CICS Family

Interproduct Communication

SC34-6473-02

CICS Family

Interproduct Communication

SC34-6473-02

Note!
Before using this information and the products it supports, be sure to read the general information under “Notices” on page 73.

Twelfth edition (July 2010)

This edition applies to the following IBM licensed programs, and to all subsequent versions, releases, and modifications of these programs until otherwise indicated in new editions. Consult the latest edition of the applicable IBM system bibliography for current information on these products.

- CICS Transaction Server for z/OS Version 3
- CICS Transaction Server for z/OS Version 2, program number 5697-E93
- CICS Transaction Server for OS/390, program number 5655-147
- CICS Transaction Server for VSE/ESA, program number 5648-054
- CICS/VSE Version 2, program number 5686-026
- CICS Transaction Server for Windows, Version 5.0, program number 5724-D05
- CICS Transaction Server for iSeries, program number 5722-DFH
- CICS/400 Version 4, program number 5769-DFH
- TXSeries Version 5.0 for Multiplatforms, part number 5724-B44
- TXSeries for HP-UX, Version 4.2, program number 5801-AAR

This book is based on the ninth edition of the *CICS Family: Interproduct Communication* manual, SC34-6267-00. Changes from that edition are marked by vertical lines to the left of the changes.

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

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

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

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

Contents

Preface

What this book is about

This book introduces the subject of intercommunication between CICS® family members. It shows what functions are available, and how the systems are configured.

The CICS products covered by this book are:

- CICS on System/390®
- CICS Transaction Server for Windows®
- CICS on Open Systems
- CICS/400
- CICS Clients

Important: These are generic terms for subsets of CICS products. Their meanings are defined in "Terminology."

Who this book is for

This book is for anyone who is involved in the planning and implementation of communication between different CICS products.

What you need to know to understand this book

You should have a general knowledge of the facilities of CICS, and of the communication functions of the operating environments of your CICS systems. To implement CICS interproduct communication, you will also need the detailed product-specific information that is in the Intercommunication Guide for your local CICS product. The documentation plan for the individual CICS products is explained in Chapter 1, "CICS interproduct communication," on page 3.

Terminology

Throughout this book, when the term "*CICS*" is used without specifying any particular product or version level, it can be taken as a generic term for all the CICS family products.

The following CICS products run on computers of the System/370, System/390, or zSeries® family, and support communication with CICS products that run on other hardware platforms. (Not all of these products run on all of these computers—for example, CICS Transaction Server for z/OS® Version 2 does not run on System/370.)

- CICS Transaction Server for z/OS Version 3
- CICS Transaction Server for z/OS Version 2, program number 5697–E93
- CICS Transaction Server for OS/390® Version 1, program number 5655-147
- CICS Transaction Server for VSE/ESA, program number 5648-054
- CICS/VSE Version 2, program number 5686-026

In this book, the term **System/390** is used to refer to any System/370, System/390, or zSeries computer on which one of the above products can run. The term **non-System/390** refers to the hardware platforms used by other CICS

products—for example, iSeries® (used by CICS/400), IBM-compatible personal computers (used by CICS Transaction Server for Windows), and RISC System/6000 (used by CICS on Open Systems).

In statements that apply to each of the CICS products that runs on a System/390 hardware platform, the generic term **CICS on System/390** is used to represent all of them. One of these CICS products is referred to by name only if there is a difference in its interface to non-System/390 products as compared with the interface from other System/390 products. Subject to explicitly-stated exceptions, interpret all references to CICS as applying to your CICS on System/390 product.

The term *CICS Transaction Server for z/OS*, without a qualifying Version number, is used as a generic term for:
- 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

The term *CICS Transaction Server for OS/390*, without a qualifying Version number, refers to CICS Transaction Server for OS/390 Release 3.

The term *CICS Transaction Server for Windows*, without a qualifier, means CICS Transaction Server for Windows, Version 5.0.

The term *CICS on Open Systems* is used as a generic name for:
- TXSeries Version 5.0 for Multiplatforms, which contains:
  - CICS for AIX®
  - CICS for HP-UX
  - CICS for Sun Solaris
  - CICS for Windows NT®
- TXSeries Version 4.3 for AIX (which contains CICS for AIX)
- TXSeries Version 4.3 for Sun Solaris (which contains CICS for Sun Solaris)
- TXSeries Version 4.3 for Windows NT (which contains CICS for Windows NT)
- TXSeries Version 4.2 for HP-UX (which contains CICS for HP-UX)

Where it is necessary to distinguish between these products, the full product names are quoted.

The term *CICS Transaction Server for VSE/ESA* means CICS Transaction Server for VSE/ESA Release 1.1.1.

The term *CICS/VSE* means CICS for VSE/ESA Version 2 Release 3.

The term *CICS/400* is used as a generic name for:
- CICS/400 Version 4 Release 5
- CICS Transaction Server for iSeries

The term *CICS Clients* is used as a generic term for:
- The CICS Universal Client (for Windows NT, Windows 2000, and Windows XP)
- The CICS Client elements of the CICS Transaction Gateway products
- The client daemons of the CICS Transaction Gateway products

The notation **CICS–CICS Transaction Server for Windows** is used to refer to communication in either direction. To specify communication in only one direction, an arrow is added. For example, CICS–CICS on Open Systems function shipping

refers to function shipping from CICS to CICS on Open Systems or from CICS on Open Systems to CICS. CICS/400→CICS function shipping refers only to function shipping from CICS/400 to CICS.

Summary of changes

This book is based on the ninth edition of the *CICS Interproduct Communication* manual, SC34-6267-00. Changes from that edition are marked by vertical bars in the left margin.

This part lists briefly the changes that have been made for the following editions:
- "Changes for the tenth edition"
- "Changes for the ninth edition"
- "Changes for the eighth edition"
- "Changes for the seventh edition"

Changes for the tenth edition

The more significant changes for this edition are:

- The book has been revised to take account of the following new products:
  - CICS Transaction Server for z/OS Version 3 Release 1

Changes for the ninth edition

The more significant changes for this edition were:

- The book was revised to take account of the following new products:
  - CICS Transaction Server for z/OS Version 2 Release 3
  - CICS Transaction Server for Windows, Version 5.0
- References to the following CICS products, which are no longer supported, were removed:
  - CICS Transaction Server for z/OS Version 2 Release 1
  - CICS Transaction Server for OS/390 Release 2
  - CICS Transaction Server for OS/390 Release 1
  - CICS Transaction Server for VSE/ESA Release 1.0
  - CICS/ESA Version 4.1
  - CICS Transaction Server for OS/2 Warp Version 4
  - CICS for OS/2 Version 3.1

Changes for the eighth edition

The more significant changes for this edition were:

- The book was revised to take account of the following new product:
  - CICS Transaction Server for z/OS Version 2 Release 2
- Support for the new ECI over TCP/IP function was described in Table 5 on page 13.

Changes for the seventh edition

The more significant changes for this edition were:

- The book was revised to take account of the following new product:
  - CICS Transaction Server for z/OS Version 2 Release 1
- References to CICS/ESA Version 3 were removed, because this product is no longer supported.

Part 1. Introduction to CICS interproduct communication

This part includes the following chapters:

- Chapter 1, “CICS interproduct communication,” on page 3 introduces CICS interproduct communication, and describes the documentation plan used for each CICS product.
- Chapter 2, “CICS communication support,” on page 5 describes the CICS intercommunication functions, and shows what functions are supported between any pair of CICS systems.
- Chapter 3, “CICS Clients,” on page 11 introduces the CICS client products, and shows what functions are supported with each CICS server.
- Chapter 4, “Data conversion,” on page 15 explains why data conversion is necessary, explaining its concepts and terminology.
- Chapter 5, “Configuring CICS for SNA communications,” on page 17 introduces the concepts and practice of system configuration for SNA communications.

Chapter 1. CICS interproduct communication

This manual describes how you can connect CICS systems, and explains what functions those connected CICS systems can use. The following groups of CICS systems are discussed:

- CICS on System/390
- CICS Transaction Server for Windows
- CICS on Open Systems
- CICS/400

The documentation plan

This manual provides an overview of how all the CICS products communicate. Each CICS product has its own intercommunication manual which provides greater detail on how that product is configured and what functions are available to it.

The configurations with all the CICS products, and the associated documentation, are shown in Figure 1 on page 4. This shows each CICS product in communication with another product of the same type, and then all products being interconnected. The highlighted numbers in the figure refer to the following manuals:

1. *CICS on System 390 Intercommunication Guide*. (This means the Intercommunication Guide for your CICS on System/390 product.)
2. *CICS Family: Communicating from CICS on System/390*
3. *CICS TS for Windows, Intercommunication*
4. *CICS on Open Systems Intercommunication Guide*
5. *CICS/400 Intercommunication*

The order numbers of these manuals are listed in “Bibliography” on page 69.

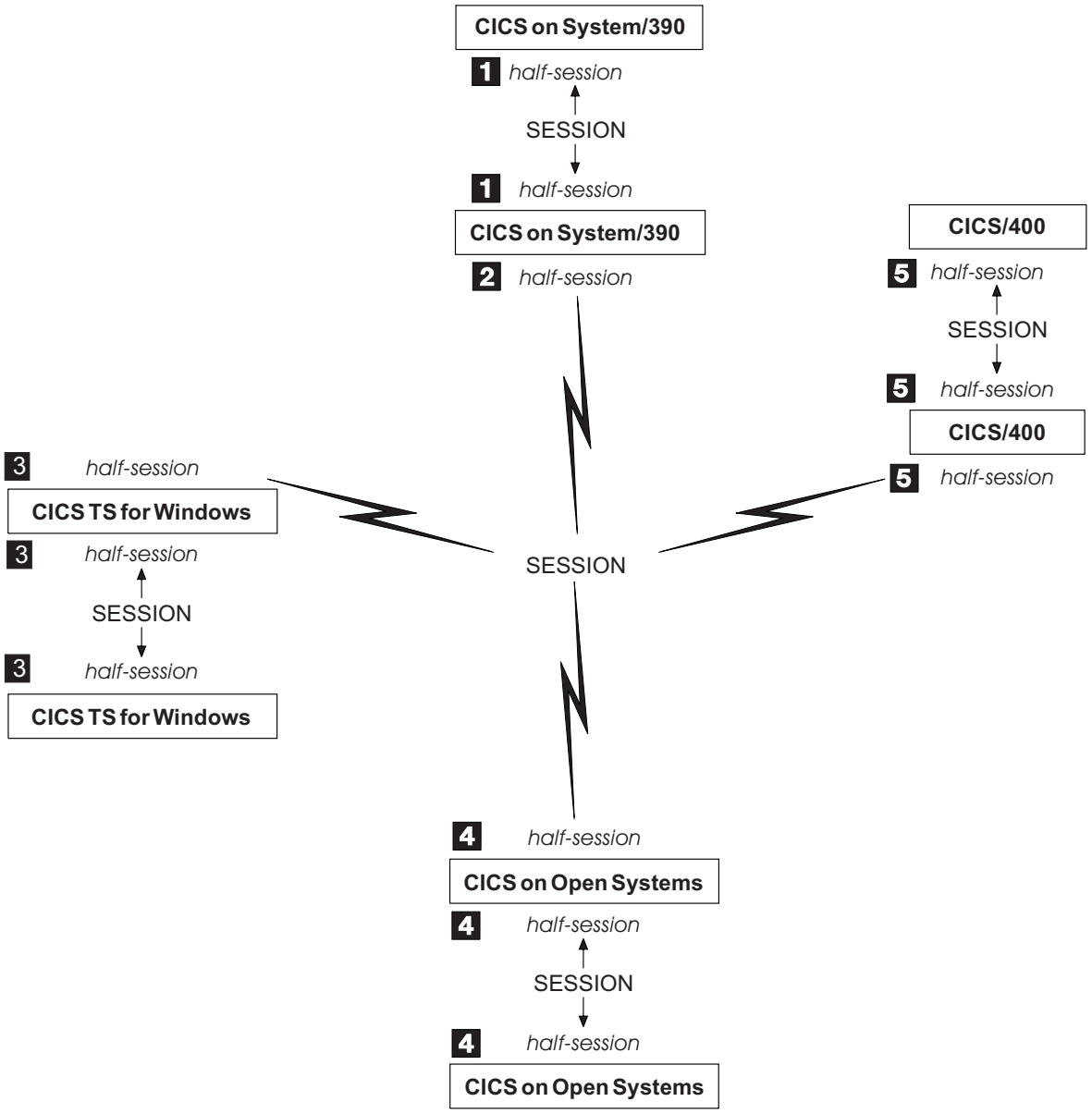


Figure 1. Sessions between CICS systems. This figure shows the coverage of each of the CICS intercommunication books. The numbers refer to the books in the "list of books" on page 3.

Chapter 2. CICS communication support

This chapter has two sections:

- "What is a product's communication ability?" is an introduction to intersystem communication terminology, and to the CICS communication functions.
- "CICS product communication support" on page 7 shows the ways in which any pair of CICS products can communicate.

What is a product's communication ability?

When planning your communication functions between different CICS systems you must consider:

- The intersystem communication functions that the CICS products support
- The communication protocols that are supported
- The synchronization levels that are supported
- What data conversion support will be required

These questions are examined in the following sections.

The CICS intersystem communication functions

CICS intersystem communication supports five basic functions:

Function shipping
: enables an application program running in one CICS system to access data resources (such as files and queues) that are owned by another CICS system.

Transaction routing
: enables a terminal connected to one CICS system to run a transaction in another CICS system.

Distributed program link (DPL)
: enables an application program executing in one CICS system to link (pass control) to a program in a different CICS system. The linked-to program executes and returns a result to the linking program.

Asynchronous processing
: enables a transaction executing in one CICS system to start a transaction in a different system. The two transactions then execute independently of each other.

Distributed transaction programming
: enables a transaction running in one CICS system to communicate with transactions running in other systems. The transactions are designed and coded specifically to communicate with each other.

For more information about these functions, see Part 2, "CICS intercommunication functions," on page 25.

Communication protocols

If two systems are to communicate successfully they must use a common set of rules that both understand. A communications protocol is such a set of rules that defines, for example, a set of standard requests and responses, and the order in which they can be sent.

For CICS products, three communication protocols are important:

SNA LU TYPE 6.2

LU TYPE 6.2 (LU 6.2) is a Systems Network Architecture (SNA) protocol that supports both system-to-system communication and system-to-device communication. LU 6.2 is also known as Advanced Program-to-Program Communications (APPC).

CICS can make use of the AnyNet® product, which allows SNA flows to be transmitted over a TCP/IP network.

All CICS products support the LU 6.2 protocol.

TCP/IP

The Transmission Control Protocol/Internet Protocol (TCP/IP) is a set of protocols that are used for both LANs and Wide Area Networks (WANs).

TCP/IP is supported natively by CICS Transaction Server for Windows, CICS on Open Systems, and versions of CICS Transaction Server for z/OS from Version 2.2 onwards.

Note: CICS TS for z/OS Version 2.2 and later support native TCP/IP connections to clients. The CICS External Call Interface (ECI) is supported, but not the External Presentation Interface (EPI) nor the External Security Interface (ESI). See Table 5 on page 13.

All CICS on System/390 products except CICS TS for VSE/ESA and CICS/VSE 2.3 can make use of the AnyNet product, which allows SNA flows to be transmitted over a TCP/IP network.

NetBIOS

The Network Basic Input/Output System (NetBIOS) is a local area network (LAN) protocol for personal computers. It is supported by CICS Transaction Server for Windows.

You can use IBM® NetBIOS, or another NetBIOS emulator. For example, Novell's NetBIOS emulator provides NetBIOS flows over its Internetwork Packet eXchange (IPX) protocol.

Synchronization

During CICS interproduct communication, partner transactions may make logically-related updates to their data stores — data sets, databases, temporary storage, transient data, and so on. Data integrity would be lost if both transactions did not commit (or back out) the updates they made to their resources.

The process used to ensure data integrity is called synchronization. Synchronization has to prevent one transaction completing normally and committing its updates while its partner transaction abends and backs out its updates. Synchronization must also handle situations when network problems prevent the transactions from communicating and informing each other of their actions.

Synchronization levels

SNA APPC architecture defines three levels of synchronization, which it calls NONE, CONFIRM, and SYNCPOINT. CICS refers to these as synchronization levels 0, 1, and 2, respectively.

Level 0 Provides no synchronization support.

Level 1 Allows transactions to exchange confirmation requests which they may use to provide some degree of synchronization. CICS is not involved in this process.

Level 2 Provides system-level syncpoint exchanges.

Synchronization level 2 provides **two-phase commit**. In a two-phase commit process, one CICS system initiates the syncpointing and acts as coordinator for the operation. The coordinating system:

1. Asks each connected system to **prepare** to commit
2. When each system has signalled readiness, it tells each to **commit**, *or*, if any resource manager signals that it cannot commit, it tells each to **back out**.

The synchronization level that two connected CICS systems can use is established when they first establish the connection. A connection established at synchronization level 2 can support a synchronization level 0, 1, or 2 conversation, and a connection established at synchronization level 1 can support a synchronization level 0 or 1 conversation.

The synchronization level you can use depends upon the capabilities of the particular CICS systems, and the capability of the network that they are using for the connection. The synchronization levels for all pairings of CICS systems is summarized in "CICS product communication support."

Data conversion

CICS products use two interchange codes for character data representation, EBCDIC and ASCII. Data in CICS on System/390 products and CICS/400 is held in EBCDIC format. Data in CICS Transaction Server for Windows and CICS on Open Systems is typically held in ASCII format.

Support of an intersystem communication function between two products requires support for any necessary data conversion. For further information, see Chapter 4, "Data conversion," on page 15.

CICS product communication support

This section summarizes the support each CICS product provides when it is in communication with another CICS product of the same type, and when it is communicating with each of the other CICS family products.

CICS on System/390 interproduct communication

Functions supported

CICS on System/390 supports all CICS intersystem communication functions.

Communication protocols supported

All CICS on System/390 products support SNA.

All CICS on System/390 products except CICS TS for VSE/ESA and CICS/VSE 2.3 can make use of the AnyNet product, which allows SNA flows to be transmitted over a TCP/IP network.

Only CICS TS for z/OS Version 2.2 and later support native TCP/IP connections to clients. The CICS External Call Interface (ECI) is supported, but not the External Presentation Interface (EPI) nor the External Security Interface (ESI). See Table 5 on page 13.

Synchronization level supported

CICS on System/390 can support synchronization levels 0, 1, and 2. The synchronization level used depends upon the support available in the partner system. This is summarized in Table 1.

Table 1. CICS on System/390 synchronization level support

	CICS on System/390	CICS on Open Systems	CICS/400	CICS Transaction Server for Windows
Maximum synchronization level supported:	2	2	2	1

CICS Transaction Server for Windows interproduct communication

Functions supported

CICS Transaction Server for Windows supports all CICS intersystem communication functions.

Communication protocols supported

CICS Transaction Server for Windows supports SNA, TCP/IP, and NetBIOS. The communication protocol that can be used is dependent upon the support available in the partner system. This is summarized in Table 2.

Table 2. CICS Transaction Server for Windows communication protocol support

	CICS Transaction Server for Windows	CICS on Open Systems	CICS on System/390	CICS /400
Communication protocols supported:	SNA TCP/IP NetBIOS	SNA TCP/IP	SNA	SNA
Note: Distributed transaction processing is supported only by SNA.				

Synchronization level supported

CICS Transaction Server for Windows supports only synchronization levels 0 and 1, and either level can be used with any partner CICS system.

CICS on Open Systems interproduct communication

Functions supported

CICS on Open Systems supports all CICS intersystem communication functions.

Communication protocols supported

CICS on Open Systems supports SNA and two types of TCP/IP:

- **CICS family TCP/IP** for communication with other CICS on Open Systems regions, CICS Transaction Server for Windows, and CICS Clients.

  CICS family TCP/IP supports only synchronization levels 0 and 1, and it cannot be used for distributed transaction processing.
- **Encina PPC TCP/IP** for communication with other CICS on Open Systems regions.

  Encina PPC TCP/IP supports synchronization levels 0, 1, and 2, and can be used for distributed transaction processing.

The choice of communication protocol depends upon the function supported by the partner system, and is summarized in Table 3.

Table 3. CICS on Open Systems communication protocol, and synchronization level support

	CICS on Open Systems	**CICS Transaction Server for Windows**	**CICS on System/390**	**CICS/400**
Communication protocols supported:	SNA TCP/IP (1) TCP/IP (2)	SNA TCP/IP(1)	SNA	SNA
Maximum synchronization level supported:	2	1	2	2
Note: 1. TCP/IP (1) - CICS family TCP/IP support 2. TCP/IP (2) - Encina PPC TCP/IP support				

Synchronization level supported

CICS on Open Systems can support synchronization levels 0, 1, and 2 when using SNA and Encina PPC TCP/IP, and levels 0 and 1 when using CICS family TCP/IP. The synchronization level that CICS on Open Systems supports with other CICS systems is summarized in Table 3.

CICS/400 interproduct communication

Functions supported

CICS/400 supports all CICS intersystem communication functions.

Communication protocols supported

CICS/400 supports only SNA.

Synchronization level supported

CICS/400 supports synchronization levels 0, 1, and 2. The synchronization level you use depends upon the support available in the partner system. This is summarized in Table 4.

Table 4. CICS/400 synchronization level support

	CICS/400	**CICS on System/390**	**CICS on Open Systems**	**CICS Transaction Server for Windows**
Maximum synchronization level supported:	2	2	2	1

Chapter 3. CICS Clients

Terminology

In this book, we use the term *CICS Clients* as a generic term for:

- The CICS Universal Client (for Windows NT, Windows 2000, and Windows XP)
- The CICS Client elements of the CICS Transaction Gateway products (which are available on all the platforms listed in "CICS Clients" on page 12)
- The client daemons of the CICS Transaction Gateway products

CICS Clients allow a workstation to access the transactions and resources in a CICS system.

The CICS Clients are not themselves full-function CICS systems, but they contain functions that enable them to access the resources of CICS systems running on other machines in the network. The CICS systems to which Clients are connected are known as **CICS servers**.

There is a CICS Client for many different operating systems. These are described in "CICS Clients" on page 12.

Functions that the CICS Clients provide

CICS Clients provide a set of functions for client/server computing. This section gives an overview of the most important functions; it is not meant to be exhaustive.

The External Call Interface

The External Call Interface (ECI) is an application programming interface (API) that allows a non-CICS program running on a workstation to call a CICS program located on a CICS server. This enables the program to make use of existing server routines that could be used, for example, to make enquiries on a database.

The client program can make the following types of call to a CICS server:

- Program link calls, which can be **synchronous** (that is, the calling program waits for a response from the linked-to program), or **asynchronous** (that is, the two programs continue to execute independently). The client program can issue a number of such calls, which can all run within the same unit of work (UOW), or they may run as individual units of work.
- Calls to retrieve a response from a previous asynchronous call.
- Calls that return a value indicating the status of the CICS system. This allows an application to test for availability of the CICS server or to monitor it by waiting for a change in its status.

The External Presentation Interface

The External Presentation Interface (EPI) is an API that allows a client program to appear to a CICS server as a 3270 terminal. You could, for example, use the EPI to enable workstation users to access CICS server transactions written for 3270 terminals, without changing the server code.

The client program can start CICS transactions and send and receive standard 3270 data streams to and from the transactions. It can present the 3270 data to the user by emulating a 3270 terminal, or by means of a graphical user interface such as Windows (Windows clients).

The EPI consists of a set of calls that can be made from an application program. The calls are provided in a library that is linked to the application. Among the functions available are calls to:

- Initialize the EPI.
- Terminate the EPI.
- Obtain a list of CICS servers to which a terminal may attach.
- Attach a pseudo-terminal.
- Detach a pseudo-terminal.
- Start a transaction for a terminal.
- Send data from a terminal to a transaction.
- Obtain details of an *event* that has occurred for a terminal. An example of an event is when the transaction is expecting a reply from the terminal.
- Obtain detailed error information for the last error that occurred for a terminal.

The External Security interface

The External Security Interface (ESI) is an API that allows a non-CICS Client program to verify and change the passwords used by Clients to connect to a CICS server.

Terminal emulation

CICS Clients can run 3270 terminal emulators. A client terminal emulator transmits or receives standard CICS transaction routing flows to or from a CICS server. This allows a user to interact with the server, and run transactions, as if the client were a locally-attached 3270 terminal.

It is possible to run multiple terminal emulators on a single client. The emulators can be connected to the same CICS server, or to different servers.

Users can customize the colors and keyboard mapping of their emulators. Double-byte character sets (DBCS) are supported but note that CICS clients attached to CICS/400 servers do not support double-byte character sets.

CICS Clients

There is a CICS Client for each of the following operating systems:

- AIX
- Microsoft® Windows XP
- Microsoft Windows 2000
- Microsoft Windows NT
- Solaris
- HP-UX
- Linux 390

Each Client can attach to any or all of the following CICS servers:

- CICS Transaction Server for z/OS
- CICS Transaction Server for OS/390
- CICS Transaction Server for VSE/ESA
- CICS/VSE Version 2.3

- CICS/400
- CICS Transaction Server for Windows
- CICS on Open Systems

Supported functions and protocols

Table 5 shows the functions and communication protocols that are supported on each CICS Client–CICS server link.

Table 5. IBM CICS Client function and protocol support

CICS Servers	ECI	EPI	Terminal Emulator	Auto install	LU 6.2	NetBIOS	TCP/IP
CICS Transaction Server for Windows	Y	Y	Y	Y	Y	Y	Y
CICS on Open Systems	Y	Y	Y	Y	Y	-	Y
CICS/400	Y	Y	Y	Y	Y	-	-
CICS TS for z/OS Version 2.2 and above	Y	Y	Y	Y	Y	-	Y[1] [2]
CICS TS for z/OS Version 2.1	Y	Y	Y	Y	Y	-	Y[2]
CICS TS for OS/390	Y	Y	Y	Y	Y	-	Y[2]
CICS TS for VSE/ESA	Y	Y	Y	Y	Y	-	-
CICS/VSE V2.3	Y	-	-	Y[3]	Y	-	-

Notes:

Y Function or protocol is supported

- Function or protocol not supported

ECI External Call Interface

EPI External Presentation Interface

Autoinstall
User does not need to predefine the client connection to the server

1. Native TCP/IP is supported. Using ECI over TCP/IP, all clients are supported, but only the ECI (not the EPI nor the ESI) can be used.
2. TCP/IP is supported through the use of IBM TCP62 and the AnyNet feature of VTAM®. Using this method, only the Windows clients are supported, but both the ECI and EPI can be used.
3. Only single-session LU6.2 connections can be autoinstalled.

Chapter 4. Data conversion

The CICS family of products runs on a variety of operating system and hardware platforms. Numeric and character data can be held in different ways in different systems. When intercommunication between CICS products requires the transfer of data, data conversion may be necessary.

This chapter gives an overview of data conversion that serves as an introduction to the task-oriented information in the CICS Intercommunication manual for your CICS product. It does not describe the mechanics of the process.

CICS products do most of the necessary data conversion automatically. Some conversion requires no setup by the user, for example, file names in function shipping requests. For other data, such as file records, you supply resource definitions that identify the types of conversion to be applied to specified fields in data records. For non-automatic conversion, exits or user-replaceable conversion programs are available in some products.

Numeric data

The main ways of holding numeric data are binary and packed decimal. If these types of data are held differently in two CICS systems, resource definitions in each system may be sufficient to cause automatic data conversion.

In some cases, you can arrange for one system to hold data in a way that is compatible with the other, avoiding the need for conversion. For example, if a CICS Transaction Server for Windows COBOL/2 application program is compiled with the IBMCOMP and SIGN EBCDIC directives, packed decimal data is held in System/390-compatible format.

Automatic data conversion is not available in some cases. For example, conversion between workstation packed decimal data and System/390 packed decimal data requires user-written conversion code. This code can be inserted in a user-replaceable conversion program in the CICS on System/390 product.

See the platform-specific CICS intercommunication guides for a detailed explanation of which CICS system is responsible for converting the data when two systems are exchanging data.

Character data

The smallest unit of computer data is a **bit**, or binary digit. A bit has only two possible values, 0 or 1. To represent character data, bits must be grouped. The most common grouping is the 8-bit **byte**, providing up to 256 different characters.

A named **character set** is a particular set of characters—“Latin-1”, for example, is a set of Western uppercase and lowercase letters, numbers, and a selection of symbols.

A **single-byte character set** (SBCS) allows the representation of up to 256 characters, each character being represented by a single byte.

A **double-byte character set** (DBCS) allows the representation of more than 256 characters, each character being represented by a pair of bytes. Some languages

require characters to be represented by multiple bytes in what is called a **multi-byte character** set (MBCS); here, each character is represented by up to 4 bytes.

The different languages in some countries (such as Japan) may use both SBCS and DBCS.

A computer program or operating system must assign a byte, 2-byte, or multi-byte value to each character that it wants to represent. Several conventions exist for character representation. In this chapter, these conventions are called interchange codes. Two different interchange codes are used in the hardware platforms on which CICS products run:

- Extended Binary-Coded Decimal Interchange Code (EBCDIC), typically used on System/370 System/390 and AS/400® machines.
- American National Standard Code for Information Interchange (ASCII), typically used in personal computers and RISC System/6000 (RS/6000®) machines.

Code pages

A **code page** defines the code points for the characters in a particular character set. It consists of a list of byte values or 2-byte values and the characters they represent. The EBCDIC and ASCII interchange codes include more than one code page, so data conversion can be necessary even between two systems that use the same interchange code.

A **Coded Character Set Identifier (CCSID)** is a combination of a character set and its associated code page. A CCSID may be composite; that is, it may contain multiple character sets and code pages—for example, Katakana (1-byte) and Kanji (2-byte).

Most CICS products use in-built conversion tables to handle conversion between common code pages. Some products allow you to define your own conversion tables. For nonstandard conversion, you can supply a user-written conversion program.

For detailed information about data conversion, see the CICS Intercommunication manuals for your CICS products.

Notes:

1. **CICS for OS/400®** uses a comprehensive set of conversion tables provided by AS/400, and does not support user-defined tables.
2. **CICS on Open Systems** uses the operating system's ***iconv*** routines, which provide data conversion by both table-driven and algorithmic methods. A comprehensive set of converters is supplied. A CICS on Open Systems user can create or customize a converter.

Chapter 5. Configuring CICS for SNA communications

Before two systems can communicate, they each need to know:
- The identity of the other system
- The characteristics of the other system
- The communication methods to be used
- The services and functions to be used

This chapter shows how you specify that information when configuring CICS systems for SNA communication.

There are a number of stages in the process of configuring a system for communication:

1. Establish a basic connection between the two systems that can be used by a simple test application.
2. Build on that initial configuration to enable support for your own applications.
3. Refine that working configuration to expand facilities and enhance performance.

This chapter gives you the information on SNA and CICS that you will need to establish that first basic configuration. You should not consider proceeding beyond that first stage until you have completed and successfully tested a basic, simple SNA connection.

There are other manuals that give broader and more detailed information about SNA. There are manuals that explain the SNA architecture, manuals that explain how individual products have implemented the architecture, and manuals that explain how to configure various combinations of different systems. There is a list of useful books in “SNA books” on page 70.

The scenario chosen for the discussion is based on an established SNA network of interconnected CICS on System/390 systems. To this we will be adding a CICS for AIX system, which will use SNA to connect to one of the CICS on System/390 systems.

The rest of this chapter contains:

Introduction to SNA terminology

IBM’s System Network Architecture (SNA) defines a set of rules that systems use to communicate. These rules define the layout of the data that flows between the systems and the action the systems take when they receive the data. The layout of the data is known as the **format**, and the action the systems take when they receive that data is known as the **protocol**. Together, formats and protocols constitute the architecture.

SNA does not specify how a system should implement the architecture. Indeed, a fundamental objective of SNA is to allow systems that have very different internal hardware and software design to communicate. The only requirement is that the network flows meet the rules of the architecture.

One consequence of this independent implementation is that each system brings its own terminology into its SNA implementation, and this can lead to a confusion of terminology when you start to establish a connection between two different systems. This section introduces those particular SNA concepts and terms that you will encounter when configuring CICS systems.

SNA concepts

network
A network is a collection of interconnected computers and devices, together with the physical and logical connections that connect them.

network address
A network address is a unique code that is assigned to every device in a network. With a personal computer, for example, it is likely to be the medium access control (MAC) address in its network adapter card.

link and node
A link connects two nodes, where a node is any device in a network that transmits and receives data.

logical unit (LU)
A logical unit represents the logical destination of a communication data flow. The formal definition of an LU is that it is the means by which an end user gains entry into a network, and an end user is defined as the ultimate source, or destination, of data flow in a network. SNA supports several different types of LUs. These are grouped together in numbered **LU types**, such as LU type 2 for 3270 display terminals, and LU type 4 for printers. The LU type for CICS-to-CICS communication is LU type 6.2, and is frequently referred to as **advanced program-to-program communication** (APPC). Each LU is given a unique name that identifies it in the network, and this is referred to as the **LU name**.

Sometimes the name of the network that the LU is in is appended to the name of the LU. It is then known as the **network-qualified LU name**, or the **fully qualified LU name**, and it takes the form *network-name.LU-name*.

physical unit (PU)
A physical unit is the hardware and software components in a device that manage its network resources. LUs reside within a PU, and one PU may hold many LUs. There are several different types of PU. VTAM running in a mainframe host is a PU type 5, and NCP running in a 37x5 network controller is a PU type 4. When workstations connect together in a peer-to-peer manner they act as PU type 2.1. When a workstation connects to a mainframe host in a hierarchical manner, it acts as a PU type 2.0. The PU type 2.1 is described as an independent node (because it is independent of a mainframe host), and the PU type 2.0 is a dependent node.

control point (CP)
The SNA concept that relates LUs to PUs is a control point. A CP can be thought of as that part of a PU that manages the LU.

exchange identification (XID)
Associated with the PU and CP is the exchange identification. XID is actually the name of a data flow that PUs exchange during the early stages of their attempt to establish a connection, but, in the context of SNA configuration, XID refers to one of the fields within that XID flow. It is a hexadecimal field which the PUs use to confirm the identity of each other.

session
SNA uses the term session to refer to various types of data flow in a network. To avoid ambiguity, it should always be qualified by a description of the type of data flow, for example CP-CP session. However, when used by CICS for APPC, it can be assumed to refer to data flow between LUs, and so is an LU-LU session. There are usually several sessions between any two LUs, and these are known as *parallel sessions*.

connection
CICS uses the term connection to refer to a group of sessions that connect two CICS systems.

transaction program (TP)
In SNA, the term transaction program refers to the application program in an APPC environment. The TP uses the LU to gain access to the network. When CICS is using APPC, the TP is a CICS transaction.

conversation
In SNA, the term conversation describes the communication between two TPs. That is, when two APPC TPs are in communication, they are said to be holding a conversation. Conversations flow on LU-LU sessions. Each conversation is allocated a session for its own private use. When the conversation ends, the session is free to be used by another conversation. There can only be one conversation between any two TPs, but one TP could have multiple conversations with different TPs.

mode
There may be a choice of many routes and paths in the network that an LU-LU session could use. One route might be suitable for large volumes of batch data, another might be reserved for smaller, high speed exchanges. SNA allows for these different route types to be grouped into *modes*. A TP can select an appropriate mode when it first establishes a conversation. The conversation will flow on an LU-LU session that follows the route defined by the node.

local, remote, partner
These terms are used in many contexts. For example, you could refer to the local system, the remote system, and the partner system. When you are at a workstation, you regard the workstation as the local system, the machine your workstation is communicating with is the remote system, and that remote system is your partner. However, these terms are all relative to your point of reference, so the remote system regards itself as being a local system with your workstation being its remote system, and each system is a partner of the other.

In summary, it may help you to understand these terms if you visualize a session as being like a pipe that links LUs. When the LU inserts data into the pipe it is inevitable that the data will be passed to the LU at the other end of the pipe. Pipes with similar characteristics are grouped together in a mode. These pipes are created when the systems are first initialized and SNA is started, or when a TP first requests the use of one. Several pipes usually run in parallel between two CICS systems. When a CICS transaction wants to hold a conversation with a transaction on another system, it requests the use of a pipe. It is given the sole use of a pipe. When the transaction ends, the pipe is returned to the pool and can be allocated to the next transaction.

SNA products

CICS does not provide its own SNA support. That is, CICS does not structure the network data flows into the SNA format, nor is it responsible for initiating the SNA protocol when a data flow is received. These SNA functions are provided by a

separate SNA product, and CICS uses the services of that product. On a System/390 host the SNA product is VTAM, and on an AIX machine it is AIX SNA Server/6000 (AIX SNA).

When you configure your systems, you have to configure both the CICS product and the associated SNA product.

Preparing for SNA configuration

To establish a SNA connection, each system must:
1. Define itself to the network
2. Define the connection to its partner
3. Ensure that the remote system has a definition of the local system

You could regard the first step as the system registering its name and address with the network; the second step as a system specifying its intended partner's name and address so that the network can establish a link to it; and the third step as your partner registering your system's name so that it will recognize your system's requests when it receives them.

Matching parameters

The names that are used to define system resources include the network name, LU name, PU name, CP name and XID.

Some of these names are used as parameters when configuring the CICS product, and others are used as parameters when configuring the SNA product. Some are used in both products. Some are used in one product on the local system and in another product on the remote system. When a parameter is used in more than one place, it is important that the same name is used. That is, the parameters must match.

Mode name

The partners must use matching mode names. Any mode may be used, but a convenient one is the **#INTER** architected mode.

Alias names

Some resources (typically the LU) may have an alias or local name as well as a real name. The alias is only used internally within a local system. The partner system recognizes only the real name, and it is that name that must match across the systems.

Platform specific implementation

Depending upon the particular network configurations and SNA products being used, you may find that not all the parameters are needed. For example, the XID may be used rather than the CP name to identify the partner. In other circumstances when, for example, there is only one LU in the PU, the CP may have to have the same name as the LU. You should refer to the product-specific manuals for clarification of these points when you are ready to proceed beyond this first basic configuration.

The scenario

In the scenario chosen for this discussion, we assume that there is an established SNA network of interconnected CICS on System/390 systems into which we are adding a CICS for AIX system. This chapter is written from the perspective of the AIX systems' administrator who connects the workstations to the mainframe host.

To avoid the potential problem of duplicate names being used, and to assist management of the network, the VTAM system administrator may act as coordinator for the network resource names. You therefore need to agree the network names of your workstations with the VTAM systems administrator.

Configuration details

The tables in the following sections show where the resource names are configured in the CICS and SNA products. The information is generally given by naming the attribute in which the resource name is specified as a parameter, and giving the name of the definition statement or front-end panel in which that attribute appears. If you need further information on the configuration tools used by the different product you should refer to the product-specific manuals listed in “SNA books” on page 70.

If the tables show that a name is specified in both CICS and the SNA product, then the same matching name must be used in both. When a name is not defined in one of the products, this is indicated by a dash (-).

There is not complete symmetry in the configuration details in the mainframe host and the workstations. For example, the workstation configuration requires that it knows the address of the mainframe host, but the mainframe is not given the address of the workstation. This is because the workstations are regarded as the *calling* system, and the mainframe the *listener*. This means that the session will be initiated by the workstation calling the mainframe. Therefore the information the workstation needs about the mainframe is different from the information the mainframe needs about the workstation.

Mainframe host configuration

CICS on System/390 and VTAM will have already defined their network name and LU name to the network. Table 6 shows where these names are defined in CICS on System/390 and VTAM. Here they refer to the mainframe’s local resources. You use these names when you later configure the workstations where they refer to the workstation’s partner resources.

The workstations need to know the address of the mainframe host. This is a 12-character hexadecimal code assigned to the front-end processor. This code is not used in the configuration of CICS on System/390 or VTAM and so is not shown in the table. The network administrator will be able to tell you the address.

Table 6. Defining local resources to CICS on System/390 and VTAM.

	CICS on System/390	**VTAM**
Network name	-	NETID= attribute in VTAM startup procedure
LU name	APPLID= attribute in the SIT	The label on the APPL statement that defines CICS to VTAM

Defining the workstations to CICS on System/390 and VTAM

You now have to agree with the CICS and VTAM system administrator on the resource names that you use for your workstations. The workstations have to be defined to CICS on System/390 and VTAM so that CICS on System/390 recognizes and accepts session initiation requests from them. For each workstation the system administrator creates a new CONNECTION definition in CICS, and new PU, LU, and MODEENT definition statements in VTAM.

Table 7 shows where the LU, PU, CP and XID names are defined. Remember that these are the names of the mainframe's partner resources. You use these same names when configuring the workstations, where they refer to those workstation's local resources (see Table 8).

Table 7. Defining the workstation to CICS Transaction Server and VTAM.

	CICS on System/390	**VTAM**
LU name	NETNAME attribute in the CONNECTION definition	The label on the LU statement
PU name	-	The label on the PU statement
CP name	-	CPNAME= attribute in the PU statement
XID	-	IDBLK= and IDNUM= attributes in the PU statement
Mode	MODENAME attribute in the SESSIONS definition	LOGMODE= attribute in the MODEENT statement
Note: The workstation is in the same network as the mainframe, and so the network name it uses is the same as that defined to VTAM in Table 6 on page 21.		

AIX machine configuration

To configure the AIX machine, you must:
1. Define it to the network
2. Define a connection to its partner

AIX is a versatile platform and offers many ways of connecting CICS to the network. The AIX SNA product can be in a different machine from the CICS for AIX product. It can be using an Encina PPC Gateway, with CICS using TCP/IP to communicate with the gateway before gaining access to the SNA network. The machines can be in different networks. The example shown here describes the basic case of a single machine, that is running both CICS for AIX and AIX SNA, and that is in the same network as the mainframe.

Defining the AIX machine to the network

Table 8 shows where the local resources are defined to CICS for AIX and AIX SNA. You use the names you have previously agreed with the mainframe system administrator that are shown in Table 7.

Table 8. Defining local resources in CICS for AIX and AIX SNA

	CICS for AIX	**AIX SNA**
Network name	LocalNetworkName= attribute in the Region Definition	Local network name attribute in the Initial Node Setup panel
LU name	LocalLUName= attribute in the Region Definition	Local LU name attribute in the Add LU 6.2 Local LU Profile and the Side Information Profile panels
CP name	-	control point name attribute in the Initial Node Setup panel
XID	-	XID node ID attribute in the Initial Node Setup panel
Mode	DefaultSNAModeName= attribute in the Communication Definition	Profile name attribute in the Add LU 6.2 Mode Profile panel

Defining the connection to CICS on System/390

Table 9 shows how you define the connection to the mainframe. All you have to provide is the LU name (which is effectively the name of the CICS on System/390 system), and the hardware address.

Table 9. Defining the connection to CICS on System/390

	CICS for AIX	**AIX SNA**
LU name	RemoteLUName attribute in the Communications Definition	Fully qualified partner LU name attribute in the Add LU 6.2 Partner LU Profile panel
Address	-	Link address attribute in Link station information in the Initial Node Setup panel

Configuring CICS for SNA—next steps

This chapter shows what has to be done to configure your workstations for SNA communications. Only information on the logical connections has been given. You need to add information on the particular physical connections you are using (such as whether you are using a Token Ring or Ethernet LAN, or a SDLC connection to the mainframe).

The *CICS on Open Systems Intercommunication Guide* gives complete configuration examples for various types of connections.

As stated in the introduction, your first aim must be to configure a basic setup and test it with a simple application. For instance:

- APING is the APPC equivalent of the TCP/IP PING program. It can be used independently of CICS as a stand-alone APPC application and so will test the configuration of your SNA products.
- The CICS Server ISC PING Transaction is available as SupportPac CC02 at this web site:

  www.ibm.com/software/ts/cics/txppacs/cc02.html

Only after you are satisfied that you have a working APPC link should you expand the configuration to include your own applications.

Part 2. CICS intercommunication functions

This part describes the CICS intercommunication functions. The information is intended for planners and analysts, and as an introduction for programmers.

This part includes the following chapters:

Chapter 6. Introduction to the CICS intercommunication functions

This chapter gives a simple description of the CICS intercommunication functions.

In “Summary of CICS intercommunication functions,” the figures and explanations deliberately gloss over technical details, but present the functions as seen by an application programmer. The important point is that function shipping, asynchronous processing, transaction routing, and distributed program link (DPL) are powerful distributed functions that are transparent to the application programmer. Applications can access local or remote resources and programs entirely under the control of definitions created by the systems programmer.

Distributed transaction programming is a more complex facility and application programmers need to be aware that a dialog is taking place with a remote transaction. The program logic must react to the current state of the dialog.

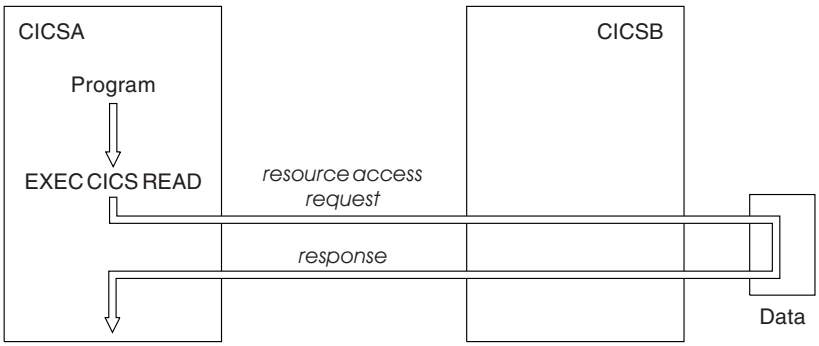
“Which intercommunication function?” on page 29 gives some guidelines for the selection of the correct intercommunication function for particular requirements.

Summary of CICS intercommunication functions

As an introduction for new CICS users, this section presents deliberately simplified definitions of the CICS intercommunication functions.

Function shipping

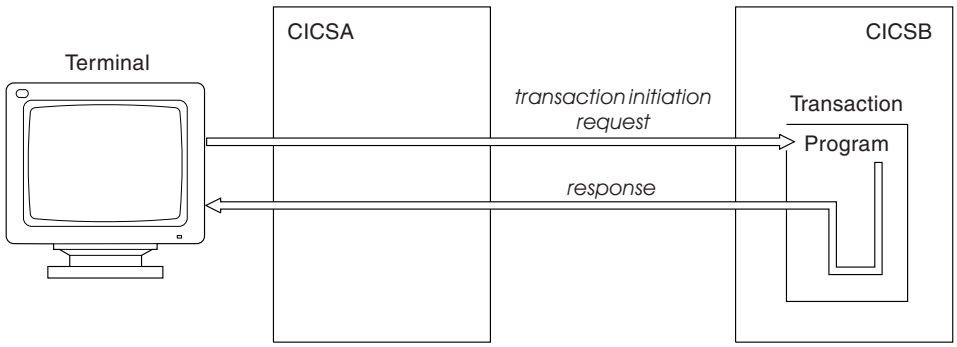
A program in system CICSA accesses resources (such as files or transient data queues) that are owned by remote system CICSB as though they were locally owned. The diagram shows a data-access request.



Asynchronous processing is an example of function shipping that does not access data – the shipped request is an EXEC CICS START command for a remote transaction.

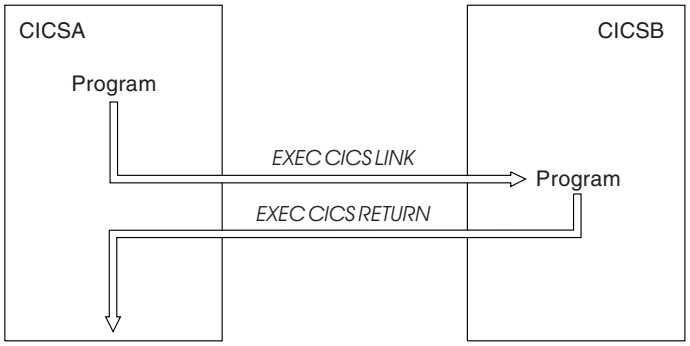
Transaction routing

A terminal attached to system CICSA runs a transaction in remote system CICSB as though it were a local transaction.



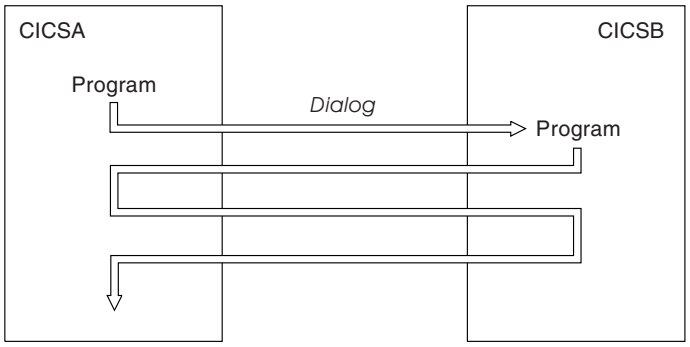
Distributed program link

A program in system CICSA links a program in remote system CICSB as though it were running in the local system.



Distributed transaction programming

Two programs, one in system CICSA and one in system CICSB, communicate synchronously with each other. This dialog is called a conversation.



Which intercommunication function?

To help you choose the right intercommunication function, the list below gives typical requirements and the functions that meet them:

- **A terminal user needs to run a transaction owned by a different system than the one to which he or she is connected**.

  Use transaction routing.

- **A transaction wants to read/write data owned by another system**.

  Use function shipping. However, function shipping has higher overheads than transaction routing, so it is better to use transaction routing unless the transaction accesses data in the local system as well as data in the remote system.

- **A transaction wants to read/write IMS™ or DL/I data to which another system has access**.

  Use distributed program link.

- **A transaction needs to signal a remote system that it should start a named transaction**.

  Use asynchronous processing.

- **A transaction wants several accesses to remote data, possibly with some processing between the accesses**.

  Use distributed program link if possible. This requirement can be met with several function shipping requests, but DPL minimizes the data flows on the network. If DPL cannot meet the whole requirement, a mixture of DPL and function shipping is more efficient use of the network than total reliance on function shipping.

- **An application requires communication between two or more systems with interdependent resource updates based on data exchanges**.

  Use distributed transaction programming.

Chapter 7. Function shipping

This chapter contains the following topics:

Introduction to function shipping

CICS function shipping enables CICS application programs to use EXEC CICS commands to:

- Access CICS files owned by other CICS systems by shipping file control requests.
- 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 START requests. This form of communication is described in Chapter 10, "Asynchronous processing," on page 51.

An illustration of a shipped file control request is given in Figure 2. In this figure, a transaction running in a CICS system, CICA, issues a file control READ command against a file called NAMES. From the resource definition table, CICA discovers that this file is owned by a remote CICS system called CICB. CICA 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.

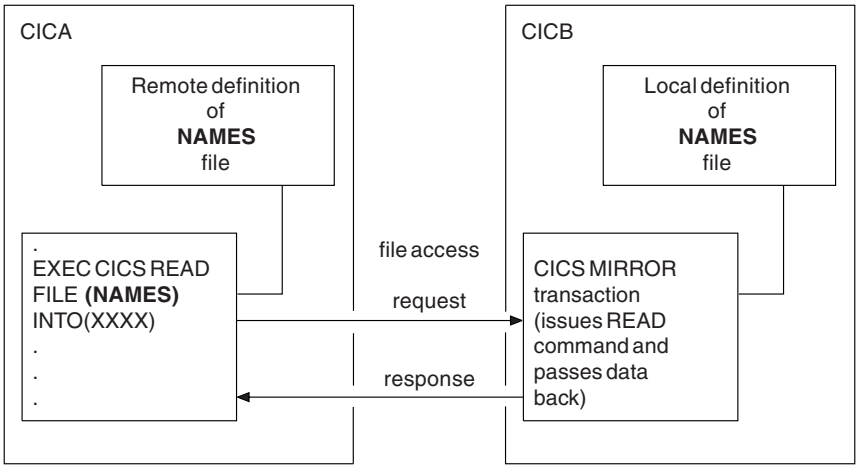


Figure 2. Function shipping

Transparency to application

An application that uses function shipping need not know the location of the requested resources; it uses file control commands, temporary storage commands, and so on, as if all resources are owned by the system in which the application runs. 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.

The definition of a remote resource can include both the name by which the resource is known in the remote system, and a different 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 resources exist with the same name on more than one system, but each contains data peculiar to the system on which it is located.

Application programs can use the SYSID option of various EXEC CICS commands to name remote systems explicitly. 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. Using SYSID in this way destroys the program's independence of the resource's location. The advantage is that ***any*** system, including the local system, can be named in the SYSID option. The decision whether to access a local resource or a remote one can be taken at execution time.

Remote resources that can be accessed

Function shipping requests can access the following remote resources:

- CICS file control data sets
- Temporary storage
- Transient data
- IMS and DOS DL/I databases (from CICS on System/390 systems).

CICS file control data sets

Function shipping allows access to files located on a remote CICS system. The following EXEC CICS commands are not supported:

- INQUIRE FILE
- INQUIRE DSNAME
- SET FILE
- SET DSNAME

Both read-only and update requests are allowed. Protection of data depends on the security facilities available to the different CICS products. Updates to remote recoverable files are not committed until the application program issues a syncpoint request or terminates successfully. Logically-related updates of local and remote files can be performed within the same logical unit of work, even if the remote files are located on more than one connected CICS system.

Care should be taken when designing systems that use remote file requests containing physical record identifier values (examples include BDAM, VSAM RBA, and files with keys not embedded in the record). Application programs in remote systems must have access to the correct values following updating or reorganization of such files.

IMS databases

Function shipping allows a CICS on System/390 transaction to access IMS/ESA® DM and IMS/VS DB databases associated with a remote CICS OS/390 system, or DL/I for VSE/ESA databases associated with a remote CICS/VSE system.

CICS Transaction Server for Windows, CICS/400, and CICS on Open Systems systems cannot access IMS or DL/I databases by function shipping, but can do so by distributed program link (see Chapter 9, “Distributed program link,” on page 47). The following discussion applies to CICS on System/390 systems only.

The IMS/ESA DM (DL/I) database associated with a remote CICS 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. In IMS/ESA DM, it is not possible to schedule more than one PSB for a single logical unit of work, even when the PSBs are defined to be on different remote systems. For this reason, logically-related DL/I updates on different systems cannot be made in a single logical unit of work.

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

Temporary storage and transient data

Many of the uses made of transient data and temporary storage queues in a single CICS system can be extended to an interconnected system environment. For example, a queue of records can be created in a remote system for processing overnight. Queues also provide a means of handling responses from remote systems, while keeping local terminals free to enter other requests. A response can be returned to a terminal when it is ready, and delivered to the operator when there is a lull in entering transactions.

Temporary storage

Function shipping enables application programs to send data to, or retrieve data from, temporary storage queues located on remote systems. A recoverable queue must be defined as recoverable in its local (resource-owning) system.

Transient data

An application program can access intrapartition or extrapartition transient data queues on remote systems. The resource definition in the requesting system defines the named queue as being on the remote system. The queue definition in the remote system specifies whether the queue is recoverable, and whether it has a trigger level and associated terminal.

If a transient data destination has an associated transaction, the named transaction must be defined to execute in the system owning the queue; it cannot be defined as remote. If a terminal is 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.

How function shipping works

Two CICS components implement function shipping:
- The CICS transformer programs
- The CICS mirror transaction.

Figure 3 illustrates how these components work together.

The transformer programs

If a CICS transaction issues a request to access a resource, the command level EXEC interface program determines whether the resource is owned by the local CICS system. If the resource is on another system, control passes to the function-shipping transformer program.

DL/I (EXEC DL/I or CALL DLI) requests use the DL/I interface, which also provides part of the transformer program's function.

A transformer program converts the request to a form suitable for transmission, and calls the intercommunication component to send the request to the resource-owning system.

The CICS intercommunication component sends the request to the remote system. On the first request to a particular remote system on behalf of a transaction, the communication component in the local system precedes the formatted request with the mirror-transaction identifier, in order to attach this transaction in the remote system. The local transformer program keeps track of whether the remote mirror transaction terminates, and reinvokes it as required.

When a reply is received from the remote system, a second transformer program decodes it. CICS uses the decoded reply to complete the original command-level request.

The remote system uses its own transformer programs in dealing with the request (see next section).

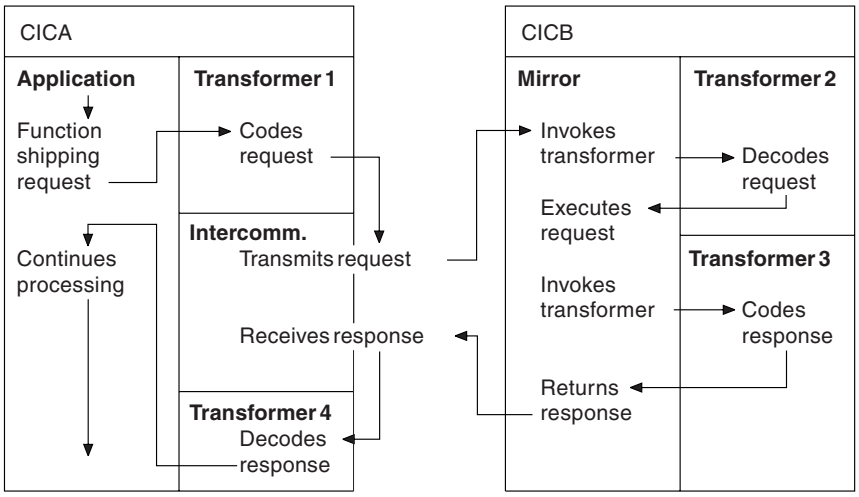


Figure 3. Relationship of mirror and transformers

The mirror transaction

A resource-owning system passes an incoming function-shipping request to the mirror transaction. The first request from a particular remote transaction causes the initiation of a new instance of the mirror transaction, which uses CICS intercommunication facilities to communicate with the requesting system.

Using a CICS transformer program, the mirror transaction decodes the formatted request, and executes the command. On completion of the command the mirror transaction uses a transformer program to construct a formatted reply, and returns this to the requesting system.

The mirror transaction remains active after sending its reply to the current command in any of the following cases:

- Execution of a future command depends on the retention of system-specific information established during execution of the current command, for example:
  REWRITE depends on prior READ UPDATE
  READNEXT depends on prior STARTBR
- Execution of a future command may depend on the retention of application-specific information established during execution of the current command, for example when a recoverable resource has been updated.
- The mirror remained active after replying to a previous command for one of the reasons above (the mirror then remains active until the end of the logical unit of work in the requesting system).

In other cases, the mirror terminates after replying to the current command.

An active mirror always terminates when the requesting transaction issues a synchronization request or terminates successfully. The mirror always terminates after executing a LINK command with the SYNCONRETURN option. For a further explanation of SYNCONRETURN, please refer to “Synchronization” on page 48.

Multiple mirrors

A transaction can access recoverable and nonrecoverable resources in any order, and is not affected by the location of recoverable resources (they could all be in different remote systems, for example). When a local transaction accesses resources in more than one remote system, the intercommunication component invokes a mirror transaction in each remote system to execute requests for the local transaction. Each mirror transaction follows the above rules for termination, and when the transaction reaches a synchronization point, the intercommunication component exchanges synchronization point messages with those mirror transactions that have not yet terminated (if any).

Chained mirrors

The mirror transaction uses the EXEC CICS 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 the specified remote system, which activates its own mirror transaction. This is called a **chained-mirror**.

Synchronization

CICS recovery and restart facilities ensure that when the requesting transaction reaches a synchronization point, any mirror transactions that are updating recoverable resources also take a synchronization point, so that changes to recoverable resources in remote and local systems are consistent. The CICS master terminal (or, with CICS/400, the control region) receives notification 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.

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

Function shipping examples

Figure 4 and Figure 5 on page 37 give examples to illustrate the lifetime of the mirror transaction.

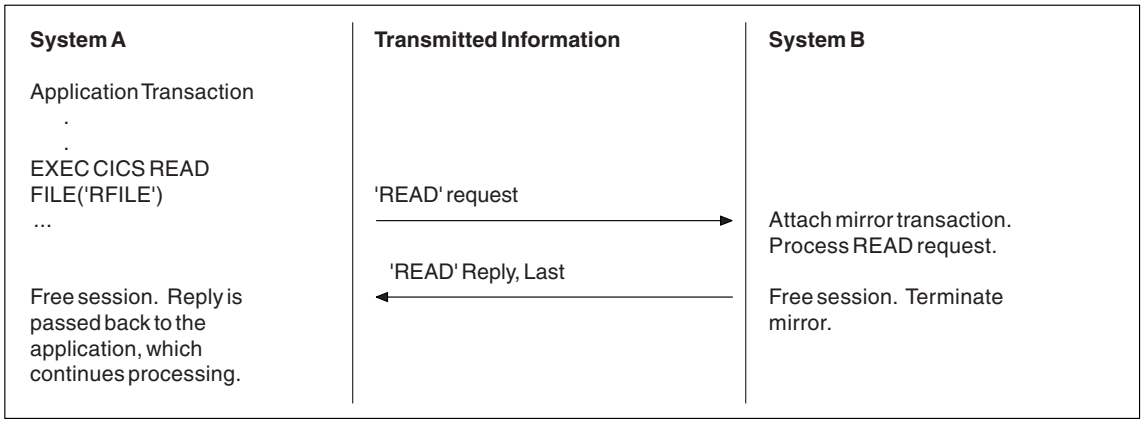


Figure 4. Function shipping—simple inquiry. Here, no resource is being changed; the session is freed and the mirror task is terminated immediately.

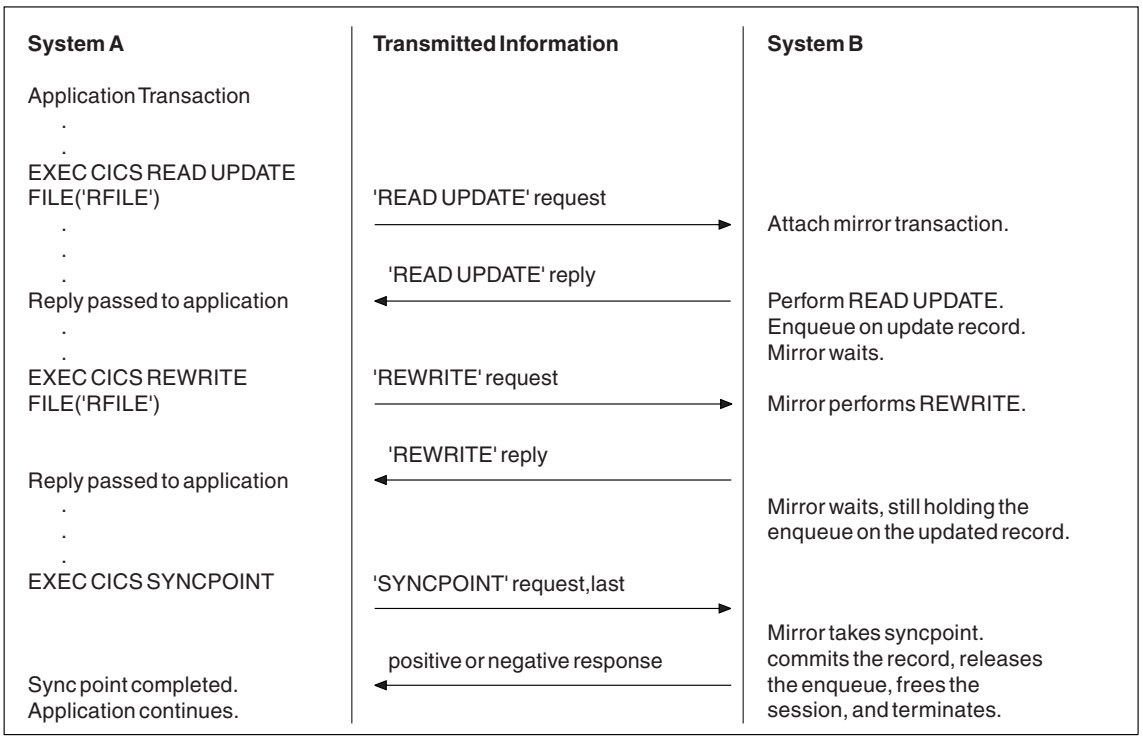


Figure 5. Function shipping—update. Because the mirror must wait for the REWRITE, it does not terminate until SYNCPOINT is received. Note that the enqueue on the updated record would not be held beyond the REWRITE command if the file was not recoverable.

In Figure 5 the mirror is long-running.

Chapter 8. Transaction routing

This chapter contains the following topics:

Introduction to 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 6 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**.

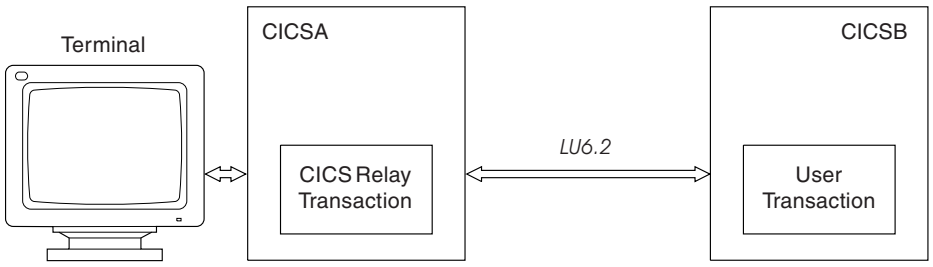


Figure 6. The elements of transaction routing

Two different CICS products must be connected by an LU6.2 link. Transaction routing over LU6.1 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 on System/390 are eligible for transaction routing, **except** those given in the following list:
- LUTYPE6.1 connections and sessions
- Pooled 3600 or 3650 pipeline logical units
- MVS™ system consoles.

CICS Transaction Server for Windows, CICS/400, and CICS on Open Systems support minimum BMS. (They support SEND TEXT.) Only CICS on System/390 systems support batch data interchange, standard BMS, and full BMS. Depending on these product capabilities, a user transaction can use CICS terminal control, BMS, or batch data interchange facilities to communicate with the terminal, as appropriate for the terminal or session type. Mapping and data interchange functions are performed in the application-owning system (CICS B in Figure 6). BMS paging operations are performed in the terminal-owning system (CICS A in Figure 6).

Pseudoconversations are supported (except when the terminal is an APPC session), and the various transactions that make up a pseudoconversation can be in different systems.

Initiating transaction routing

Transaction routing can be initiated in the following three ways:

1. A terminal sends a request to the CICS terminal-owning system to start a transaction. On the basis of an installed resource definition for the transaction (and possibly, in CICS on System/390 and CICS Transaction Server for Windows, on decisions made by a user-written dynamic routing program) the request is routed to an appropriate transaction-owning system, and the transaction runs as if the terminal were attached to the transaction-owning system.
2. A transaction started by automatic transaction initiation (ATI) acquires a terminal that is owned by another CICS system.
3. A CICS on System/390 transaction issues an ALLOCATE command for a session to an APPC terminal or connection that is owned by another CICS system.
4. The CICS-supplied transaction CRTE is used to invoke transactions in other systems. See “The routing transaction (CRTE)” on page 44.

Terminal-initiated transaction routing

When a terminal requests transaction initiation, CICS examines the installed transaction definition. If the transaction is defined with a remote system named in the REMOTESYSTEM option (or REMOTESYSID in CICS on Open Systems), CICS initiates the relay transaction, which passes control to the relay program to drive transaction routing.

Static transaction routing

Static transaction routing is a term used to distinguish standard transaction routing from dynamic transaction routing. In CICS/400, all transaction routing is static. In CICS on System/390 and CICS on Open Systems, static transaction routing occurs when DYNAMIC(NO) is specified in the transaction definition.

In static transaction routing, 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 transaction routing allows a user-written program to select the system to which a transaction routing request is to be directed. Dynamic transaction routing is provided by CICS on System/390, CICS on Open Systems, and CICS Transaction Server for Windows. The implementations are different (see following descriptions). CICS/400 does not support dynamic transaction routing.

CICS on System/390 and CICS on Open Systems: In a CICS on System/390 or CICS on Open Systems system, you can use the DYNAMIC option when defining a transaction. This includes a remote definition of a CICS Transaction Server for Windows or CICS/400 transaction.

Specifying DYNAMIC(YES) means that you want the opportunity to route the terminal data to an alternative transaction at the time the defined transaction is invoked. CICS enables this by allowing a user-supplied program, called the

dynamic transaction routing program, to examine the terminal input data and redirect it to any transaction and system it chooses. In CICS on System/390, this program has the default name of DFHDYP, but an alternative name may be defined by using the DTRPGM system initialization parameter.

Parameters are passed in a communications area between CICS and the routing program. The program may change some of these to influence subsequent CICS action. For example, in CICS on System/390, some of the parameters are:

- Invocation reason. The routing program can be reinvoked after an unsuccessful routing attempt or when the target transaction has terminated.
- Error information.
- The SYSID of the target system. Initially, the one specified on the REMOTESYSTEM option for the installed transaction (or REMOTESYSID in CICS on Open Systems).
- The name of the target transaction. Initially, the name specified on the REMOTENAME option for the installed transaction.
- A pointer to a data area containing the initial input data from the terminal.

The dynamic transaction routing program can issue most EXEC CICS commands.

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.

CICS Transaction Server for Windows: CICS Transaction Server for Windows provides a user exit in which a user program can change the target system of a transaction routing request. This exit can also change the target system for ***any*** shipped function request and provide a target system for a transaction not defined in the local system.

The exit, if defined, is called before processing any of the following:
- An undefined transaction code
- A transaction routing request
- A function shipping request
- A DPL request
- Any command with a SYSID option.

Shipping terminal definitions

Terminals defined as shippable do not need a definition in the transaction-owning system. If necessary to support transaction routing, the terminal-owning system sends the terminal definition to the transaction-owning system.

Automatic transaction initiation

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

An ATI request can cause the initiation of a transaction in a remote system (see Figure 7 on page 43).

ATI also allows a request for a transaction owned by a particular CICS system to name a terminal that is owned by another, connected system (see Figure 7). Although the original ATI request occurs in the application-owning system, it is sent

by CICS to the terminal-owning system, where it causes the relay transaction to be initiated in conjunction with the specified terminal.

The user transaction in the application-owning system is then initiated as described on page 40 for terminal-initiated transaction routing, with one important difference. The extra factor in this case is the AID (automatic initiate descriptor) associated with an ATI request. The AID specifies the names of the remote transaction and system, (**P1** and **S1** in Figure 7). In Figure 7, the terminal-owning system (**S2**) must find a transaction definition that specifies REMOTENAME (**P1**) and REMOTESYSTEM (**S1**) (or REMOTESYSID on CICS on Open Systems).

In a CICS on System/390 system, an alternative to REMOTESYSTEM is DYNAMIC(YES). In CICS/VSE and CICS Transaction Server for VSE/ESA, if DYNAMIC(YES) is coded, the dynamic routing program is *not* driven and the remote system name is taken from the AID. In CICS Transaction Server for OS/390 and CICS Transaction Server for z/OS, if DYNAMIC(YES) is coded, the dynamic routing program is driven, but the remote system name is still taken from the AID; the routing program can do other things—for example, update counts of requests routed to various remote systems.

In summary, if **S2** finds a transaction definition that specifies REMOTENAME (**P1**) and either REMOTESYSTEM (**S1**) or DYNAMIC(YES), the START command is routed back to **S1**; otherwise the START command is rejected.

ATI requests are queued:

- If the link to the terminal-owning system is not available, in the application-owning system.
- If the terminal is not available, in the terminal-owning system.

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

In the transaction-owning system, 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.

Figure 7 on page 43 shows an example of ATI involving three systems:
S0 The initiating system
S1 The transaction-owning system
S2 The terminal-owning system.

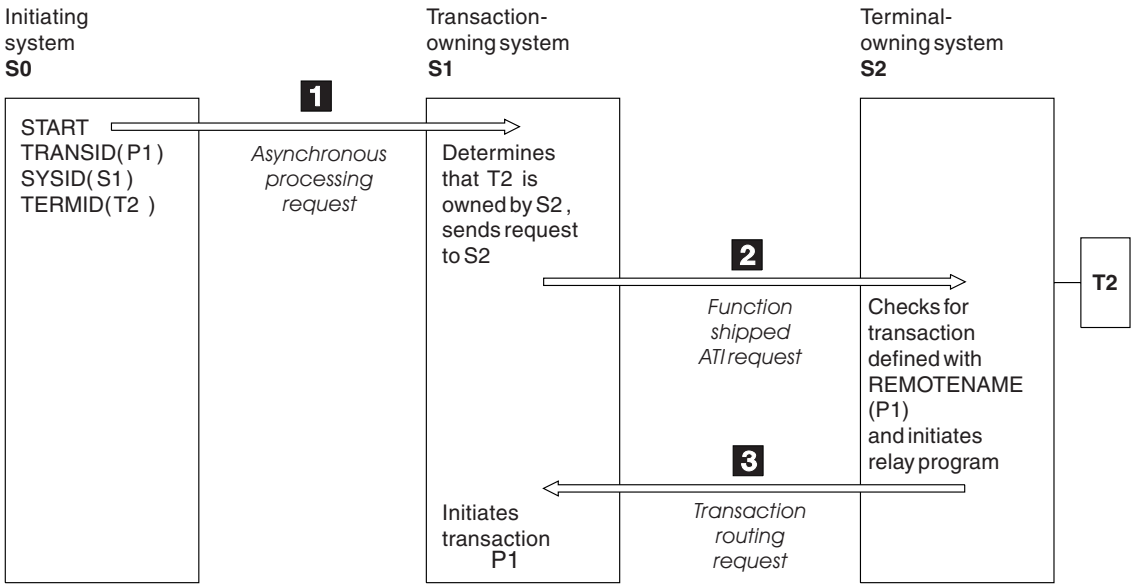


Figure 7. Automatic transaction initiation involving three CICS systems

Terminal definitions not shipped with ATI requests

As mentioned in "Shipping terminal definitions" on page 41, CICS ships a copy of the terminal definition to the transaction-owning system when necessary to support transaction routing. This definition is available for use with ATI requests received subsequently by the transaction-owning system.

However, terminal definitions are ***not*** shipped at the time an ATI request is received. If an ATI request names a terminal not already known in the transaction-owning system, the terminal-not-known condition occurs.

User exits for terminal-not-known condition: In CICS on System/390 systems only, two global user exits, XALTENF and XICTENF, help you to deal with the terminal-not-known condition.

The relay program

When a terminal operator enters a transaction code, and CICS determines that the transaction is in a remote system, a local relay transaction is attached to execute a CICS-supplied program known as the **relay program**. The **relay program** provides the communication mechanism between the terminal and the remote transaction. Note that in CICS on Open Systems, the relay is a function running in a shared library.

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

When the relay transaction is attached, it acquires an intersystem session and sends a request to the application-owning system, to cause the "real" user transaction to be started. In the application-owning system, the terminal is represented by a control block known as the **surrogate** TCTTE (in CICS on Open Systems, terminal surrogate). This TCTTE becomes the transaction's principal

facility, and is indistinguishable by the transaction from a “real” terminal entry. If the transaction issues a request to its principal facility, the request is shipped back to the relay transaction over the 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 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 system. 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 terminates and frees the terminal and the intersystem session.

Basic mapping support

CICS Transaction Server for Windows, CICS on Open Systems, and CICS/400 support only the minimum level of BMS. (They support SEND TEXT.)

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

When transaction routing with BMS, you should be aware of the limitations of BMS, and of the possible different levels of support in the application owning region and the terminal owning region when they are running on CICS systems on different platforms.

The routing transaction (CRTE)

The routing transaction (CRTE) is a CICS-provided transaction that enables a terminal operator to invoke transactions that are owned by a connected CICS system. It differs from normal transaction routing in that the remote transactions do not have to be defined in the local system. However, the terminal through which CRTE is invoked must be defined on the remote system (or defined as “shippable” in the local system), and the terminal operator needs security authorization if the remote system is protected. For the security support available, see the CICS Intercommunication manual for each product.

CRTE can be used from any 3270 display device.

To use CRTE, the terminal operator enters:

```
CRTE SYSID=xxxx
```

where *xxxx* is the local name of the remote system. 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.

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 sign-on transaction CESN; that is, the operator signs on to the remote system. The use of CRTE itself can also be restricted by security.

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 is an alternative to installing a definition of the remote CEMT in the local system. CRTE is also useful for testing remote transactions before final installation.

Chapter 9. Distributed program link

This chapter contains the following topics:

Introduction to DPL

When a CICS application program issues an EXEC CICS LINK command, control passes to a second program, named in the command. The second program executes and, when it completes, control returns to the first program at the instruction following the LINK command. The linked-to program can return data to the linking program in either of the following cases:

- The LINK command has used the COMMAREA option to pass the address of a communication area.
- The CICS region is CICS TS for z/OS Version 3.1 or later, and the programs exchange data by means of a "channel". (For information about channels, see the Application Programming Guide for your CICS System/390 product.)

Distributed program link (DPL) extends the use of the EXEC CICS LINK command so that the linking and linked-to programs can be in different CICS systems. The systems can be copies of the same CICS product or of different CICS products.

In certain circumstances, the use of DPL can improve performance by reducing the number of data flows between connected CICS systems.

You can specify that the linked program is to run on a connected CICS system by coding the SYSID option in the LINK command, or by specifying the remote system name in the local definition of the program.

In Figure 8, program A links to a program B, which is in a different CICS system. The arrowed line represents the flow of control. To program A, program B appears as a called subroutine.

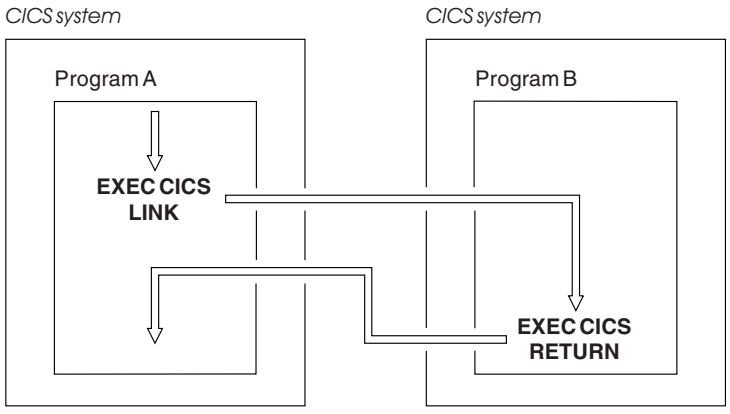


Figure 8. Distributed program link

Why use DPL?

The following are some reasons why you use might use DPL:

- To separate the end-user interface (for example, BMS screen handling) from the application business logic (for example, accessing and processing data). This makes it easier to port part of an application between systems; an example would be moving the end-user interface from a 370/390 system to a workstation.
- To obtain performance benefits from running programs closer to the resources they access, reducing the need for function shipping requests.
- Where applicable, to provide a simpler solution than distributed transaction programming (DTP).

Synchronization

DPL provides two ways to handle synchronization:

1. If you code SYNCONRETURN, the linked-to program commits its resource updates immediately before returning control to the linking program. There are separate units of work in the two communicating systems.

   The linked-to program may take one or more syncpoints during its execution. However, the response the linking program is given (which may be Normal or Rollback) corresponds to the outcome of the syncpoint taken by CICS on return from the linked-to program, and does not relate to the outcome of any of the syncpoints the linked-to program may have initiated.
2. If you do not code SYNCONRETURN, the linking program initiates commitment in both systems, either by issuing a SYNCPOINT command or implicitly at task end.

Data integrity considerations govern the decision whether or not to use the SYNCONRETURN option.

DL/I and SQL databases

DPL is an easy way for a CICS Transaction Server for Windows, CICS on Open Systems, or CICS/400 application program to access DL/I and SQL databases and BDAM files on a remote CICS system. The program simply links to an application program (in the data-owning system) that reads and updates the databases or files.

Another way to access this data is to use distributed transaction programming (DTP) as described in Chapter 11, “Distributed transaction programming,” on page 59. DTP is more difficult and therefore more costly to develop, but, if well designed, should be more efficient in use.

Restrictions when using DPL

There are a number of restrictions on the programs that you can link to using DPL.

You should not link to programs that issue syncpoints (unless SYNCONRETURN is coded in the LINK command). See “Taking syncpoints” on page 64.

Because there is no terminal involved, you should not link to programs that issue:

- Terminal control commands to the system in which the linking program is running
- Commands that inquire on terminal attributes (such as ASSIGN commands)
- BMS commands

- Batch data interchange commands
- Commands that address the TCTUA (programs can use the communication area to pass data).

For DPL between any combination of CICS products, the maximum recommended length of a communications area is 32500 bytes.

Abends when using DPL

If the linked-to program terminates abnormally, the mirror transaction returns the last abend code to the linking system. The abend code returned is the ***last*** abend to occur in the linked-to program, which may have handled other abends before terminating.

Chapter 10. Asynchronous processing

This chapter contains the following topics:

Introduction to asynchronous processing

Asynchronous processing is a way of distributing the processing of an application between connected systems. In contrast to distributed transaction programming, the processing is **asynchronous**.

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

In asynchronous processing, requests and replies are transmitted on different sessions. No processing dependency exists between a request and a reply, and no assumptions are made about the timing of the reply. The differences between synchronous and asynchronous processing are illustrated in Figure 9. The starting of TRAN4 can be time-dependent and can be delayed by scheduling constraints in System B.

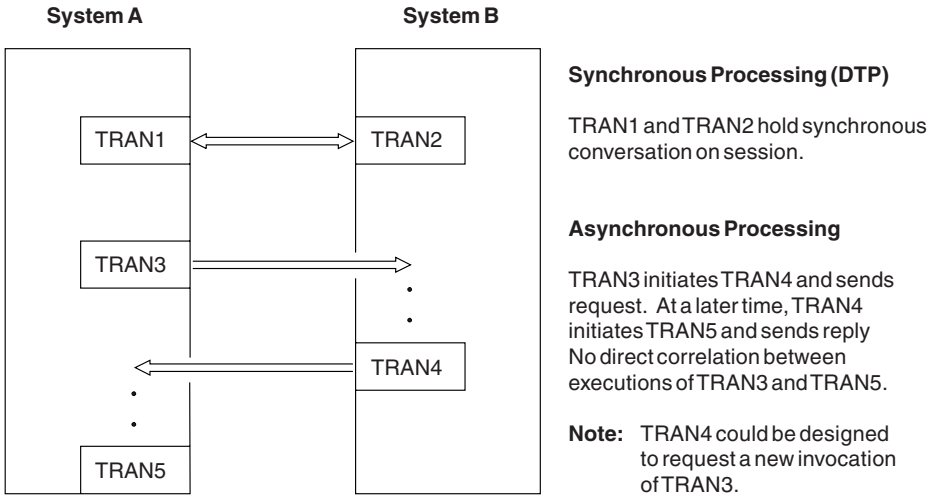


Figure 9. Synchronous and asynchronous processing compared

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 require synchronized changes to local and remote resources; for example, it cannot be used to process concurrent logically-related updates to data in different systems.

Note that, in Figure 9 on page 51, any changes made by the synchronous transactions TRAN1 and TRAN2 can be co-ordinated for recovery purposes; any changes made by the asynchronous transactions TRAN3, TRAN4, and TRAN5 cannot.

Example

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

Asynchronous processing methods

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

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

   You can use the START command to schedule a transaction in a remote system in much the same way as you would in a single CICS system. This type of asynchronous processing is essentially a form of CICS function shipping, and as such, is usually transparent to the application. CICS recognizes that a transaction is remote in one of two ways:

   - The transaction resource definition specifies that it is remote
   - The START command includes the SYSID option to specify a remote CICS system.

   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 initiated transaction.

   A CICS transaction can use the EXEC CICS ASSIGN STARTCODE command to determine how it was initiated.

   Asynchronous processing is more fully discussed under “Asynchronous processing using START/RETRIEVE commands” on page 53.

2. By using distributed transaction programming (DTP), a cross-system method with no single-system equivalent.

   When you use DTP to attach a remote transaction, you also allocate a session and start a conversation. This permits you to send data directly and, if you want, to receive data from the remote transaction. Your transaction design determines the format and volume of the data you exchange. For example, you can use repeated SEND commands to pass multi-record files. However short the conversation, during the time it is in progress, the processing is synchronous. Error recovery and syncpoint functions are available using the normal DTP commands.

   When you have exchanged data, you terminate the conversation and quit the local transaction, leaving the remote transaction to continue processing asynchronously.

   DTP can be used for CICS—non-CICS communication, as well as for CICS—CICS communication.

If data conversion is required, the DTP application must contain logic to handle this.

Distributed transaction programming is more fully discussed under Chapter 11, “Distributed transaction programming,” on page 59.

Asynchronous processing using START/RETRIEVE commands

The interval control commands that can be used for asynchronous processing are:
- START
- CANCEL
- RETRIEVE.

Starting and canceling remote transactions

The interval control START command is used to queue a transaction-initiation request in a remote CICS system. The command is effectively function shipped. In the remote system, the mirror transaction is invoked to issue the START command.

You can include time-control information on the shipped START command, using the INTERVAL or TIME option. Before a command is shipped, a TIME specification is converted by CICS to a time interval relative to the local clock. If the ends of a link are in different time zones, use the INTERVAL option.

The time interval specified in a START command is ***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, ***before the expiry of the time interval***, by shipping a CANCEL command to the same system. The START command to be canceled is uniquely identified by the REQID value specified on the START command and on the associated CANCEL command. Any task can issue the CANCEL command.

Passing information with the START command

The START command has a number of options that enable information to be made available to the remote transaction when it is started. If the remote transaction is in a CICS system, the information is obtained by using the 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.
- Additional user data can be located by using the QUEUE option, which could (for example) identify a temporary storage queue.
- A terminal name—specified in the TERMID option. This is the name of a terminal that is to be associated with the remote transaction when it is initiated. If a terminal is defined in the system that owns the remote transaction but is not owned by that system, it is acquired by transaction routing. This is illustrated in Figure 7 on page 43.

  In a CICS on System/390 system, the global user exits XICTENF and XALTENF can be coded to cover the case where the terminal is not defined in the application-owning system.
- The transaction name and terminal name to be used for replies—specified in the RTRANSID and RTERMID options. These options provide a 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 RTRANSID and RTERMID specified by the local transaction on the initial request.)

Passing an APPLID with the START command

If you have a transaction that can be started from several different systems, and that is required to issue a START command to the system that initiated it, you can arrange for each invoking transaction to send its local system APPLID as part of the user data in the START command. A transaction can obtain its local APPLID by using an ASSIGN APPLID command.

Improving performance of intersystem START requests

In some inquiry-only applications, sophisticated error-checking and recovery procedures may not be justified. When 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 data flows to and from the remote system can be substantially reduced by using the NOCHECK option of the START command. When the connection between the two systems is through VTAM, this can result in considerably improved performance. The trade-off is between performance and sophisticated recovery procedures.

A typical use for the START NOCHECK command is in the remote inquiry application described in “Example” on page 52.

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 10 on page 57.

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 for remote transactions” on page 55).

Including start request delivery in a logical unit of work

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

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

Deferred sending of START requests with NOCHECK option

For START commands with the NOCHECK option, CICS defers transmission of the request to the remote system.

START requests with NOCHECK are deferred until one of the following events occurs:

- The transaction issues a syncpoint.
- The transaction terminates with an implicit syncpoint.
- The transaction issues any other type of function shipping request—for example, a WRITE TS, or a START without the NOCHECK option.
- A sufficient number of START NOCHECK requests for the same remote system have accumulated on the local system to make transmission efficient.

There are exceptions to the above rules:

- In CICS on System/390 MRO intercommunication, START commands with NOCHECK are not deferred.
- In CICS on Open Systems, a START NOCHECK with the PROTECT option causes the buffer to be flushed.

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

Local queuing of START commands for remote transactions

When a local transaction is ready to ship a START command, the intersystem facilities may be unavailable, either because the remote system is not active or because a connection cannot be established. The normal CICS action in these circumstances is to raise the SYSIDERR condition. This can be avoided by using the NOCHECK option, and arranging for CICS to queue the request locally and forward it when the required link is in service.

If the required connection is out of service, CICS queues a START NOCHECK command for a remote transaction when two conditions are satisfied:

1. The SYSID option is not coded in the START command
2. The local definition of the transaction specifies LOCALQ(YES).

CICS on System/390 products support the XISLCLQ global user exit, which allows flexibility by enabling a user-written program to make a decision about local queueing, overriding the LOCALQ option.

Data retrieval by a started transaction

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

In accordance with the normal rules for interval control, CICS queues a start request for a transaction that carries both user data and a terminal identifier 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. Such an access automatically cancels the queued start request.

Thus, it is possible to design a transaction that can handle the data associated with multiple start requests. Typically, a long-running 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 could 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. Overall application design should ensure that a transaction cannot get into a permanent wait state due to the absence of further start requests—for example, the transaction can be defined with a timeout interval. (Note that this option is not supported by CICS on Open Systems.)

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

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 start request is a system, it selects an available session to that system and makes it the principal facility of the started transaction. If no session is available, the request is queued until there is one.

System programming considerations

Resources must be defined for asynchronous processing, as briefly indicated below. Detailed information on how to define the resources is given in the resource definition guide for your CICS product.

- 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 additional user data located by that option must be accessible from the system to which the START command is shipped.
- In a START command, the RTRANSID option specifies a transaction identifier that can be retrieved by the started transaction and used in a subsequent START command. The transaction named by the RTRANSID option must be defined in the remote system.

  For example, in Figure 9 on page 51, the START command in TRAN3 would include the options TRANSID(TRAN4) and RTRANSID(TRAN5). These options cause TRAN4 to be started, and indicate to TRAN4 that it in turn should start

TRAN5. This is the intended use of the RTRANSID option; its actual use is application-dependent. TRAN5 must be defined to System B.

Asynchronous processing example (with NOCHECK)

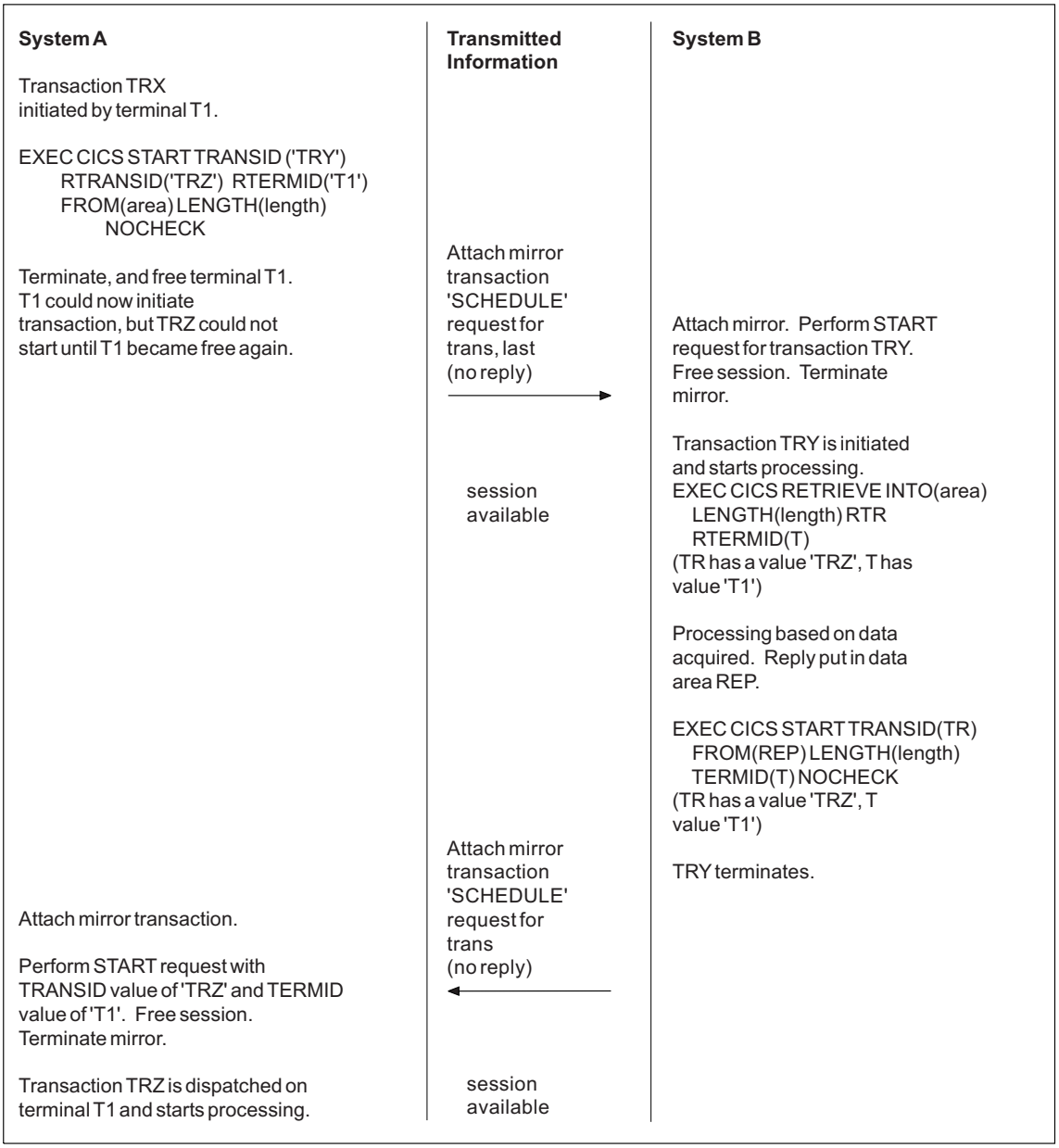


Figure 10. Asynchronous processing—remote transaction initiation using NOCHECK

Figure 10 shows an example of asynchronous processing using the NOCHECK option of the START command.

Chapter 11. Distributed transaction programming

When CICS arranges function shipping, asynchronous transaction processing, transaction routing, or distributed program link, it establishes a logical data link with a remote system. A data exchange between the two systems follows. CICS-supplied programs control this exchange, issuing commands to allocate conversations, and send and receive data between the systems.

CICS supplies equivalent commands to enable application programs to converse under their own control across intercommunication links. Using these commands, you can distribute the functions of a business transaction over several transaction programs within a network. This technique is called **distributed transaction programming** (DTP).

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

Why use distributed transaction programming?

Distributed transaction programming is needed because of the possible costs of other intercommunication functions (see “Limitations of function shipping”) and because of its own advantages (see “Advantages of distributed transaction programming” on page 60).

Limitations of function shipping

Function shipping gives you access to remote resources, and transaction routing lets a terminal communicate with remote transactions. From a functional point of view, these two facilities are probably sufficient for most intercommunication needs. However, design criteria go beyond pure function. Machine loading, response time, continuity of service, line traffic, and economic use of all resources are factors that affect transaction design.

The following two examples describe cases where function shipping is an obvious but not ideal solution.

Example 1

You are browsing a remote file to select a record that satisfies some criteria.

Solution 1: Use function shipping. CICS ships each GETNEXT request across the link, and the mirror reads the record and ships it back to the requester.

There are two network data flows per record; the data flow can be quite significant. For a browse on a large file, the overhead can be unacceptably high.

Solution 2: Use distributed program link. CICS links to a program that is running in the system that owns the file.

Solution 3: Use a DTP conversation. The local transaction sends the selection criteria. The remote transaction returns the keys and relevant fields from the selected records. This drastically reduces both the number of flows and the amount of data sent over the link.

Example 2

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 to be sorted for the separate distribution centers, and transmitted to them to enable reordering and distribution.

Solution 1: Use function shipping to write each reorder record to a remote file as it arises. This method is simple, but has several drawbacks:

- Data is transmitted to the remote systems irregularly in small packets. This means inefficient use of busy 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 causes a catastrophic suspension of operations at a branch.
- Intensive intercommunications activity (for example, at peak periods) causes reduction in performance at the point-of-sales terminals.

Solution 2: Each sales transaction writes its reorder records to a transient data queue, and continues its conversation with the terminal.

Restocking requests are not urgent, so sorting and sending the data is delayed until an off-peak period. Alternatively, the transient data queue can 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.

The sender transaction can use function shipping to transmit the reorder records. After the sort process, each record is written to a remote file in the relevant remote system.

This method is not ideal either. The sender transaction must wait after writing each record to make sure that it gets the right response. Apart from using the link inefficiently, waiting between records makes the whole process very slow.

Solution 3: Using distributed transaction programming, a transaction in the branch:

1. Sorts the reorder records and creates a file for each distribution center.
2. Sends each file to a partner transaction at the appropriate distribution center.

Each distribution center then processes the reorder records like any other local file. This solution is a much more efficient use of the link.

Advantages of distributed transaction programming

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 places the processing associated with a resource in the resource-owning region.

DTP (like asynchronous processing and distributed program link) lets you process data at the point where it arises, instead of overworking network resources by assembling it at a central processing point. However, DTP is much more flexible than either asynchronous processing or DPL. For example, it:

- Can be used to communicate with both CICS and non-CICS systems.

- Enables synchronous communication and data transfer between applications running on different systems.
- Can provide a common interface to transactions owned by different systems.
- Allows some measure of parallel processing to shorten response times.
- Provides a buffer between a security-sensitive file or database and applications, so that no application need know the format of the file records.
- Enables batching of less urgent data destined for a remote system.

Conversations

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 11 on page 62 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 running SUBR.

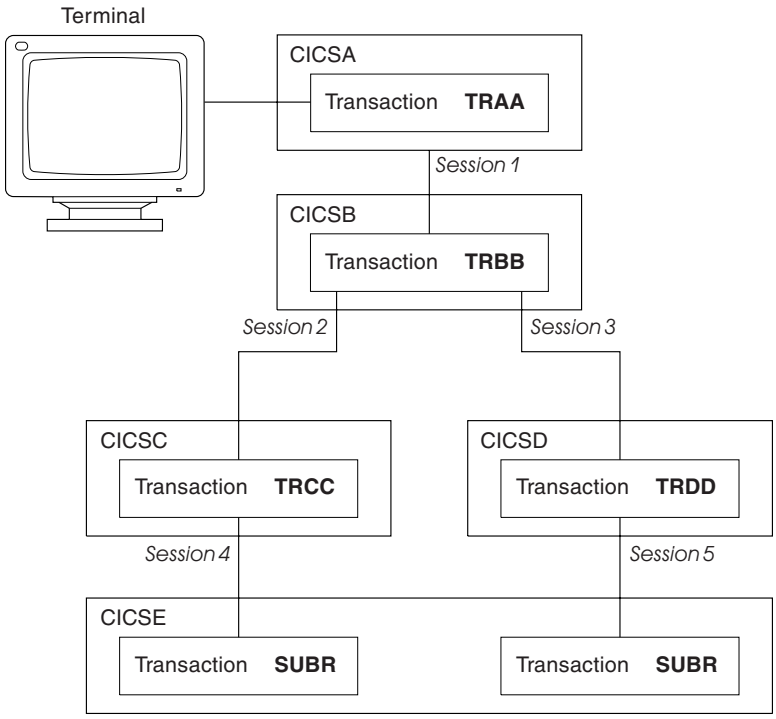


Figure 11. DTP in a multisystem configuration

Structure of a distributed process

The structure of a distributed process is determined dynamically; it cannot be specified beforehand in transaction definitions. For each 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 one transaction to activate another transaction is called the **alternate facility** of the allocating transaction. In Figure 11, session 1 is the principal facility of transaction TRBB and an alternate facility of transaction TRAA. A transaction has only one principal facility, but can have any number of alternate facilities. Transaction TRBB has two alternate facilities, sessions 2 and 3.

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. In Figure 11, transaction TRBB is the front end of the conversations on sessions 2 and 3, and the back end of the conversation on session 1. (Some publications 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.

Application design

DTP has none of the transparency of function shipping or transaction routing. A conversation transfers data from one transaction to another. For this to function properly, each transaction must know what the other intends. It is therefore necessary to design, code, and test front end and back end as one software unit. The same applies when there are several conversations and several transaction programs. Each new conversation adds to the complexity of the overall design.

In “Example 2” on page 60, the DTP solution (Solution 3) is to transfer a file of data from one transaction to another–in this case, transmit the entire contents of the transient data queue from the front end to the back end. 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.

Among other things, the designer of a DTP application must decide:
- Which syncpoint-level to use for conversations
- If data conversion is necessary, which partner in the conversation should handle it

Control flows

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. Then they are added to the header.

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**.

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 be in a state to receive more data.

Either end of the conversation can cause a change of state, usually by issuing a particular command from a particular state. CICS tracks these changes, and stops a transaction from issuing a command that is wrong for its current state.

Synchronization

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

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

Example

A transaction is transmitting a queue of data to another system to be written to a file. 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 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. The sending system must restore the queue entries that have been sent, and the receiving system must delete any entries made in the file. CICS helps you to do this (see “Taking syncpoints”).

Rollback and backout

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**. For the most part, the two terms are used interchangeably. The condition that exists as long as there is no loss of consistency between distributed data resources is called **data integrity**.

Application-initiated rollback: There are cases where you want to recover resources, even though there are no error conditions detectable by CICS. 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 can be programmed to respond by restoring the data resources to the state they were in at the start of the order. At synchronization level 2 (see “Synchronization levels” on page 6), rollback occurs automatically in all remote partner transactions.

Taking syncpoints

If you log your own data movements, you can arrange backout of your files and queues. However, this involves very complex programming. To save you the trouble, CICS arranges resource recovery for you.

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

Taking a syncpoint, if successful, **commits** all changes to 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.

An unsuccessful syncpoint causes rollback. Recoverable resources are restored to their state at the start of the UOW.

CICS can commit and back out changes to resources, but the service has a performance trade-off. Some transactions do not need such facilities. If the recovery of resources is not a problem, use simpler methods of synchronization.

The three synchronization levels

The APPC architecture defines three levels of synchronization:

Level 0 – NONE
Level 1 – CONFIRM
Level 2 – SYNCPOINT

At synchronization 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 synchronization level 1, you can use specific 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 synchronization 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 synchronization 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 current synchronization level 2 conversation within the transaction.

A transaction specifies the required synchronization level in the CONNECT PROCESS command that initiates a conversation. The requested level must not be higher than that supported between the two products. Support for the different synchronization levels varies between products. Refer to “CICS product communication support” on page 7.

EXEC CICS or CPI Communications?

Some CICS products give you a choice of two application programming interfaces (APIs) for coding your DTP conversations on APPC sessions. The first, the **CICS API**, is the end-user 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 for Communications** (CPI Communications) is the communications interface defined by Systems Application Architecture® (SAA). It consists of a set of defined verbs, in the form of program calls, which are adapted for the language being used.

Table 10 on page 66 compares the two methods to help you to decide which API to use for a particular application.

Table 10. CICS API compared with CPI Communications

CICS API	CPI Communications
Portability between different members of the CICS family.	Portability between systems that support SAA.
Synchronization levels 0, 1, and 2 supported.	Synchronization levels 0, 1, and 2 supported, *except for transaction routing, for which only synchronization levels 0 and 1 are supported*.
Program initialization parameter (PIP) data supported (CICS on System/390 and CICS on Open Systems only).	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.
Mapped conversations (see note 3) can be programmed in any of the languages supported by CICS.	Mapped conversations can be programmed in any of the languages supported by CICS.
Basic conversations (see note 3) can be programmed only in assembler language or C, and only on a CICS on System/390 system.	Basic conversations can be programmed in any of the languages supported by CICS, but only on a CICS on System/390 system.

Additional notes on the two APIs

1. You can mix CPI Communications calls and EXEC CICS commands in the same transaction, but not on the same side of the same conversation. ***In other words, each half-session can use only one application interface.***
2. One partner in a conversation can use CPI Communications calls while the other uses the CICS API. ***In other words, the half-sessions at either end of the same conversation can use different application interfaces.***

   To correctly coordinate a conversation that is using a different API in each half-session, the programmers must know the details of how both APIs map to the APPC architecture.
3. Both interfaces, CICS API and CPI Communications, support APPC **mapped conversations**, in which the systems provide and interpret protocol headers, and the application programs deal only with user data. In an APPC **basic conversation**, the sending application must prefix the data with the header required by the communications protocol. The receiving application must interpret this header.
4. CICS/VSE 2.3 does not support the CPI Communications API.

Part 3. Appendixes

Bibliography

This section lists those books in the System/390 and non-System/390 CICS libraries that are related to intercommunication.

Note: To help you find the information you need, some books are listed in more than one category.

CICS Family intercommunication books

CICS Family: Communicating from CICS on System/390, SC34-6474
CICS Family: Interproduct Communication, SC34-6473

CICS on System/390 intercommunication books

CICS Transaction Server for z/OS Version 3 Release 1

CICS Distributed Transaction Programming Guide, SC34-6438-00
CICS External Interfaces Guide, SC34-6449-00
CICS Front End Programming Interface User's Guide, SC34-6436-00
CICS Intercommunication Guide, SC34-6448-00
CICS Internet Guide, SC34-6450-00

CICS Transaction Server for z/OS Version 2 Release 3

CICS Distributed Transaction Programming Guide, SC34-6236-00
CICS External Interfaces Guide, SC34-6244-00
CICS Front End Programming Interface User's Guide, SC34-6234-00
CICS Intercommunication Guide, SC34-6243-00
CICS Internet Guide, SC34-6245-00

CICS Transaction Server for z/OS Version 2 Release 2

CICS Distributed Transaction Programming Guide, SC34-5998-00
CICS External Interfaces Guide, SC34-6006-00
CICS Front End Programming Interface User's Guide, SC34-5996-00
CICS Intercommunication Guide, SC34-6005-00
CICS Internet Guide, SC34-6007-00

CICS Transaction Server for OS/390 Release 3

CICS Distributed Transaction Programming Guide, SC33-1691-02
CICS External Interfaces Guide, SC33-1944-01
CICS Front End Programming Interface User's Guide, SC33-1692-02
CICS Intercommunication Guide, SC33-1695-02
CICS Internet Guide, SC34-5445-00

CICS Transaction Server for VSE/ESA Release 1.1.1

Distributed Transaction Programming Guide, SC33-1661
External CICS Interface, SC33-1669
Front End Programming Interface User's Guide, SC33-1662
Intercommunication Guide, SC33-1665

CICS/VSE Version 2

Distributed Transaction Programming Guide, SC33-0898
Intercommunication Guide, SC33-0701
Server Support for CICS Clients, SC33-1712

CICS non-System/390 intercommunication books

CICS TS for Windows, Intercommunication, SC34-6209
CICS on Open Systems Intercommunication Guide, SC33-1564
CICS/400 Intercommunication, SC33-1388

CICS Transaction Gateway and CICS Universal Clients

CICS Transaction Gateway: Programming Guide, SC34-6141
CICS Transaction Gateway: Programming Reference, SC34-6140
CICS/VSE Version 2 Release 3 Server Support for CICS Clients, SC33-1712

Non-CICS books

SNA books

Systems Network Architecture Technical Overview, GC30-3073

Systems Network Architecture Transaction Programmer's Reference Manual for LU Type 6.2, GC30-3084

Systems Network Architecture--Sessions Between Logical Units, GC20-1868

Systems Network Architecture Format and Protocol Reference Manual: Architecture Logic for LU Type 6.2, SC30-3269

Systems Network Architecture LU 6.2 Reference–Peer Protocols, SC31-6808

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.

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. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at www.ibm.com/legal/copytrade.shtml.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

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

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

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

Index

E

F

H

I

L

M

N

P

R

S

T

U

W

X

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To ask questions, make comments about the functions of IBM products or systems, or to request additional publications, contact your IBM representative or your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  IBM United Kingdom Limited
  User Technologies Department (MP095)
  Hursley Park
  Winchester
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

SC34-6473-02

Spine information:

CICS Family

Interproduct Communication