide*, SC31-8622
- *OS/390 eNetwork Communications Server: SNA Messages*, SC31-8569
- *OS/390 eNetwork Communications Server: SNA Network Implementation*, SC31-8563
- *OS/390 eNetwork Communications Server: SNA Operation*, SC31-8567
- *OS/390 eNetwork Communications Server: SNA Programming*, SC31-8573
- *VTAM Release Guide*, GC31-6545
- *OS/390 eNetwork Communications Server: SNA Resource Definition Reference*, SC31-8565

Determining if a publication is current

IBM regularly updates its publications with new and changed information. When first published, both hardcopy and BookManager® softcopy versions of a publication are usually in step. However, due to the time required to print and distribute hardcopy books, the BookManager version is more likely to have had last-minute changes made to it before publication.

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

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

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

Accessibility

Accessibility features help a user who has a physical disability, such as restricted mobility or limited vision, to use software products successfully.

You can perform most tasks required to set up, run, and maintain your CICS system in one of these ways:

- using a 3270 emulator logged on to CICS
- using a 3270 emulator logged on to TSO
- using a 3270 emulator as an MVS system console

IBM Personal Communications provides 3270 emulation with accessibility features for people with disabilities. You can use this product to provide the accessibility features you need in your CICS system.

Index

A

B

C

D

E

F

G

I

L

M

N

P

Q

R

S

T

U

V

W

X

Notices

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

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

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

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

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

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

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

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

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

Trademarks

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Adobe and the Adobe logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

Other product and service names might be trademarks of IBM or other companies.

Readers' Comments — We'd Like to Hear from You

CICS Transaction Server for z/OS
Intercommunication Guide
Version 3 Release 2

Publication No. SC34-6829-04

We appreciate your comments about this publication. Please comment on specific errors or omissions, accuracy, organization, subject matter, or completeness of this book. The comments you send should pertain to only the information in this manual or product and the way in which the information is presented.

For technical questions and information about products and prices, please contact your IBM branch office, your IBM business partner, or your 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. IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you state on this form.

Comments:

Thank you for your support.

Submit your comments using one of these channels:

- Send your comments to the address on the reverse side of this form.
- Send a fax to the following number: +44–1962–816151
- Send your comments via email to: idrcf@hursley.ibm.com

If you would like a response from IBM, please fill in the following information:

Name	Address
Company or Organization	
Phone No.	Email address

Readers' Comments — We'd Like to Hear from You
SC34-6829-04

Cut or Fold
Along Line

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

PLACE
POSTAGE
STAMP
HERE

IBM United Kingdom Limited
User Technologies Department (MP095)
Hursley Park
Winchester
Hampshire
SO21 2JN
United Kingdom

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

SC34-6829-04

Cut or Fold
Along Line

Product Number: 5655-M15

SC34-6829-04

Spine information:

CICS Transaction Server for z/OS **Intercommunication Guide** Version 3 Release 2